Congestion Management in Lossless Interconnection Networks

Ernst Gunnar Gran

Doctoral Dissertation

Submitted to
the Faculty of Mathematics and Natural Sciences
at the University of Oslo in partial fulfillment
of the requirements for the degree
Philosophiae Doctor

September 2013

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1462

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AIT Oslo AS.

Produced in co-operation with Akademika Publishing.
The thesis is produced by Akademika Publishing merely in connection with the thesis defence. Kindly direct all inquiries regarding the thesis to the copyright holder or the unit which grants the doctorate.

Abstract

In supercomputers and modern data center clusters lossless interconnection networks are frequently used to achieve high throughput and low latency. It has been known for three decades however, that congestion and congestion spreading in such networks can lead to severe performance degradation if no countermeasure is taken. Nevertheless, for a long time, the challenges related to network-wide congestion in lossless interconnection networks received little attention. A combination of tuning and tailoring of network characteristics for a given application, together with overprovisioning of network resources, kept congestion and congestion spreading from occurring in practice. To be able to dynamically manage congestion was then not really needed.

During the last decade, however, we have seen a renewed interest in congestion management for lossless interconnection networks. The use of virtualization together with an increased focus on cost-efficient green computing have spawned a desire to operate networks with dynamic and unpredictable traffic patterns closer to saturation. As such, proper congestion management is needed. In this thesis, we study congestion management in lossless interconnection networks in general, while giving special attention to the congestion control mechanism specified for InfiniBand, currently one of the most popular interconnection network standards. The contributions of the thesis include guidelines on how to implement congestion detection in switches facilitating injection throttling at the source nodes to avoid unfair treatment of contributors to congestion; an exploration of the rich InfiniBand congestion control parameter space and the corresponding influence on the performance of the congestion control mechanism; a study of the scope of an injection throttling based congestion management mechanism, like the one specified for InfiniBand; an abstract classification scheme for congestion trees of varying degree of dynamics; and finally, two novel congestion management mechanisms for input buffered switches and switches utilizing virtual output queuing, respectively, to overcome the weaknesses of current congestion management mechanisms based on injection throttling or hot-flow dynamic isolation.

Acknowledgements

First and foremost I would like to thank my three supervisors Olav Lysne, Tor Skeie and Sven-Arne Reinemo for their support, guidance and inspiration throughout the work related to this thesis, and for the numerous fruitful discussions we have had. In addition, I would in particular like to thank Olav for always believing in me and for giving me the opportunity to pursue a Ph.D. at Simula Research Laboratory; Tor for his never ending, encouraging enthusiasm for my work; and Sven-Arne for his precious and conscientious day-to-day supervision.

Next, I would like to thank my coauthors of the papers constituting the scientific work of this thesis. In addition to my supervisors, this includes Magne Eimot, Lars Paul Huse, Gilad Shainer, Eitan Zahavi, and, last but not least, my Spanish collaborators and friends José Duato, Jesús Escudero-Sahuquillo, José Flich, Pedro J. García, and Francisco J. Quiles. Furthermore, I thank the HPC Advisory Council, Mellanox Technologies and the Oracle division in Norway (previously part of Sun Microsystems) for their support.

Many thanks are also due to my former and current coworkers at Simula Research Laboratory, and in particular the people at the Network Systems department, for a pleasant and inspiring working environment. It is dangerous to start mentioning names, and certainly a bit unfair, but still I feel the need to acknowledge three persons in particular: A huge thank you goes to Wei Lin Guay for being a one of a kind fellow Ph.D. student. His blissful optimism and humble personality turned the late evenings and nights we shared working hard to meet common deadlines into memorable moments, and the trip we made to Los Angeles and the CCGrid conference will forever stay in my book of favorite memories. A special thank you also goes to Ahmed Elmokashfi for the numerous discussions we have had on subjects totally irrelevant to our work, and for a countless number of friendly hugs. Thomas Dreibholz, I take off my hat to the dedication he shows in every aspect of his work, to his admirable work capacity, and to the level of detail of which he reviewed this thesis, a thesis written on a subject outside of his field of expertise.

Finally, I would like to thank my whole family for the endless support they have shown during my years as a Ph.D. student. In particular, I would like to thank my parents, Maja and Lasse, for their warm and unconditional love; My extraordinary sisters, Kari, Trine and Marte, for cheering me up and carrying me forward; My beloved wife, Gunn Marit, for showing patience out of this world; and last, but maybe most of all, my two dearly loved sons, Isak and Jakob, for constantly reminding me of God's grace. You are the joy of my life!

Contents

Prologue

Here I am, stuck, in my car, on the highway, in a traffic jam. I'm surrounded by cars without knowing exactly why we're not moving at all. Although it doesn't actually matter, I can't help but start guessing. What's going on further down the road, causing this traffic jam? Are they doing some construction work on the road? Has there been an accident? Maybe we're just talking about an overloaded road or an inefficient intersection? There is no news on the radio. They did mention something about a large pop concert earlier today, though, didn't they? Well, anyway, I'm here, I'm stuck, and I'll have to wait like everyone else for my fair share of the resource we're all requesting, the road.

One thing strikes me, though, as I'm waiting. No matter the cause of this traffic jam, I know for sure that I will never request the resources at the location actually originating the problem. I'll never pass the spot down the road being the root of the congestion. How can I be sure about this? Well, the thing is, I can actually see my exit road just a few hundred meters away, and it's free! There are no cars over there, but it doesn't really matter, does it? I'll still have to wait as I'm blocked by these other cars. I'm trapped. This fact makes my situation less comfortable. The cars surrounding me, blocking my path, are doing so even though they are actually waiting for a resource I do not want. How annoying! I am an innocent victim of the congestion! In addition, being blocked, I block others, and by that, I add to the problem! I've ended up contributing to a congestion originally created by others, and the net result is what seems to be an unneeded waste of resources; time, money, fuel, the road, the environment. The performance of the road has gone belly up. Can't we do better than this? Can't we find a better way of utilizing the roads, beneficial for all, a way where cars requesting a given resource are not being blocked by cars waiting for other, distinct, resources?

Chapter 1

Introduction

Data networks come in all shapes and sizes. They have truly become ubiquitous, ranging in size from the on-chip network inside the CPU of a quad core mobile phone resting in the palm of your hand, to our precious Internet connecting autonomous networks all over the world into a single global arena of digital communication. The networks are everywhere. The characteristics of different types of networks, however, vary vastly, depending on the area of application, where the disruptive nature of a mobile ad-hoc network stands in stark contrast to the lossless high performance interconnection network we find in a supercomputer or in a modern data center. The latter type of network is the subject of this thesis, and in particular the management of traffic in such a network during "rush hours" to avoid "traffic jams".

1.1 Motivation

Strictly speaking, the term *interconnection network* may refer to any infrastructure that allows distributed digital units to communicate [1, 2]. In this thesis, however, the term *interconnection network*, or just *network*, should refer to the type of network that typically serves as the shared communication infrastructure in a High Performance Computing (HPC) supercomputer or a modern data center (e.g. facilitating cloud computing). To achieve the high throughput and low latency required by a HPC or data center application, it is beneficial to apply a *lossless* interconnection network. Contrary to a *lossy* network, where forwarding nodes inside the network are allowed to drop data if they receive data faster than what they are able to forward, a lossless network is not allowed to drop data during regular operation. Dropping data is considered to be too expensive and inefficient, both in terms of the wasted resources use by data that is later dropped, as well as the potential addition in communication latency experienced by an application as data has to be retransmitted. The lossless nature of an interconnection network, however, comes at a cost. While it potentially improves the performance of the network by increasing the utilization of the shared network resources, it comes with its own set of challenges and pitfalls.

Lossless behavior in an interconnection network is typically achieved using a *link-level flow control mechanism*. By means of the flow control mechanism, a forwarding node will

make sure not to forward any data to another node unless this other node has buffer space available to receive it. While this simple idea ensures that no data will be lost due to buffer overflow inside the network, the mechanism at the same time allows for saturation at one forwarding node to spread to others, and by that, congest a part of the network. As a node runs out of buffer space, other surrounding nodes will not be allowed to forward traffic to it. This could obviously have the effect that the buffers inside these surrounding nodes also start to fill up, and, as they become full, further block other surrounding nodes again. This *backpressure effect* of the flow control mechanism leads to *congestion spreading* in the network, where a growing tree of buffer occupancies blocks an increasingly larger part of the network as the branches of the tree stretch towards the data sources contributing to congestion. The phenomenon is the same as the one most of us unfortunately have experienced out in the streets, driving a car. Due to an accident, construction work, or maybe just a narrow road or an intersection being a bottleneck during rush hours, traffic starts to pile up in one area of the road, and then quickly spreads to other parts, blocking an increasing numbers of cars as the phenomenon develops. A traffic jam is created. To keep the backpressure effect of the flow control mechanism in a lossless interconnection network from creating a similar traffic jam in our network, a traffic jam that hampers overall network performance, *congestion management*[1] is needed.

Previous Congestion Management

The negative consequences of congestion and congestion spreading in lossless interconnection networks was first identified and expressed by Pfister and Norton in 1985 [3]. They showed that the network performance might collapse in such a network if no countermeasure is taken against congestion. While this discovery immediately spawned quite some efforts to solve the challenges of congestion spreading in interconnection networks, the efforts soon died out due to the fact that congestion seemed to be less of a problem in practice. The reason was twofold: Interconnection networks were usually over-provisioned, or they were configured and tailored for the specific application the network was to support at a given time. Under such circumstances, any presence of congestion was considered to be a system bug rather than a general interconnection network design challenge. As a consequence, congestion management during the nineteen nineties and the beginning of the new millennium primarily consisted of congestion avoidance through *over-provisioning* and *network tailoring*. The world is changing, however, and these two congestion avoidance techniques are, as elaborated on in the following sections, about to become obsolete.

The major problem with over-provisioning is the lack of scalability. As Moore's law [4] is being challenged by the laws of nature, further growth in compute performance has to be achieved through added parallelism. Note then, that as the number of compute nodes is increased to add parallelism, a correspondingly decreasing fraction of each node's traffic is needed to create congestion in the network. Add to this the fact that the number of CPUs, cores and threads running at each node increases concurrently, and it becomes clear that the degree of over-provisioning has to increase together with the

[1] In the literature the terms *congestion control* and *congestion management* are used interchangeably.

added parallelism to ensure that congestion does not occur. Even though the advances in processor, memory and I/O technologies are held back by the laws of nature, so is the case for the interconnection network technology itself, where the speed of light is the most obvious limiting factor when moving data in the network. Furthermore, the exchange of larger and more sophisticated data structures like XML-based messages, or the transfer of complete virtual disk images between nodes in the network, increases the bandwidth requirements even further. All together, the increasing bandwidth requirement makes over-provisioning of the network an unsuited congestion management mechanism for current and future interconnection networks.

For network tailoring to work as a congestion avoidance technique, the requirements of the running application or algorithm, and the corresponding traffic pattern produced, needs to be known in advance. Then, the network administrator may alleviate the negative consequences of congestion (if possible) by doing efficient load balancing of traffic in the network. While information about the traffic pattern previously was readily available as a machine or cluster typically was running only a single well-known application or algorithm at a time, virtualization changes this dramatically. Using virtualization, a machine or cluster typically runs a dynamic set of concurrent applications and algorithms. The traffic pattern in the network at any given time then becomes an overlay of several applications' and algorithms' patterns, depending on what applications and algorithms that are currently active, and where in the network they run. In addition, the virtualization software, and the administration of it, may itself use the network. All in all, it becomes extremely hard, if at all possible, to predict the traffic pattern while utilizing virtualization, and by that, network tailoring becomes invalid as a congestion avoidance technique.

Economically Efficient Green Computing

The environmental challenges facing the world, together with the economic slowdown influencing especially the western society, have lead to an increased focus on economically efficient green computing. A necessity in this respect is to be able to fully utilize both the compute nodes and the interconnection network in an HPC installation or modern data center. Better utilization of the compute (or I/O) nodes can be achieved through virtualization as it allows applications and algorithms to share common hardware in an efficient and safe way, while better utilization of the interconnection network means running the network closer to saturation. Notice that this directly contradicts the two before mentioned congestion avoidance techniques, over-provisioning and network tailoring. Over-provisioning the network not only increases the explicit costs of additional hardware, or cost of hardware with greater capacity, to provide the over-provisioning itself, but it is also likely to increase hidden costs in terms of additional space for hardware, supplementary cooling needed, greater power consumption, and so on. On the other hand, the dynamic and unpredictable traffic patterns resulting from virtualization, hampers the possibility to do network tailoring. To be able to realize economically efficient green computing in HPC installations and modern data centers, proper congestion management in the interconnection network is needed.

Research Challenges and Problem Statements

Due to the focus on economically efficient green computing, the last decade has slowly rekindled an interest for congestion management in lossless interconnection networks – both in the research community as well as in the industry. Contributions resulting from this increased interest will be further described and elaborated on in chapter 2.3, containing a review of congestion management in general. One major industrial event, however, was the extension of the InfiniBand [5] specification to include an appendix detailing support for congestion management, or *congestion control* as it is named in the context of InfiniBand.

InfiniBand (IB) is currently one of the most popular interconnection network standards in the world. Referring to the top 500 list [6], a list being updated every half a year containing the most powerful HPC supercomputers in the world, 52% of the top 100 supercomputers are based on IB technology (November 2012)[2]. The congestion control (CC) mechanism of IB is based on a closed-loop feedback control system, where a forwarding node detecting congestion inside the network is responsible for informing all nodes injecting traffic contributing to the congestion about the situation. These contributors to congestion are then expected to lower their injection rate of the corresponding traffic flows to remove the congestion. It is important, though, that this injection throttling is just sufficient to remove the congestion, and not too aggressive as it could leave the resources at the root of the congestion idle. If these resources, originally causing the congestion, are left idle, the result is an underutilization of highly sought-after network resources, which again could have a dramatic negative impact on overall network performance.

The exact behavior of the IB CC mechanism depends on the values of a set of CC parameters. These parameters determine characteristics like how aggressive the congestion detection should be, the rate of feedback from the forwarding node detecting congestion to the contributors of congestion, and then by how much and for how long the contributors should lower their corresponding injection rates. While the set of CC parameters adds flexibility to the IB CC mechanism, the IB CC specification gives no or little guidance on how to set the parameter values for the CC mechanisms to be efficient. Therefore, knowledge is needed on how to use the parameters, what effect the individual parameters have on network performance, and how they relate to each other. Furthermore, the IB CC specification leaves a lot of freedom to the hardware designer and manufacturer when it comes to exactly how congestion should be detected at a forwarding node. It is important that care is taken to make sure all contributors to congestion are treated in a fair manner, i.e. that the throttling initiated by the CC mechanism ensures that the contributors all get their fair share of the scarce resources at the root of the congestion. We are now ready to state our first two research questions (RQs):

> RQ1: *How can the IB CC parameters be tuned to ensure efficient behavior of the IB CC mechanism for a given scenario, and how do different parameters impact the network performance?*

[2]The second place is held by Cray Interconnect [7], accounting for 10% of the top 100 supercomputers, while 29% of the installations are based on custom-made interconnection networks.

RQ2: *How should congestion detection and notification be carried out at a forwarding node implementing an injection throttling based congestion management mechanism, like IB CC, to ensure fair treatment of the contributors to congestion?*

Any injection throttling based congestion management mechanism has a major challenge: No matter how efficient and clever the congestion detection at a forwarding node is, it is an indisputable fact that it will take time from congestion is detected until the contributors of congestion have been notified about the situation. Then, even more time will pass by before the effect of any changes in injection rates at the contributors reaches the location in the network originating the problem, the current *hotspot*. Even if we were able to bypass the laws of nature, and instantly notify the contributors of congestion, there are likely to be flows of traffic already on the way between the contributors to congestion and the hotspot. As such, an injection throttling based congestion management mechanism is bound to operate behind schedule, basing the injection rate adjustments on data reflecting a previous state of the network. This fact has spawned questions and doubts related to the possible success of any injection throttling based congestion management technique in general, and the IB CC mechanism in particular. This challenge gives raise to our third research question:

RQ3: *What is the scope of an injection throttling based congestion management mechanism like IB CC, and how does such a mechanism behave depending on traffic dynamics in the network and the lifetime of the hotspots, and in the case of IB CC, how robust are the IB CC parameters with respect to such changing traffic dynamics?*

To overcome the limitations imposed on reaction time by the closed-loop feedback control system of an injection throttling based congestion management mechanism, action needs to be taken immediately and locally at the root of congestion as soon the congestion is detected. This local action at a forwarding node needs to make sure any negative effects caused by the congestion are neutralized while the forwarding node waits for the throttling based mechanism to take effect. As such, the local and immediate action could be of a temporary nature. Our fourth research question seeks to cover the general limitation of injection throttling based congestion management:

RQ4: *How can we extend a throttling based congestion management mechanism, like IB CC, to overcome the challenges imposed by the feedback control loop by taking immediate local action at a forwarding node, while we wait for the throttling mechanism to have effect?*

These four research questions, *RQ1* to *RQ4*, will be addressed throughout this thesis. To answer the questions, the conducted research has included analyses of IB CC performance measurements from carefully designed network experiments, in-depth analyses of congestion management and IB CC based on network simulations, and finally the design and implementation of new and improved congestion management mechanisms, evaluated using network simulations. The two research methods *experiments* and *simulations* will be further described in the next section.

1.2 Research Methods

In the field of network research, when studying for example the characteristics of a network phenomenon, comparing different network configurations, or evaluating a new network technology, four research methods are commonly being used: *empirical measurement*, *analytical modeling*, *experiments*, and *simulations* [8, 9]. Which method, or combination of methods, that is applicable for a given scenario depends on the research questions being addressed and the availability of resources (e.g. existence of hardware with the required functionality). The following subsections will give a brief introduction to the four research methods in the context of network research, and give reasons for our choice of methods; experiment and simulation.

The idea behind *empirical measurement* is to gain knowledge about an existing network through observation, without affecting the network or controlling its behavior. As such, this research method is well suited to study and explore phenomena in a live network like the Internet. While it might be challenging to collect the desired information in an accurate way in a live system, and care needs to be taken to correctly analyze the collected data, the validity of the underlying observed system itself is unquestionable. The researcher is observing "the real world", and not e.g. a simplified abstract or theoretical model of it. Empirical measurement, however, is an option only if a network suitable to address the research questions at hand is available. In particular, this is generally not the case when the aim of the research is to reveal and validate new design principles (*RQ2*) or develop and evaluate a new network technology like a novel congestion management mechanism (*RQ4*). In addition, empirical measurement implies little or no control over the network and its characteristics. When it is desirable to control a network while studying it, e.g. to study the performance of an IB network as a function of IB CC parameter values (*RQ1*) or the performance of an injection throttling based congestion management mechanism as the traffic dynamics increases (*RQ3*), experiment is a more suitable research method.

The first step in *analytical modeling* is to map the subject of research to a mathematical or statistical model. Then, the model is analyzed using logical reasoning and mathematical tools to acquire new knowledge about the subject. In network research, analytical modeling provides a powerful tool in situations where available methods from network calculus [10], queuing theory [11] and control theory [12] can accurately capture the relevant aspects of a network. The complexity of the analytics increases rapidly, however, as the model of the network becomes more advance and detailed. To overcome the growth in complexity, a simplification of the model, including the introduction of new assumptions about the underlying network, could be needed to make the corresponding analysis manageable. Such a devaluation of the analytical model will challenge the usefulness of the analytical results, though, and by that influence the applicability of the analytical model itself. To address the research questions of this thesis, the level of detail required to capture all relevant aspects of congestion management in a network was considered to make an analytical model too complex to be manageable using the available set of analytical tools. As such, analytical modeling is not a viable option for our studies.

An *experiment* is an empirical measurement carried out in a controlled environment.

By controlling the environment, it becomes possible to repeat an experiment while changing only a subset of the experiment's parameters, e.g. the parameters being subject for research, and monitor the corresponding effect of the parameter change. In our case, this translates into running experiments in a physical network to study the change in network performance and network behavior as network characteristics and network parameters, like IB CC parameter values, are changed. As such, experiments are well suited to address both *RQ1* and *RQ3*. Again, however, adequate hardware and software, implementing the functionality required to run the experiment, needs to be available. To address *RQ1*, it was through the HPC Advisory Council[3] possible to get hold of hardware and software with preliminary support for IB CC. Using this equipment, experiments were run and *RQ1* addressed in Paper III[4] [13]. To address *RQ3*, on the other hand, the small IB network used to address *RQ1* is inadequate. A larger network is needed to study the scope of an injection throttling based congestion management mechanism in a realistic manner. At the time the research addressing *RQ3* was conducted, no such feasibly sized IB CC capable network was available, and as such, experiment was rendered invalid as a research method[5]. Therefore, simulation was picked as the reseach method of choice to address *RQ3*.

As with analytical modeling, the first step when using *simulation* as a research method is to create a model of the subject of research. Instead of defining a purely mathematical or statistical model, however, an abstract model is designed and mirrored in software. This model seeks to capture all relevant characteristics of the subject of study, including functional relations, to allow the execution of the software, the simulation itself, to represent the behavior of the model over time. The simulation is then used to draw conclusions about the subject being researched. In the case of computer networks, this translates into mapping all relevant network entities in software, including the interaction between the entities, and then study the simulation as an imitation of a corresponding physical[6] network. The software executing the simulation is commonly referred to as a *network simulator*.

The main strength of network simulation is tied to the abstract model's independence from available hardware, both with respect to functionality as well as quantity; simulation makes it possible to study functionality yet not implemented in hardware, and in a flexible way create network configurations that otherwise would not be available to the researcher. Even though using simulation often implies a development phase to implement the needed functionality into the network simulator, this phase is generally both faster and cheaper than it would be to implement the same functionality in hardware. Furthermore, the sim-

[3] http://www.hpcadvisorycouncil.com/

[4] A list of the six research papers written as part of this thesis is included at the end of this chapter.

[5] Software stack support for CC in IB has later been included in the Open Fabrics Enterprise Distribution (OFED) [14], and as such, IB CC has in theory been made more available. However, as a large IB installation with hardware support for CC is still an expensive and highly precious resource for the relatively few owners of such installations, experiment has remained a theoretical, but not practical, option. Nevertheless, at the time of writing this thesis, access to a large CC-capable IB installation was still pursued to further extend the study later presented in this thesis to address *RQ3*.

[6] The network being studied does not necessarily have to be physical, as virtual networks like an overlay network could also successfully be studied through simulation. In this thesis, however, we are concerned about physical networks.

ulator itself can often be used to automatically repeat several simulations using different network configurations. The main challenge of network simulation is to make sure all relevant aspects of the network of study have been captured by the abstract model being used, and that the model is correctly implemented by the simulator. To make sure that the executed simulations provide a trustworthy source of information, great effort needs to be spent to ensure the validity of the simulator. In addition, as the execution time and memory requirements of the simulator is a function of the abstract model's complexity, the researcher, when implementing the model, could be challenged by a trade-off between simulation accuracy and resource usage. In such situations, additional care needs to be taken to avoid disrupting the validity of the simulator.

Within network research, discrete event simulation [15, 16] (DES) has gained widespread use. Using DES, the operation of a network, or the subset of it being studied, is imitated as a sequence of discrete events in time, where each event represents a change of state in the model representing the network. Between consecutive events, no change of state occurs, and as such the simulation progress in time by handling the sequence of events[7]. The granularity of the division of time associated with the discrete events, together with the level of detail of the abstract model, determines the overall precision of the DES.

To address *RQ2*, *RQ3* and *RQ4*, two discrete event-based simulators have been used. To accurately capture all the characteristics of IB CC, including the IB CC parameters, a high-precision IB CC simulator was developed, based on a publicly available IB model implemented in the OMNeT++ framework [17]. A description of the IB CC simulator, as well as a validation of it, is given in Paper I [18]. The IB CC simulator is used to address *RQ2* and *RQ3*. In addition, the key elements of an injection throttling based congestion management mechanism was implemented[8], in part by porting from the IB CC simulator, into a C++ based custom-made network simulator, the INASim, developed at the Department of Computing Systems at the University of Castilla-La Mancha, Spain. Even though no separate paper has been written on the subject, this C++ based simulator has undergone a validation similar to the one described in [18]. The INASim simulator is used to address *RQ4*.

1.3 Thesis Outline

This thesis is divided into two parts. The first part consists of three chapters: chapter 1, Introduction (this chapter); chapter 2, Background; and chapter 3, Summary of Research Papers. Chapter 2 starts with a brief introduction to interconnection networks, and continues with a detailed discussion of congestion and congestion management in such networks, including a description of IB CC. Chapter 3 consists of extended abstracts of the 6 research papers written as part of the work with this thesis, and relates the papers to our four research questions from chapter 1.1. Finally, the second part of this thesis

[7] It is possible to assign more than one event to a given timeslot. All events given the same timeslot are then thought to happen concurrently.

[8] The implementation was done by Jesús E.-Sahuquillo, a coauthor of two of the papers included in this thesis.

contains the complete versions of the 6 research papers summarized in chapter 3. These 6 papers reflect the major part of the research being conducted as part of the author's Ph.D. studies. Five of the papers have been published, while the last one has been submitted, awaiting review. All paper titles, including place of publication, are listed below.

The Research Papers

Paper I	**InfiniBand Congestion Control, Modelling and Validation** [18]
published at	The 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools2011, OMNeT++ 2011 Workshop)
authors	Ernst Gunnar Gran and Sven-Arne Reinemo
Paper II	**On the Relation Between Congestion Control, Switch Arbitration and Fairness** [19]
published at	The 11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2011)
authors	Ernst Gunnar Gran, Eitan Zahavi, Sven-Arne Reinemo, Tor Skeie, Gilad Shainer and Olav Lysne
Paper III	**First Experiences with Congestion Control in InfiniBand Hardware** [13]
published at	The 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2010)
authors	Ernst Gunnar Gran, Magne Eimot, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, Lars Paul Huse, and Gilad Shainer
Paper IV	**Exploring the Scope of the InfiniBand Congestion Control Mechanism** [20]
published at	The 26th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2012)
authors	Ernst Gunnar Gran, Sven-Arne Reinemo, Olav Lysne, Tor Skeie, Eitan Zahavi, and Gilad Shainer
Paper V	**Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks** [21]
published at	The 40th Annual Conference – International Conference on Parallel Processing (ICPP 2011)
authors[9]	Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García, José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato
Paper VI	**Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks** [22]
submitted to	IEEE Transactions on Parallel and Distributed Systems
authors[10]	Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García, José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato

[9] The two first authors have equally contributed to the paper, and are listed in alphabetical order.
[10] See footnote 9.

Chapter 2

Background

This chapter starts with a brief introduction to the basic characteristics of lossless interconnection networks, presented with an eye to congestion and congestion management in such networks. Then, a detailed treatment of congestion and congestion management is given, including an introduction to the congestion control mechanism specified for InfiniBand.

2.1 Interconnection Networks

As mentioned in chapter 1.1, the type of interconnection network studied in this thesis, is the kind of network that typically serves as the shared communication infrastructure in a HPC supercomputer or modern data center. The overall performance of such computer systems are highly dependent on the performance of the interconnection network, which again depends on the characteristics and properties of the network itself [1, 2]. The basic characteristics of an interconnection network include the *topology*, the *routing* algorithm, the *switching* technique and the *flow control* mechanism. These four characteristics will be further described in the following sections, after we have defined the basic building blocks of a network: nodes and links.

2.1.1 The Basic Building Blocks

From an abstract, high level point of view, an interconnection network consists of three different elements: end nodes, forwarding nodes, and links, whereof the two first elements are sometimes combined into a single unit. An end node is an injector or a sink of traffic in the network, or both. In the capacity of an injector the end node is often referred to as a *source node*, while an end node in the capacity of a sink is often referred to as a *destination node*. A forwarding node, or just *switch*, is a node that forwards traffic in the network on behalf of other nodes, while a link is the bidirectional communication channel between two adjacent nodes[1].

[1]We will not be concerned about shard communication channels like buses in this thesis, and as such, a link is always connected to two nodes only. Furthermore, all links can carry traffic in both directions concurrently (full-duplex).

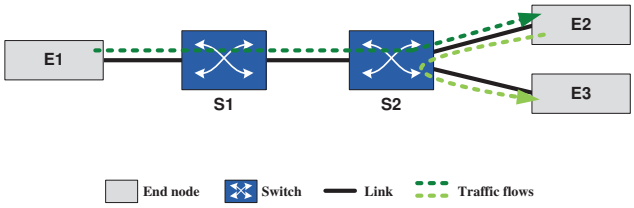


Figure 2.1: A small network with three end nodes and two switches.

Figure 2.1 shows a small network consisting of three end nodes, two switches, and four links. Moreover, two traffic flows have been added, represented by the dotted lines. In this scenario, the source node $E1$ is sending traffic to the destination node $E2$, while at the same time $E2$ acts as a source node for another traffic flow headed for the destination node $E3$. With respect to the flow from $E1$ to $E2$, the switch $S1$ is an *upstream* switch of the switch $S2$ (as $S1$ is closer to the source than $S2$), while $S2$ is a *downstream* switch of $S1$ (as $S2$ is closer to the destination than $S1$).

2.1.2 Topologies

The topology of a network defines how links and nodes are interconnected. In addition to be seen as either *regular* or *irregular*, a topology can be classified as *direct* or *indirect*, or in some cases as a hybrid of the two classes[2].

In a direct network, every node acts both as an end node and as a switch. That is, each node in the network is providing some sort of end node functionality (e.g. computation or storage), in addition to conducting forwarding of traffic on behalf of other nodes. On the contrary, in an indirect network there is a clearer distinction between end nodes and switches. The switches are purely acting as forwarding nodes for the end nodes, and as such, in an indirect network, it is possible to have a switch connected only to other switches and no end nodes. The small network in figure 2.1 is an indirect network, while figure 2.2 shows two examples of direct network topologies, the 4D hypercube and a 2D mesh (we will soon get back to the two green traffic flows plotted in the figure).

In a regular topology, the links interconnecting the nodes follow a well-defined connection pattern. Two large, general groups of such patterns include the ordering of nodes along orthogonal n-dimensional rooms, and different strictly defined tree structures [1, 2]. The two topologies in figure 2.2 are both examples of the former. In an irregular topology, no similar connection pattern is followed. Nodes are freely connected. This freedom adds

[2]In addition, the *shared-medium networks*, like buses and ring networks, constitute a third topology class. Despite their relatively simple structure and natural support for broadcast communication, a shared-medium network, where only one node can send traffic at a time, is seen to be unsuited for HPC supercomputers and data centers. Consequently, we will pay no further attention to shard-medium networks in this thesis.

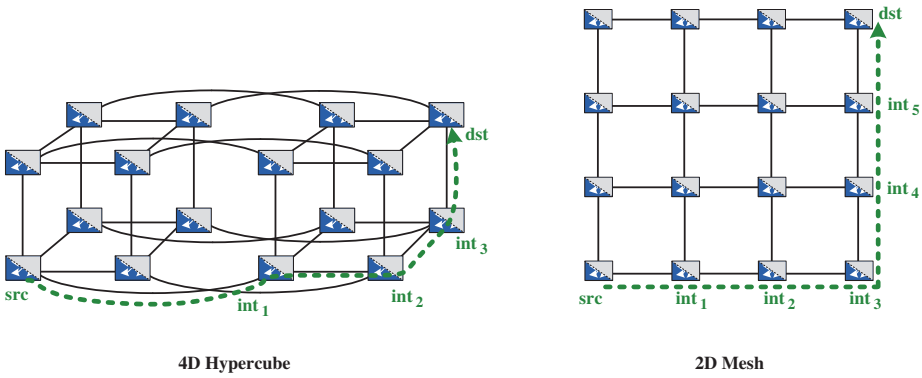


Figure 2.2: Two direct networks, each having 16 nodes.

flexibility to the network, particularly when it comes to incremental expansion, as new nodes can be added as needed without adhering to a regular connection pattern. While such a flexibility is highly appreciated in e.g. and office environment, a particular quality of regular topologies are usually more sought-after when designing an high performance supercomputer and data center: When using a regular topology, it becomes possible to utilize knowledge about the structure of the topology to optimize the network, e.g. when defining legal routes for traffic to follow through the network (for more about routing, see section 2.1.3). Notice though, that a regular topology could quickly be transformed into an irregular one when faults happen in the network.

The choice of topology will influence how traffic can be distributed in the network, and in particular (together with the routing algorithm and the applications being executed) influence how likely links and switches are to be saturated and congested. Notice that both topologies in figure 2.2 have 16 end nodes. The hypercube, however, has 33% more links than the mesh. That is, 32 v.s. 24 links, correspondingly. Not only does the additional links imply more roads in the network to distribute the traffic onto, but the additional links also decrease the distance between non-neighboring nodes, and by that the average distance traffic has to travel in the network. The *network diameter*, defined as the longest shortest-path between any two nodes in the network, is smaller in the hypercube than in the mesh. The difference in network diameter between the two topologies in figure 2.2 is exemplified as traffic sent from the node *src* to the node *dst* (the dotted lines in the figure). In the hypercube, the traffic has to pass through three intermediate nodes, while the number has increased to five in the case of the mesh. Summing up, the additional links of the hypercube are potentially lowering the likelihood of congestion to occur in the network, both because the traffic could be distributed onto more links, and because the traffic is potentially spending less time inside the network because of the smaller network diameter. Nevertheless, congestion may still occur as the links are shared between nodes in the topology. Among all possible topologies, there is actually only one that by its own nature makes congestion impossible and congestion management unneeded, and that is

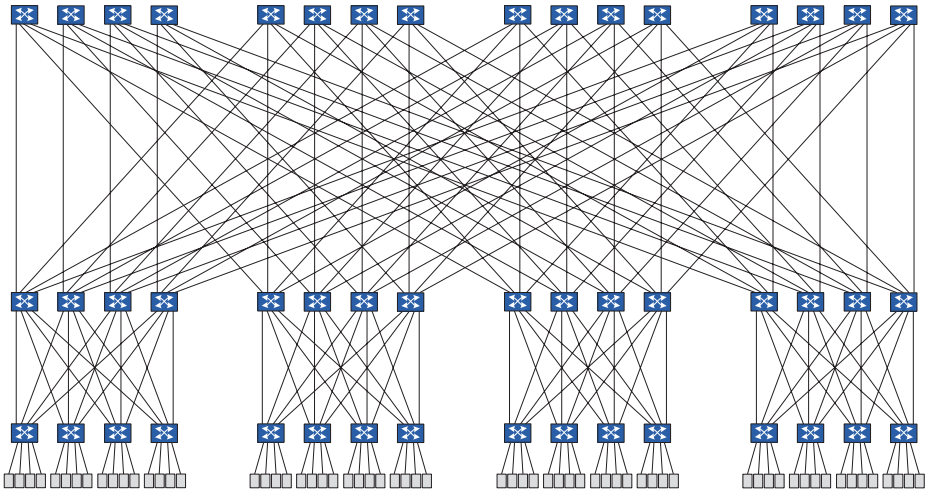


Figure 2.3: A fat-tree with 16 root switches, 32 internal switches, and 64 end nodes.

the fully connected topology where every node in the network is connected to every other node with a separate, dedicated communication channel. In such a network, even though a link can be saturated if a source node is sending more traffic than a destination node can handle, no congestion spreading is possible, and by that, no congestion management is needed. A fully connected topology does not scale, however, and is of little interest in the context of HPC supercomputers or large data centers.

There is currently a range of different topologies in use in HPC supercomputers and high performance data centers. A popular family of topologies is the k-ary n-cubes (torus topologies) [23]. A k-ary n-cube is a k-ary n-mesh with wraparound links, that is, a direct orthogonal topology over n dimensions and k nodes in each dimension. By adding wraparound links to the 4-ary 2-mesh in figure 2.2 (the 2D mesh), the topology is transformed into an 4-ary 2-cube. Topologies used in real life HPC installations are naturally much larger than this small example though, where e.g. the 3D torus of the Titan Cray XK7 [24] at the Oak Ridge National Laboratory interconnects 18.688 compute nodes, while the 5D torus of the IBM Sequoia [25] at the Lawrence Livermore National Laboratory interconnects 98.304 compute nodes (the top two supercomputers in the top500 list [6], November 2012).

Another popular family of topologies is the fat-tree multistage interconnection networks [26, 27]. This type of networks has almost become the de facto standard for InfiniBand based installations on the top500 list. The fat-tree is a family of regular indirect tree topologies with multiple roots, where the roots and the branches of the interconnected trees are build from switches, while the end nodes reside as leaves of the trees[3]. Figure 2.3 depicts a fat-tree consisting of 16 root switches (at the top), 32 internal switches, and 64

[3]Hybrid topologies also exist where additional end nodes are connected to the internal switches in the tree structures.

end nodes (at the bottom). This particular tree is an example of a parameterized subtype of the fat-tree family, the k-ary n-tree [27], where k defines the number of "downward ports" at each switch (e.g. the number of end nodes connected to each leaf switch), while n denotes the number of levels in the tree. Thus, the fat-tree in figure 2.3 is a 4-ary 3-tree. Notice that each level closer to the roots interconnects a set of smaller fat-trees (further away from the root) in a regular fashion, where the full bisection bandwidth [2] is maintained at each level. This characteristic, making it possible in an hierarchically way to increase the size of the topology while maintaining full bisection bandwidth, together with favorable properties when it comes to fault tolerance and non-blocking deadlock-free routing, has made the fat-tree a widely deployed topology. Due to its popularity, particularly in the context of InfiniBand networks, the fat-tree has received special attention when we have addressed *RQ3* and *RQ4* in the research papers IV, V and VI. The top eight of the top500 list (November 2012) includes three HPC supercomputers utilizing the fat-tree topology; the SuperMUC [28] at the Leibniz Rechenzentrum, the Stampede [29] at the Texas Advanced Computing Center, and the Tianhe-1A [30, 31] at the National Supercomputing Center of Tianjin.

2.1.3 Routing

The routing algorithm of a network defines which paths, or *routes*, through the network traffic flows are allowed to follow. That is, when a source node is sending traffic to a destination node, the routing algorithm decides which links and switches through the network this particular flow will use. Notice, looking at figure 2.2 and figure 2.3, that more than one possible path could be available when a routing algorithm is to make its decisions. How the routing algorithm then through path selection chooses to distribute the traffic in the network could greatly influence the likelihood of switches and links in the network being congested.

A routing algorithm can be classified as *deterministic*, *oblivious*, or *adaptive*, depending on the nature of operation and the degree of freedom the algorithm provides in dynamically changing routes in the network. A deterministic routing algorithm always chooses a single fixed route from a source node to a destination node. That is, all traffic going from a specific source to a given destination will always use the same set of intermediate links and switches. An oblivious routing algorithm, on the other hand, provides a set of legal routes from a source node to a destination node, where one of the routes is picked when traffic is to be sent. Notice then, that traffic from a specific source to a given destination does not always follow the same path through the network. The selection procedure of an oblivious routing algorithm, the procedure selecting one of the possible paths through the network, does not, however, use any information about the current state of the network when a path is to be selected. For instance, the selection procedure could randomly pick one of the possible routes through the network each time a routing decision is to be made. On the contrary, the selection procedure of an adaptive routing algorithm, an algorithm that also provides a set of legal routes from a source node to a destination node, takes into account the current state of the network each time a route is to be selected. As such, an adaptive routing algorithm is able to dynamically change the routing, and by that

the traffic distribution, in the network depending on the current network state, e.g. the historical link and switch load in the network could be considered.

In general, the goal of a routing algorithm is to realize the potential of the topology. To reach this goal, the routing algorithm needs to distribute the traffic in the network in a way that maximizes the utilization of the network resources, i.e. the links and the switches, on behalf of the end nodes and their requirements, while avoiding deadlocks[4] [1]. An efficient routing algorithm then keeps a good balance between distributing load in the network and keeping the length of the routes short. Ideally, the routing algorithm would be both non-blocking[5] and provide shortest-path routes between any pair of nodes.

While a deterministic routing algorithm could provide shortest-path routes, the lack of path diversity inhibits the algorithm from performing load balancing. As a single fixed path is always used for each source-destination pair, other paths that could potentially balance the load in the network for a given scenario, and by that lower the probability of congestion, are not utilized. In fact, for every deterministic routing algorithm, there exists a traffic pattern that will cause large load imbalance in the network [2]. On the contrary, an oblivious routing algorithm could distribute the traffic in the network through utilization of its path diversity, though possibly at the expense of not selecting a shortest-path route. Furthermore, an adaptive routing algorithm could even react and reroute traffic as a consequence of congestion in the network. As such, both oblivious and adaptive routing algorithms could alleviate the problems of congestion. However, as we will get back to in section 2.3, an oblivious or adaptive routing algorithm can by itself not always successfully avoid or resolve congestion. More specifically, an oblivious or adaptive routing algorithm could actually make a situation of congestion worse if there is no alternative route around the location in the network originating the congestion. This is in particular so if the root of congestion is at the last downstream switch connected to an end node, or the end node itself.

While an adaptive routing algorithm ideally could be the algorithm of choice for a high performance lossless interconnection network, and indeed is in use in some large installations like the before mentioned Titan Cray XK7 and IBM Sequoia, deterministic routing algorithms are still in widespread use. They are relatively simple and inexpensive to implement, ease deadlock prevention, and has by nature a property that is important for certain applications; they provide in-order delivery of traffic. In addition, there is no selection procedure present that could potentially introduce additional latency to the routing algorithm. More specific, the InfiniBand architecture does not currently support

[4] A deadlock is a phenomenon that might arise in a network if traffic being sent is allowed to hold on to a resource in the network while requesting another. If care is not taken, and chains of such requests are allowed to be made in a circular fashion, a set of traffic flows could end up sharing the fate of the dining philosophers [32, 33]; they could in a circular manner be waiting for resources held by others, while refusing to let go of the resource they already hold on to themselves. Thus, no further progress is possible, the traffic is halted, and (a part of) the network has deadlocked. If the topology by itself does not prevent circular chains of requests, it is common to limit the available routes provided by the routing algorithm to avoid that such chains form.

[5] A routing algorithm is non-blocking if it is always possible to route traffic from a source node *src* to a destination node *dst* without interference from any other routed traffic flow in the network, given that *src* is the only node sending to *dst*. For a routing algorithm to be non-blocking, the topology needs to have a full bisection bandwidth.

adaptive routing, and in particular, popular InfiniBand supported routing algorithms like FTREE [34], DOR (Dimension Order Routing) [35], and LASH [36] are deterministic.

Before we move on to talk about switching and flow control, it is appropriate to mention that the responsibility of choosing a route through the network can be given to the source node, *source routing*, or handed to the intermediate switches, *incremental routing*. When source routing is used, the source node chooses the entire route traffic is to follow through the network. While source routing offloads the switches as they are exempted from making routing decisions, it comes at the cost of embedding information about the entire chosen route into the traffic being sent. On the contrary, when incremental routing is used, the source node does not make any routing decisions, but includes information about the destination node in the traffic. The intermediate switches then have to base their forwarding at each intermediate step on this information about the destination node, typically the destination node ID or *address* (e.g. through looking up the destination address in a forwarding table).

2.1.4 Switching and Flow Control

While the topology of a network defines the interconnection of nodes and links, and the routing algorithm provides legal routes through the network, the switching and flow control mechanisms determine how traffic is forwarded along a given route through the network. Roughly speaking, the switching technique defines how traffic flows through the switches, while the flow control mechanism defines when traffic is allowed to flow between neighboring nodes[6]. As switching and flow control together determine how traffic flows through the network, they are naturally coupled tightly together. In addition, they are closely connected to the buffer management in the network. A *buffer* in this context is a storage area in a switch (or an end node) temporarily holding traffic on its way through the network.

Switching

Several switching techniques exist, each having its set of advantages and disadvantages. In the followings sections we will briefly describe two main families of such techniques; *circuit switching* and *packet switching*, whereof the last one is further divided into three subclasses, *store-and-forward*, *virtual cut-through switching*, and *wormhole switching*.

The carrying idea in circuit switching is a heritage from telecommunication, where a dedicated communication channel had to be established before two parties could communicate. Thus, in a circuit switched network, the first step of communication is to reserve the appropriate resources throughout the network between the nodes that want to communicate. Once this reservation has been made, and by that a *circuit* established, data may be sent over the circuit until the circuit is deallocated. Circuit switching removes the need for general congestion management. As resources are reserved in advance, no

[6]In this thesis, the term *flow control* will be used rather narrowly to refer to flow control at the link-level. That is, flow control between two directly connected nodes. In the network literature in general, the term flow control could include both switching and switching techniques in general, as well as flow control between end nodes.

contention arises (after a circuit has been established), and congestion is avoided. Note, though, that care needs to be taken during the set up phase of a circuit, especially if the requests for resources are allowed to happen at the head of the first data being transmitted, as contention, and even deadlocks, could occur during this phase. While the use of reserved resources makes forwarding efficient for the users of already allocated circuits, the efficiency of the network as a whole could be low. Resources assigned to a given circuit are private to that particular circuit and cannot be used by others. Thus, if a circuit is currently not carrying traffic, resources are left idle, even if other nodes in the network are awaiting access to the same resources. Furthermore, the set up phase of a circuit adds initial delay to communication. If most or all communication is short-lived, e.g. consists of a small number of short messages (or maybe even a single short one), the added delay from a set up phase could make up a considerable part of the total communication latency between two nodes.

Packet switching, having roots back to the early 1960's, is a switching technique that, compared to circuit switching, more dynamically is able to utilize resources in the network. The traffic being sent between nodes is partitioned into *packets*. Each packet is then sent and routed individually, without any initial reservation of resources. This packet independence and freedom when it comes to resource usage, enables the switches in the network to independently forward packets onto free links without considering if the link is reserved by others, but currently idle (as could be the case with circuit switching), given that the forwarding happens in accordance with the routing algorithm in use. The increased flexibility comes at a cost though. The lack of a reserved path through the network implies that a switch might have to store a packet while the packet is waiting for access to a currently busy link. As such, additional buffers are needed at the switches to handle contention for outgoing links, and communication latency may increase for individual packets[7].

The most basic form of packet switching is store-and-forward (SAF). Each time a packet arrives at a switch, the complete packet is buffered before it is forwarded. A switch then needs to be able to store at least one packet per incoming link, while it is sufficient for the flow control to operate at the packet level. Recall that the task of the (link-level) flow control is to make sure that one node does not send more traffic to another node than this other node can handle. While SAF is easy to implement, and makes it possible to remove erroneous packets from the network (if e.g. CRC [37] is used at the packet level), the added latency per intermediate switch as a packet traverses the network is high; the time it takes to receive the complete packet at each intermediate switch is added to the total transfer latency of a packet.

Virtual cut-through switching [38] (VCT) takes advantage of the fact that it is possible to decrease the total transfer latency of a packet if the packet is allowed to "cut through" a switch before the complete packet is received. If the destination address of a packet is stored at the head of the packet, i.e. as part of the *packet header*, which the destination address typically is to allow for a fast selection of the next link to use, the packet could be forwarded as soon as the packet header is received and the outgoing link has been selected

[7]When circuit switching is used, no buffering is actually needed in the switches after a circuit is established [2].

(and is free). Thus, the added latency of a packet per intermediate switch could be reduced to the time it takes to receive the packet header only, compared to SAF switching where the whole packet needs to be received before forwarding takes place. The flow control still operates at the packet level, though. Consequently a switch still needs to be able to store at least a complete packet at each incoming link in the case of contention for outgoing links.

A switching technique that has the potential of not only reducing the transfer latency of packets compared to SAF, but in addition lowers the buffer requirement at the switches compared to VCT (and SAF), is wormhole switching [39, 40]. Using wormhole switching, each packet is partitioned into *flits*, where the first flit of a packet contains the destination address. As soon as the first flit has arrived at a switch, the forwarding decision can be made, and the forwarding take place as soon as the selected outgoing link is free. This behavior is comparable to VCT switching. However, as opposed to VCT, wormhole switching makes use of flow control at the flit level. This increase in the granularity of the flow control mechanism makes it possible for a downstream switch to halt a neighboring upstream switch before a whole packet is received. Thus, the buffer requirements at a switch can be reduced to a single, or a few, flits per incoming link. The packet is still the smallest routable unit, though, meaning that flits from different packets cannot be allowed to mix. The result is that a packet traversing the network will remain stretched like a worm through switches and links, even when the head of the worm is blocked at a switch due to link contention. Notice then, that such a worm may occupy resources in the network that could be used by other packets, packets that are not headed for the switch where the head of the worm is being blocked.

As larger buffers in switches became easier to integrate, saving buffer space got less focus, and VCT increased in popularity. In particular, VCT is now the common switching technique used in InfiniBand switches [41], and thus the switching technique we have chosen for our congestion management studies. Notice that VCT behaves like wormhole switching in a network with little or no contention, allowing for low transfer latency. As the network moves closer to saturation, however, the behavior of VCT gets closer to the behavior of SAF. VCT's ability to store at least a complete packet at a switch then effectively removes any "frozen worms", worms that could potentially block other traffic in the network when wormhole switching is used. The absence of such worms eases the congestion management.

Flow Control

To prevent an upstream node from overwhelming a directly connected[8] downstream node, and by that avoid packet loss at the downstream node, *flow control* is needed at the link level in a lossless interconnection network utilizing packet switching[9]. The task of the flow

[8] In this section, when we talk about upstream and downstream nodes in the context of flow control, we will always assumed that the two nodes are directly connected with a link, even when the term "directly connected" is left out to make the text more readable.

[9] We will in this thesis not be concerned about circuit switched networks, or any other potentially "bufferless" networks e.g. utilizing continuous rerouting in an attempt to avoid packet drops, and as such, we will restrict our discussion of flow control to buffered packet switched networks only.

control mechanism is then to make sure that a node always has buffer space available when a packet or flit arrives. In particular, to ensure this invariant, a downstream node needs to be able to communicate to an upstream node when buffer space at the downstream node is available (or not), and possibly the amount of such storage.

There are three main families of flow control mechanisms [2]: *credit-based* flow control, *on/off* flow control, and *ack/nack* flow control. In a credit-based flow control scheme, an upstream node keeps track of how many *credits* of free buffer space a downstream node currently has. One credit could for example correspond to one packet or one flit. For the sake of simplicity, we will in the following assume that the flow control operates at the packet level, and that one credit equals one packet. Each time the upstream node then forwards a packet, the local credit counter is decreased by one. When the counter reaches zero, forwarding to the downstream node is prohibited. At the reception of the packet, the downstream node does nothing credit-wise until the downstream node itself is able to forward the packet, or process it if the downstream node is an end node. Then, the downstream node sends a new credit to the upstream node, which then again increases the local credit counter. To ensure a steady flow of traffic, the buffer of the downstream node, and the corresponding credit counter at the upstream node, initially needs to be big enough to allow for continuous forwarding of packets during a *credit round-trip delay*. This delay is given by the time it takes to send a credit from the downstream node to the upstream node, process the credit, send a packet from the upstream node to the downstream node, and then finally process the packet at the downstream node, thus allowing for a new credit to be sent.

When on/off flow control is used, the upstream node is either in an *on* state or an *off* state, depending on if the node is allowed to send traffic to the downstream node or not, correspondingly. The downstream node signals the upstream node each time it is necessary to change the state. To implement this flow control scheme, the downstream node needs to keep track of two thresholds, T_{off} and T_{on}, related to its buffer. If the amount of free buffer space drops below T_{off} and the upstream node is currently in the on state, an *off* signal is sent to the upstream node. Likewise, if the amount of free buffer space increases above T_{on} and the upstream node is currently in the off state, an *on* signal is sent to the upstream node. To make sure no packets[10] are dropped, the T_{off} threshold needs to be set high enough to allow for a continuous reception of packets at the downstream node, without running out of buffer space, while the *off* signal propagates to the upstream node, including any packets already "on the wire" when the *off* signal is received. At the same time, to ensure that the downstream node does not unnecessarily run out of packets to forward, the T_{on} threshold needs to be low enough to ensure that there are still enough packets in the buffer to allow for continuous forwarding at the downstream node while the node is waiting for the propagation of the *on* signal to the upstream node and, in addition, for possible new packets to arrive from that node. Furthermore, T_{on} needs to be set higher than, or equal to, T_{off}.

Ack/nack flow control takes on an optimistic approach. Using this flow control mechanism, the upstream node is always allowed to send packets. To still achieve lossless operation, the downstream node acknowledges all packets it is able to buffer, while pack-

[10]For simplicity, we will continue to assume that the flow control operates at the packet level.

ets that are dropped result in negative acknowledgments. That is, each time a packet is received at the downstream node, an *ack* is sent to the upstream node, while a *nack* is sent to the upstream node each time the downstream node drops a packet. It is then the task of the upstream node to resend any nack'ed packets to ensure lossless operation. Thus, the upstream node needs to buffer each sent packet until the corresponding *ack* is received. Furthermore, as retransmission of nack'ed packets could result in out-of-order delivery of packets to the downstream node, the downstream node needs to halt forwarding of all ack'ed packets following a nack'ed packet, until the nack'ed packet has been retransmitted and successfully buffered at the downstream node. Then, in-order forwarding of packets can continue at the downstream node. The upstream node's need to buffer a packet while waiting for an *ack*, the possible retransmission of packets over the link, and furthermore, the possible need to buffer and rearrange packets at a downstream node, are together characteristics that could result in an inefficient use of network resources. Thus, ack/nack flow control is rarely used in lossless interconnection networks.

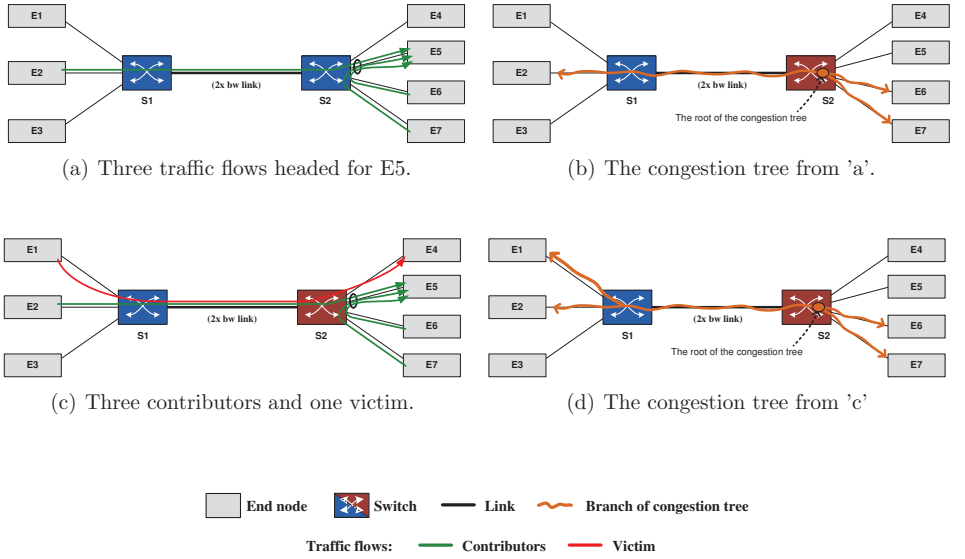
While the three families of flow control mechanisms differ in the way they implement lossless operation between neighboring nodes, they all create the before mentioned backpressure effect (chapter 1.1). This also goes for the absolute credit-based flow control scheme specified for InfiniBand, a scheme we have applied in our studies, where the upstream node is kept notified about the *total* amount of traffic an upstream node has been authorized to send to a downstream node since the link between them was initiated [42]. Indeed, the backpressure effect is the wanted local effect to hinder one node from swamping another. As the backpressure effect progresses between nodes, however, congestion spreads. To avoid any negative consequences of such congestion spreading, proper congestion management is needed. Congestion and congestion spreading is the subject of our next section.

2.2 Congestion in Lossless Networks

Congestion is a situation that occurs in a network with shared network resources if too much traffic is sent into a certain part of the network, exceeding network capacity in this area. Initially, the buffers in the node at the root of the congestion will deplete. If the situation continues over time, however, the flow control in a lossless network will make buffers in surrounding nodes fill up as well, and as this backpressure effect of the flow control continues, the phenomenon known as *congestion spreading* develops and a *congestion tree* is grown[11] [43]. The branches of this tree grow as buffers continue to deplete in surrounding nodes, and may in the end grow all the way into the source nodes in the network. Let us use a small example to further study the effect of congestion spreading.

Figure 2.4(a) shows our topology from figure 2.1 extended with four new end nodes. Two of the new nodes are connected to $S1$, while the other two are connected to $S2$. In addition, the link between $S1$ and $S2$ has been given twice the link bandwidth of the other links in the topology. Congestion spreading, and the Head-of-Line blocking phenomenon

[11] The phenomenon is called *tree saturation* in Pfister and Norton's paper from 1985 [3].



(a) Three traffic flows headed for E5.

(b) The congestion tree from 'a'.

(c) Three contributors and one victim.

(d) The congestion tree from 'c'

Figure 2.4: Congestion and congestion trees in a lossless network.

explained below, will be present even if all links have the same capacity, but by giving the switch-to-switch link twice the capacity of the other links, the effect of the Head-of-Line blocking will be more obvious. Thus, the negative consequences of congestion and congestion spreading will be easier to grasp. Furthermore, the topology shown in figure 2.4(a) is one of the topologies used in four of the research papers being part of this thesis; paper I, II, III, and V (for more information about the papers, see chapter 3).

Three traffic flows are added to the topology in figure 2.4(a); the end nodes $E2$, $E6$, and $E7$ send traffic headed for the end node $E5$. We assume that these three source nodes all want to send at the full link capacity of their connected links. The link between the switch $S2$ and the destination node $E5$ will then become a bottleneck, where each of the three sources will get access to 1/3 of the link. As the three sources continue to send traffic to $E5$, traffic will start to pile up at $S2$, the buffers in $S2$ will fill, and the flow control will notify the neighbors $S1$, $E6$ and $E7$ of $S2$ that $S2$ is running out of buffer space[12]. The three source nodes have, by their *hot* flows headed for the bottleneck link, become *contributors* to congestion. At the switch $S1$, the flow control messages from $S2$ will prevent $S1$ from forwarding traffic from $E2$ to $S2$ at the desired pace. Thus, the backpressure effect of the flow control has started, buffers will fill at $S1$, and furthermore, the flow control will notify $E2$ about $S1$'s lack of buffer space.

Figure 2.4(b) shows the congestion tree resulting from the traffic scenario in figure 2.4(a), a tree formed by the buffer occupancy of the contributors to congestion. The tree takes root at the outgoing link connecting $S2$ to $E5$, while the branches of the tree

[12]The nodes $E4$ and $E5$ could also be notified by the flow control, depending on the flow control scheme and the buffer strategy being implemented in $S2$, but for our example such possible notifications are not of importance.

stretch along links and switches towards the contributors to congestion. This far the congestion tree does not cause any real harm to the network performance. The link between $S2$ and $E5$ is a bottleneck, there is no alternative path around it, and thus there is not much else to do for the packets stored along the branches of the congestion tree than to wait for their fair share of the bottleneck link. If a new traffic flow is added from $E1$ to $E4$, however, the potential harm that can be caused by a congestion tree can be observed.

Figure 2.4(c) extends the previous example with a new traffic flow added from the source node $E1$ to the destination node $E4$. As this new flow does not request the bottleneck link between $S2$ and $E5$ (the flow is *cold*), and the link between the switches has twice the capacity of the other links in the network, we would wish for this new traffic flow to be able to progress towards its destination unaffected by the contributors to congestion and the congestion tree they have grown. However, as the congestion tree is occupying buffers in $S2$ and $S1$, the utilization of the link between the switches is held back by the congestion tree. In fact, the cold flow from $E1$ will not be able to progress any faster than the flow from $E2$ contributing to congestion, that is, at only 1/3 of the end node-to-switch link capacity. Recall that the buffer in $S2$ is full due the contested link towards $E5$. Traffic originating from $E2$ will at $S2$ get its fair share of the bottleneck link, thus access to the bottleneck link is given to the flow from $E2$ each 1/3 of the time the bottleneck link can carry traffic. When the flow from $E2$ is granted access to the bottleneck link, and a packet from $E2$ is forwarded, the flow control of $S2$ will tell $S1$ that one more packet can be sent downstream. $S1$, being a fair switch, will alternate between forwarding packets from $E1$ and $E2$. If a packet from $E1$ is chosen, the packet will immediately be forwarded at $S2$, and $S1$ will once more be allowed by the flow control to forward a packet to $S2$. This time, however, a packet from $E2$ will be chosen. Then, upon reception at $S2$ the packet from $E2$ fills the buffer, which once more will remain full until a new packet from $E2$ is granted access to the bottleneck link. Thus, the net effect is that the progress of the flows from $E1$ and $E2$ both are dictated by $E2$'s access to the bottleneck link; 1/3 of the link capacity.

This phenomenon where the traffic flow from a source node is held back by a congestion tree even though the flow itself will never request resources at the root of the tree, is referred to as *Head-of-Line* (HoL) *blocking*. A source node experiencing HoL blocking is referred to as a *victim* of congestion, while the corresponding flow is referred to as a *victim flow*. In our example, the HoL blocking results in an unnecessary $66\frac{2}{3}\%$ drop in performance for the victim $E1$. In addition, as the victim $E1$ is blocked at $S1$, traffic will start to pile up towards $E1$ as well. Hence the congestion tree will actually continue to grow, even towards the victim of congestion. Figure 2.4(d) shows the congestion tree of our example as it has grown to include $E1$. While this additional growth has no real effect in our small example, similar growth towards victims in a larger topology could make the congestion tree, and the corresponding HoL blocking, spread to a significant part of the total network.

There are two key observations to make from our small example. First of all, the main reason we see a drop in performance when congestion spreads in the network is the HoL blocking affecting the victim flows. To make matters worse, the HoL blocking may even fertilize a congestion tree and make it grow to a larger part of the network. Secondly, as

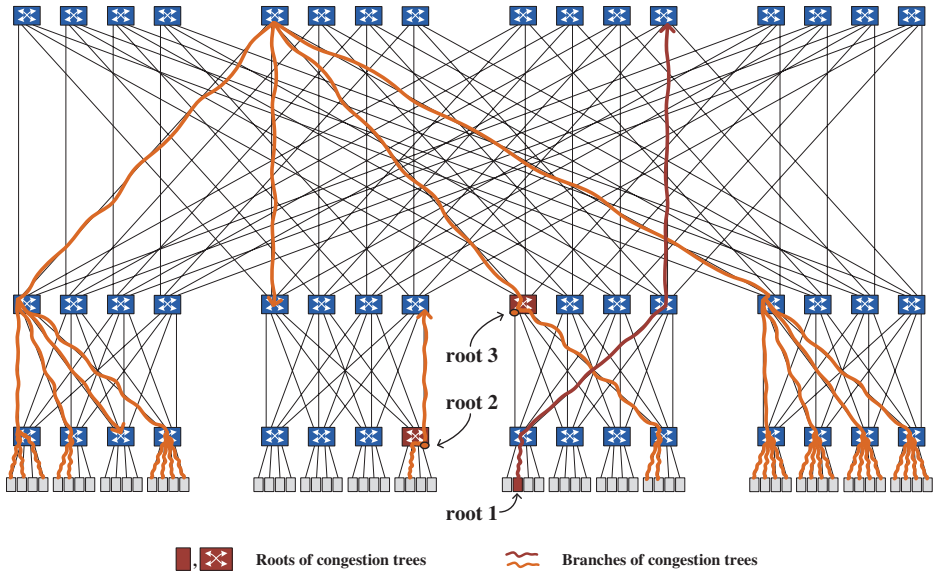


Figure 2.5: A fat-tree with three congestion trees. One root is located at an end node while the other two roots are located at switches.

there is no alternative path around the root of the congestion tree in our example, it is not possible to improve the performance of the contributors to congestion. The link towards the end node limits the performance. More specifically, the removal of any HoL blocking caused by the congestion tree will not improve the performance of the bottleneck link.

In our example, the congestion was caused by several sources concurrently sending traffic to the same destination, and the root of the corresponding congestion tree was located at the last switch before the destination node. In general, the root of a congestion tree could be located at any switch in a network, or at an end node not being able to process traffic at the capacity of the connected link, and several congestion trees might be present in the network at the same time. Figure 2.5 shows our fat-tree from figure 2.3 where three congestion trees are growing. An end node (*root* 1), not being able to process traffic at the speed of reception, is the root of a single-branched congestion tree growing towards the corresponding source node. One of the switches (*root* 2) is the root of a small congestion tree with, this far, only two branches, while another switch (*root* 3) is the root of a large congestion tree with several branches, whereof 27 of the branches have grown all the way into source nodes in the network (some of the nodes possibly being victims).

Furthermore, the reasons for congestion could be many. As in our initial example, hot spot traffic patterns could results in congestion. Other reasons for congestion could include network burstiness, rerouting around faulty regions in the network, and link frequency/voltage scaling to lower link speeds in order to save power. If a network administrator knows all these factors in advance, the consequences of congestion may be alleviated by effective load balancing of traffic. Generally, however, this is usually not the

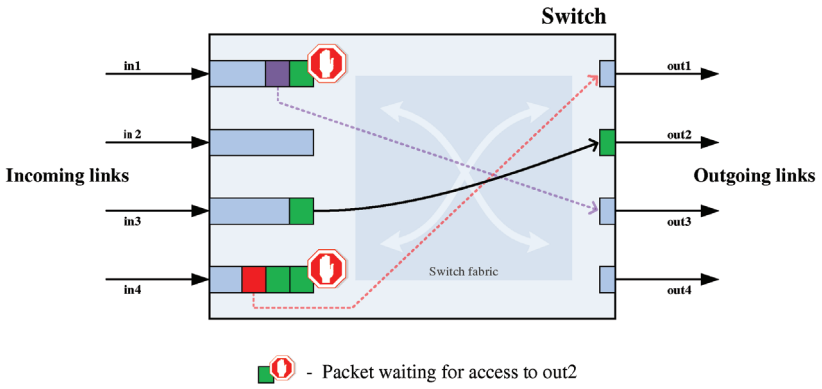


Figure 2.6: HoL blocking inside a switch.

case. In particular, in a large parallel computer running multiple jobs through virtualization, or in a modern data center accepting jobs as an on-demand service facilitating cloud computing, to have the complete knowledge about the traffic patterns in the network might be impossible. Furthermore, in the case of a hot spot destination being present in the network, no load balancing or adaptive rerouting is possible to avoid congestion. Thus, proper congestion management is needed to avoid performance degradation in the network.

High-Order and Low-Order HoL Blocking

The above mentioned HoL blocking phenomenon is the cousin of a HoL blocking phenomenon that might occur inside a switch. This phenomenon internal to switches will be explained by means of an example, using the switch shown in figure 2.6. This switch, having four links, has been drawn unfolded with the four input ports to the left and the four output ports to the right. Hence, the incoming links $in1$, $in2$, $in3$, and $in4$ correspond to the outgoing links $out1$, $out2$, $out3$, and $out4$, respectively. The switch has buffers organized as first-in-first-out (FIFO) queues at the input ports, each buffer being able to hold at least three packets. The switch fabric represents the internal network of the switch, connecting input port to output ports. Concurrent transfers of packets through the switch fabric are possible if the packets originate from different input buffers and are headed for separate output ports. If two packets are requesting the same output port, however, one of them will have to wait in its input buffer while the other packet is forwarded.

In figure 2.6, the first packet at each of the three buffers corresponding to the input links $in1$, $in3$, and $in4$, i.e. the *green* packets, is requesting the outgoing link $out2$. The packet from $in3$ is granted access to $out2$, while the two other packets residing at the head of the queues from $in1$ and $in4$ have to wait. The queues from both $in1$ and $in4$, however, also hold packets headed for other outgoing links. The queue from $in1$ holds a packet headed for $out3$, the purple packet, while the queue from $in4$ holds a packet headed

for *out*1, the red packet. Due to the FIFO nature of the buffers, the purple and the red packets both have to wait, even if the resources they are requesting, the outgoing links *out*3 and *out*1, respectively, are free. The two packets are HoL blocked internally in the switch, and will not be able to progress through the switch fabric until the green packets in front of them have been forwarded.

The HoL blocking caused by internal contention in a switch and the HoL blocking caused by congestion spreading in a network share the same abstract characteristics: In both cases some packets P_v are blocked by other packets P_c even though the packets in P_v will never request the resources holding back the packets in P_c. Thus, the two phenomena are given the same name. To keep the two phenomena apart when discussing both of them together, the switch internal HoL blocking is sometimes referred to as *low-order* HoL blocking, while the HoL blocking caused by congestion spreading is referred to as *high-order* HoL blocking.

In this thesis we will mainly be concerned about high-order HoL blocking[13]. Low-order HoL blocking can, as we will get back to in section 2.3, be avoided by clever organization of buffers in the switches. To handle high-order HoL blocking, on the other hand, interaction between traffic flows as they span more than a single node needs to be considered. As such, an efficient network-wide congestion management mechanism needs to be able to handle congestion spread by the flow control mechanism across multiple nodes in the network.

2.3 Congestion Management

Congestion trees like the ones in figure 2.5, and the possible HoL blocking they cause, may lead to severe performance degradation in a lossless interconnection network if no countermeasure is taken [3, 13, 20, 21, 22, 43, 44]. While a combination of tuning and tailoring of network characteristics for a given application, together with overprovisioning of network resources, have been able to keep the challenges of congestion at a distance, these techniques are, as mentioned in section 1.1, about to become obsolete; E.g. component failures in a large network or the use of virtualization makes it difficult to predict the traffic patterns a network will be exposed to, making tuning and network tailoring hard, while the use of overprovisioning directly contradicts the current trend of making HPC installations and data centers cost-effective and green.

Overprovisioning and network tailoring are proactive congestion management techniques in the sense that the challenges imposed by congestion is addressed by avoiding, or making it less likely, that congestion occurs in the first place. Other proactive techniques have been suggested, though they are not appropriate for general HPC systems or modern data centers as they do not in a universal and scalable way conduct congestion management; In [45] and [46], a priori knowledge about the traffic is assumed, either as hot traffic is predefined as synchronization messages [45], or as hot spot memory contention is alleviated by using a software combining tree [46]. Flit-reservation flow control [47]

[13]Research papers V and VI also consider low-order HoL blocking. However, to avoid high-order HoL blocking is still the main challenge addressed in these papers.

improves buffer utilization and lowers the transfer latency of traffic by the use of sophisticated resource scheduling. Control flits are sent ahead of data flits (possibly by the use of a separate control network) to do resource reservation for the data. While the control flits can do reservations ahead of time in the case of contention, the data flits can still be stalled and congest the network. In [48], a request-grant mechanism detains packets at the input buffers of the network until it is safe to inject them without creating congestion. While this mechanism successfully avoids the creation of congestion inside the network, i.e. the switch fabric, it is not clear how the request-grant mechanism scales as the network diameter increases. If networks using the mechanism are interconnected without the use of a common request-grant mechanism, congestion may still develop between the networks, and HoL blocking is reintroduced. On the other hand, the use of a common, global request-grant mechanism could add significantly to the transfer latency of traffic in the network.

Adaptive routing [49, 50, 51, 52] and load balancing techniques[14] [53, 54] may help to alleviate congestion by distributing the traffic in the network in a favorable way. Furthermore, if the adaptive routing or load balancing technique is able to do path selection depending on network status [52, 53, 54], congestion trees might be pruned, removed or made less likely to grow. It is important to note, though, that the act of distributing the traffic in the network, e.g. to reroute around the root of a congestion tree, cannot always be successful. This is particularly so if the congestion is caused by too much traffic continuously being sent to one particular destination node, if a destination node is not able to process received traffic fast enough, or in general if the root of a congestion tree is located at the last downstream switch prior to a destination. In such cases, an attempt to remove the congestion by adaptive routing or load balancing will not (necessarily) remove the congestion tree. Instead, the tree might grow wider and/or the root of the tree might move closer to the destination node. The amount of potential HoL blocking in the network could then increase instead of being reduced.

To see how dramatically adaptive routing can change a congestion tree, consider the fat-tree from our previous examples, now with a new root of congestion shown in figure 2.7. In this example, 12 source nodes (orange nodes) are continuously sending traffic to the same destination node (green node). For the time being, the network implements a deterministic shortest-path routing algorithm where all traffic to a given destination node is routed through the same root switch, independent of the location of the source node (for this example we do not need to consider the case where the source and the destination nodes reside inside the same subsection of the fat-tree). The FTREE routing algorithm [34] routes traffic in a similar fashion. Recall that the switches in the top row of the fat-tree are called root switches. As only a single (shortest) path between a root switch and an end node exists in the topology, the complete route from a source node to a destination node is given by the root switch selected to represent the destination. In our example, the 12 orange nodes are all sending to the same destination node, thus their

[14] Adaptive routing and load balancing techniques may share characteristics, as the aim of adaptive routing could be to perform some sort of load balancing in the network, while a load balancing technique could involve some sort of adaptability when it comes to making routing decisions (at the switches or at the end nodes).

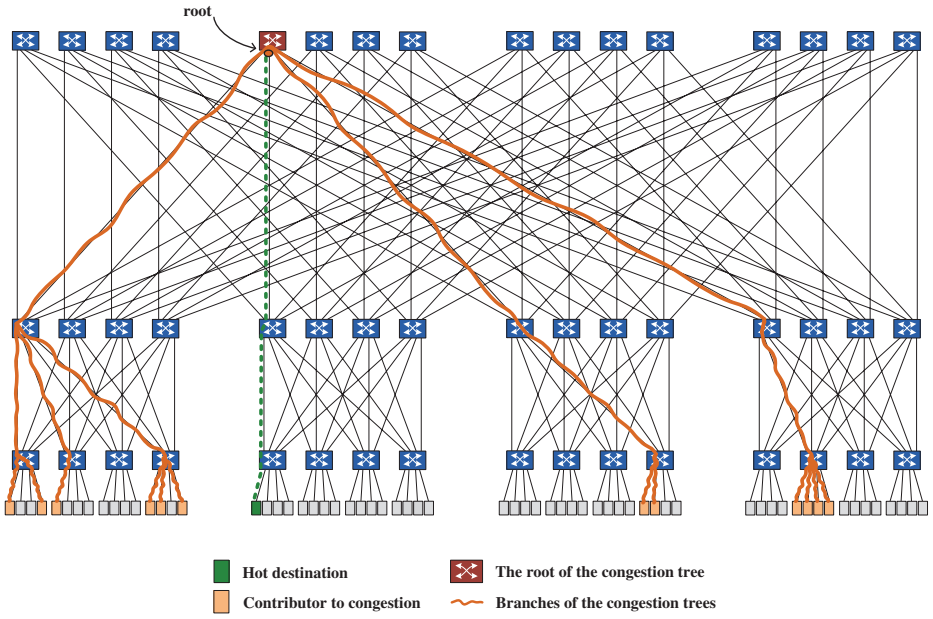


Figure 2.7: A fat-tree with a congestion tree created by 12 source nodes sending continuously to the same destination node using deterministic routing.

traffic will be routed through the same root switch, and the complete traffic pattern is given. The resulting congestion tree is shown in figure 2.7 as orange branches stretching from the root of the congestion tree, located at the indicated root switch corresponding to the hot destination.

Now let us change the routing algorithm used for the scenario shown in figure 2.7. This time, an adaptive routing algorithm is implemented, where each traffic flow is routed shortest-path and deadlock-free by using x links upward, followed by the same number of links downward, to reach the destination node. In our example, $x = 3$ for all the 12 source nodes. At a switch, the adaptive routing algorithm will, as far as possible, forward different traffic flows onto separate links to avoid sharing of resource. The 12 source nodes will still overwhelm the hot destination node, but as the adaptive routing algorithm will do whatever it can to refrain traffic flows from sharing links, the root of the congestion tree will be moved away from the previous location at the root switch in the fat-tree (figure 2.7) to the last downstream switch prior to the destination node. One possible resulting congestion tree is shown in figure 2.8. The adaptive routing algorithm has dramatically widened the congestion tree in an attempt to distribute the traffic in the network, and the amount of potential HoL blocking in the network has increased correspondingly. While it can be argued that the situation in figure 2.8 is not likely to occur in practice, it still proves to show that an adaptive routing algorithm can fail miserably in conducting congestion management when there is no possible route around a hot spot. In such cases, congestion trees are still allowed to grow and cause HoL blocking

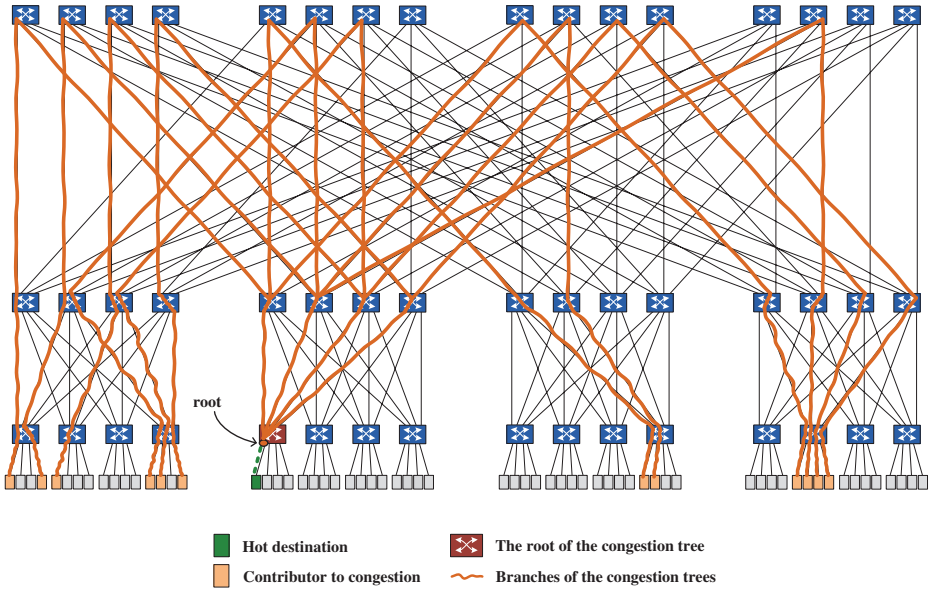


Figure 2.8: A fat-tree with a congestion tree created by 12 source nodes sending continuously to the same destination node using adaptive routing.

in the network.

The two most successful families of congestion management strategies are the *flow isolation* strategies and the *injection throttling* based congestion management mechanisms. In the following sections we will discuss these two families in more detail.

2.3.1 Flow Isolation Strategies

The general idea behind flow isolation strategies is to identify and separate traffic flows, or groups of such flows, to allow for them to travel through (parts of) the network without interfering with each other[15]. The isolation is typically done by allocating private buffers to each flow or group of flows in the switches, while access to the links in the network are still shared.

A well known flow isolation strategy is the *Virtual Output Queues* (VOQ) scheme implemented at the switch level [55, 56] (VOQ_{sw}). The buffer at an input port of a switch is in VOQ_{sw} divided into separate sections, one section per output port. The input port buffer section corresponding to a given output port is then reserved for arriving packets headed for that particular output port. When a packet arrives, the routing decision is made and the next output port selected. Then, the packet is stored in the corresponding buffer section at the input port waiting to be forwarded through the switch fabric. As

[15]In this respect, the before mentioned identification and separation of synchronization messages in [45] is also a flow isolation strategy.

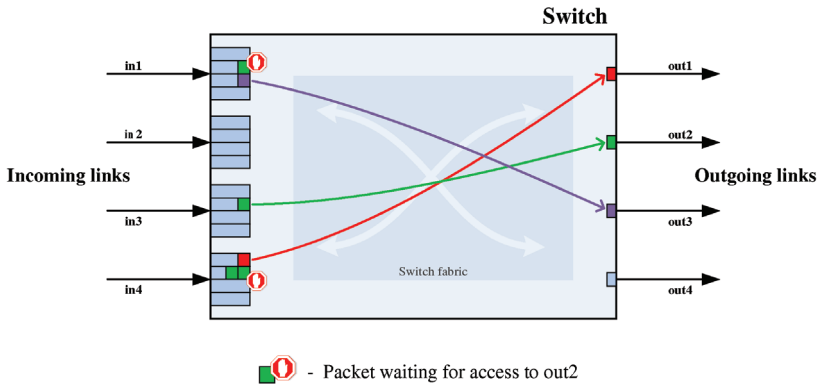


Figure 2.9: A switch implementing VOQ$_{sw}$ prevents low-order HoL blocking.

packets waiting for different output ports never share buffer sections, HoL blocking internal to the switch is not possible. Thus, VOQ$_{sw}$ can be seen as a congestion management mechanism that eliminates low-order HoL blocking.

Figure 2.9 shows our switch scenario from figure 2.6, this time with a switch implementing VOQ$_{sw}$. Now, as the red and the purple packets have been queued according to the VOQ$_{sw}$ scheme, they are no longer blocked by green packets waiting for access to *out*2, and can concurrently be forwarded through the switch fabric to their corresponding output ports, *out*1 and *out*3. The HoL blocking inside the switch is gone. At the same time, the green packet from *in*3 is forwarded to *out*2, while the three other green packets have to wait – like before.

As VOQ$_{sw}$ typically is the internal queuing scheme implemented by InfiniBand switches [41], it is also the natural scheme to consider when we study congestion management in InfiniBand. While VOQ$_{sw}$ successfully avoids low-order HoL blocking, congestion is still allowed to spread between switches, thus high-order HoL blocking may still occur. To handle high-order HoL blocking, the InfiniBand specification now includes an appendix detailing support for *congestion control*, as mentioned in section 1.1. We will get back to the throttling based congestion control mechanism of InfiniBand in section 2.3.2.

The VOQ scheme can also be implemented at the network level (VOQ$_{net}$). Each port at every switch in the whole network then needs a separate, reserved buffer section for every possible destination node. In particular, the buffer sections can never be shared, as traffic headed for one destination would then be able to block traffic headed for another destination. While an implementation of VOQ$_{net}$ would prevent both low- and high-order HoL blocking throughout the network, and arrange for efficient use of the links in the network, the solution does not scale, and the buffer space in the switches will in general be extremely poorly utilized in a network with a large number of destination nodes.

Other queuing schemes that share some characteristics with VOQ include Dynamically Allocated Multi-Queues [57] (a dynamic version of VOQ$_{sw}$), Destination-Based Buffer

Management [58], Dynamic Switch Buffer Management [59] and Output-Based Queue Assignment [60]. While all these techniques address the HoL blocking phenomenon, they are only partly able to prevent HoL blocking from happening. The weakness they bear in common is that packets belonging to cold and warm flows are not generally identified, and thus they might still share queues and cause HoL blocking in the network.

A general idea that allows for flexible flow isolation in the network is the concept of *virtual channels* [40, 44], introduced by Dally and Seitz in 1987. The idea behind this concept is to divide each physical channel into a set of virtual channels (VCs), each VC having its own private buffers. Traffic belonging to different virtual channels use the same physical link, but are mapped to their corresponding buffers at the switches. Thus, a packet travelling along one VC can in general bypass packets from other VCs. In particular, a cold packet could bypass any hot packets to avoid HoL blocking – given that the cold and hot packets reside in different VCs. Indeed, all the above mentioned queuing schemes can be mimicked using VCs. However, as each VC requires its private buffer at each switch port, the scalability issue reappears. For instance, to ensure that no HoL blocking occurs in the network, one VC per destination in the network would be required. Such a number of VCs would be too costly to implement for any large interconnection network. E.g. the InfiniBand specification supports only 16 virtual channels, or *virtual lanes* (VLs) as they are called in the InfiniBand context, while existing InfiniBand hardware typically implements only eight [61].

The vFtree [62] routing algorithm proposed for InfiniBand distributes traffic among available VLs to limit the HoL blocking introduced by a congestion tree to only those traffic flows that share VL with the tree. However, as congestion trees are still allowed to grown inside each VL, the effectiveness of the vFtree algorithm is limited by the number of available VLs and how fortunate the algorithm is in distributing the traffic. The dFtree [63] algorithm improves on vFtree by dynamically detecting congestion in the network and moving contributors to congestion into a separate VL, the *slow lane*, while regular traffic resides in the VL called *fast lane*. The detection and reconfiguration process needed when congestion occurs, however, is not very fast, thus making the dFtree algorithm best suited to handle permanent or long-lasting congestion.

All the flow isolation techniques mentioned this far, with the exception of the dFtree algorithm, have one weakness in common: They do not dynamically detect and separate flows contributing to congestion, thus their success in avoiding HoL blocking greatly depends on the number of available queues per port. By contrast, a family of flow isolation strategies we will refer to as *Hot-Flow Dynamic Isolation* (HFDI) techniques, detects roots of congestion trees as they are about to form, and separates flows contributing to congestion into special, dynamically assigned queues. Flows not contributing to congestion reside in the regular queues of a switch and experience no (or little) HoL blocking. An early HFDI-based technique was proposed by Duato et al. in 2005 [64]; Regional Explicit Congestion Notification (RECN), designed for networks implementing source routing. Other HFDI-based techniques include Regional Explicit Congestion Notification-Distributed Deallocation [65] (RECN-DD), Regional Explicit Congestion Notification-Input Queued [66] (RECN-IQ), and Flow-Based Implicit Congestion Management [67, 68] (FBICM), whereof FBICM implements HFDI for networks with

deterministic incremental routing.

HFDI techniques monitor the buffer occupancy at switch ports to detect congestion. If the occupancy at a port exceeds a given *detection threshold*, a root of congestion is detected, and a separate *congested flow queue*[16] (CFQ) is allocated and associated with the root. Packets contributing to the detected congestion are then stored in the CFQ, while other packets are stored in the regular buffer at the port, i.e. in the *normal flow queue* (NFQ). By allowing packets in the NFQ to bypass packets stored in the CFQ, low-order HoL blocking is prevented.

When a root of congestion is detected and a CFQ allocated, information about the corresponding root, the roots location, and information to identify packets headed for the root, is stored in a control memory linked to the CFQ. The content of this control memory is propagated to the next upstream switch if the occupancy of the CFQ exceeds a *propagation threshold*. The purpose of this propagation is to notify an upstream switch about a growing congestion tree before the tree itself is able to reach the switch. When the upstream switch is notified about a downstream congestion tree, the upstream switch can (when needed) allocate its own separate CFQ for the given tree and store packets headed for the root of that tree in the CFQ. By following this scheme, packets belonging to different congestion trees are separated in corresponding CFQs throughout the network, while packets not contributing to congestion remains in the NFQs. Thus, high-order HoL blocking is prevented. A CFQ is deallocated as soon as the congestion tree going through the CFQ vanishes, and later reallocated if a new congestion tree grows.

Figure 2.10 shows a switch implementing HFDI. Each input port is equipped with two CFQs in addition to a NFQ. In this particular scenario, a root of congestion has been identified and associated with $out3$. The ports $in1$ and $in2$ have allocated one CFQ each, as they both have traffic headed for $out3$ (the green packets). The port $in3$, on the other hand, has both its CFQs unallocated, as $in3$ has no packets headed for $out3$, and no other root of congestion is detected locally nor reported from downstream switches. The occupancy of the allocated CFQ at $in1$ is above the propagation threshold, thus information about the root associated with $out3$ is propagated upstream. Since $in2$ is currently forwarding traffic from its CFQ to $out3$, the green packets at $in1$ have to wait. However, as the green packets at $in1$ is waiting in a CFQ, HoL blocking is avoided, and $in1$ can forward the red packet from the NFQ to $out2$. $in3$ is at the same time forwarding its purple packet to $out1$. The root of congestion is in this scenario located locally. The scenario would be similar, however, if the root of congestion was located at a downstream switch of $out3$, and the green packets were headed for the root of the congestion tree. If we in that case assume that the red packet at $in1$ is headed for $out3$, but not the root of the congestion tree, the red packet would still reside in the NFQ of $in1$ and can be forwarded prior to the green packet in the CFQ of $in2$. Thus high-order HoL blocking is avoided.

The HFDI techniques are able to react fast and locally, and prevent any significant HoL blocking[17] by isolating the branches of the congestion trees in the CFQs. The short

[16] The congested flow queues are called Set-Aside-Queues (SAQs) in RECN.

[17] Depending on the exact implementation of HFDI and the buffer management in the switches, short-lived HoL blocking might still occur in some cases.

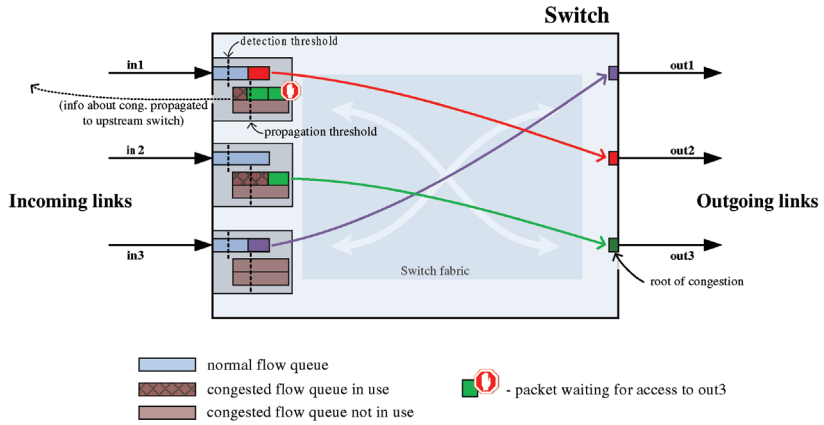


Figure 2.10: A switch implementing HFDI with one detected root of congestion.

reaction time of the HFDI techniques, and in particularly the implementation of FBICM, is an important source of inspiration when we address research question RQ4, presented in paper V and paper IV. Recall that this research question seeks to deal with the delay in reaction time imposed by the feedback control loop of an injection throttling based congestion management technique like the one specified for InfiniBand.

On the other hand, the fact that a HFDI technique does not remove a congestion tree from the network poses a challenge. The number of congestion trees a port can handle is strongly tied to the number of available CFQs. If a port has x available CFQs, and more than x congestion trees grow through the port, at least one of the trees will grow into the NFQ, and HoL blocking is reintroduced. In a network with long-lived congestion trees, this flaw could be particularly challenging, as the likelihood of several trees growing into the same port increases. Then only a few large congestion trees could be enough to exhaust all the CFQs of a large number of ports in the network.

2.3.2 Injection Throttling Based Congestion Management

An injection throttling based congestion management mechanism aims to avoid HoL blocking in a network by removing the real cause of the problem, the congestion trees. To resolve congestion, the source nodes first need to be notified, in an implicit or explicit way, about congestion in the network. Then, by adjusting their injection rates accordingly, the source nodes can remove the congestion trees.

This detect-notify-react scheme can be implemented in a number of ways, and numerous proposals exist. All the nodes in the network could gather and share, possibly by the use of a separate control network, relevant information about the global status of the network, and use this global network view to tune injection rates and avoid congestion [69]. Another option is to only notify a source node directly connected to a switch about locally detected congestion at the switch, though differentiating between congestion where the

local node is contributing, and congestion created by others [70]. In a wormhole network, blocked worms, possibly padded, can be used to detect congestion (or deadlocks) if the tail of the worm still resides inside the source node when the blocking occurs [71].

A subclass of the throttling strategies relies on the switches to detect congestion and notify the source nodes by the use of explicit congestion notifications (ECN). An ECN can be piggybacked to a packet contributing to congestion in order to notify the destination node, whereupon the destination node notifies the corresponding source node about the situation. This forward explicit notification approach is taken by InfiniBand, where packets are *marked* as contributing to congestion by setting a specific bit in the packet headers. On the contrary, the switches could take a backward explicit notification approach where switches send ECNs directly back to the source nodes. Such an approach is taken by the Data Center Bridging standard [72]. Furthermore, in the context of ECN strategies, work has been conducted on what information to include in the ECN, e.g. information about the severity of the congestion could be included; where and when to mark the packets, e.g. packets can be marked both at the input and the output buffers; and on how to design the source response function at the source node, i.e. how a node reduces and increases the injection rate of packets as a response to the ECNs [73, 74, 75, 76].

The InfiniBand congestion control mechanism is given special attention in this thesis, and as such, we will give a detailed description of the mechanism below. First, however, we will recall the general challenge related to the feedback loop of ECN-based injection throttling strategies; it takes time to propagate information about congestion from a switch to an end node. This delay imposed by the feedback loop not only challenges the effectiveness of injection throttling as a congestion management mechanism, but it could also have implications on the stability of traffic in the network. When congestion occurs, it will take time from congestion is detected at a switch until the contributors to congestion receive the notifications, and furthermore, more time will pass before the effect of lowering an injection rate at a source reaches the root of the congestion tree. During this total period of time, HoL blocking might take place in the network. Thus, when a source node starts to receive ECNs, ideally the node should immediately halt the traffic contributing to congestion just long enough to prune the congestion tree all the way down to the root – but not longer, as a further halt in traffic could result in unfortunate underutilization of the scarce resources at the root of the tree. Then, when the tree is pruned, the source node should preferably inject traffic at a rate corresponding to this traffic flow's fair share of the resources at the root of congestion. Unfortunately, this rate is neither static nor accurately known – because of the delay of the feedback loop.

Furthermore, while congestion is present, the delay created by the feedback loop makes the source nodes continue to adjust their injection rates based upon information about a previous network state. In an unfortunate situation, a source node could then end up throttling a traffic flow that is no longer contributing to congestion, as the congestion tree the flow previously was contributing to has been resolved, e.g. by a general change in the traffic pattern in the network. To minimize the effect of such dangling assumptions about congestion, it is important that source nodes increase their injection rates rapidly again as the reception of ECNs is decreasing. That is, a source node should in some situations quickly increase the injection rate, while in other situations just as quickly decrease the

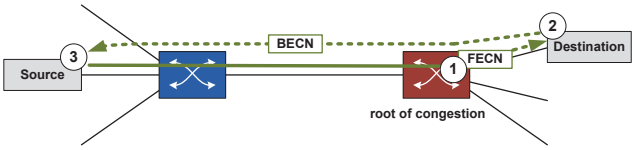


Figure 2.11: Congestion control in InfiniBand.

injection rate again. An unfortunate side effect of this, on the one hand desired, behavior at the source nodes, is the oscillation in individual traffic flow throughputs observed in [13, 77].

Altogether, ECN-based injection throttling aims to reduce HoL blocking in the network by removing the cause of the problem, the congestion tree. This group of techniques can in general be implemented with less hardware at the switches, compared to HFDI, and does not suffer from a scalability issue when it comes to the number of concurrent congestion trees supported before HoL blocking is reintroduced. On the other hand, the general effectiveness of an ECN-based injection throttling mechanism is challenged by the delay imposed by the feedback loop, and traffic oscillation might be introduced.

InfiniBand Congestion Control[18]

The InfiniBand (IB) Congestion Control (CC) mechanism [78], is based on a closed-loop feedback control system, similar to the feedback loop described in the previous section. An overview of the IB CC feedback loop is shown in figure 2.11. Here, a source node is contributing to congestion detected by the switch to the right. That is, the root of the congestion tree is located at this switch. A packet from the source node going through the root of the congestion tree is then marked at the switch as a contributor to congestion by setting a specific bit in the packet header, the *Forward Explicit Congestion Notification* (FECN) bit (figure 2.11, ①). An explicit congestion notification is then carried through to the destination by this bit. The destination registers the FECN bit, and returns a packet with the *Backward Explicit Congestion Notification* (BECN) bit set back to the source (figure 2.11, ②). By the reception of the BECN bit (figure 2.11, ③), the source node knows that the corresponding flow is contributing to congestion and reduces the injection rate of the flow accordingly.

The exact behaviour of the IB CC mechanism depends upon the values of a set of CC parameters governed by a *Congestion Control Manager*. These parameters determine characteristics like when switches detect congestion, at what rate the switches will notify destination nodes using the FECN bit, and how much and for how long a source node contributing to congestion will reduce its injection rate. Appropriately set, these parameters should enable the network to resolve congestion and avoiding HoL blocking, while still utilizing the network resources at the root of the congestion tree efficiently. The following

[18]This section is a slightly modified version of a similar section existing in the research papers (of this thesis) specifically addressing InfiniBand congestion control.

Channel Adapter		Switch
CongestionControlTable	(CCT)	*Threshold*
CCTI	(CCT index)	*Victim_Mask*
CCTI_Increase		*Packet_Size*
CCTI_Limit		*Marking_Rate*
CCTI_Timer		
CCTI_Min		

Table 2.1: InfiniBand CC channel adapter and switch parameters.

subsections detail the IB CC features at a switch and at a channel adapter (CA) (e.g. an end node), and explain the corresponding CC parameters (summarized in table 2.1).

IB CC features at a Switch:
The switches are responsible for detecting congestion and notifying the destination nodes using the FECN bit. A switch detects congestion on a given *port* and a given *Virtual Lane* (Port VL) depending on a *threshold* parameter. If the threshold is crossed, a port may enter the Port VL congestion state, which again may lead to FECN marking of packets.

The *threshold*, represented by a weight ranging from 0 to 15 in value, is the same for all VLs on a given port, but could be set to a different level for each port. A weight of 0 indicates that no packets should be marked, while the values 1 through 15 represent a uniformly decreasing value of the threshold. That is, a value of 1 indicates a high threshold with high possibility of congestion spreading, caused by Port VLs moving into the congestion state too late. A value of 15 on the other hand indicates a low threshold with a corresponding low possibility of congestion spreading, but at the cost of a higher probability for a Port VL to move into the congestion state even when the switch is not really congested. The exact implementation of the threshold depends on the switch architecture and is left to the designer of the switch.

A Port VL may enter the congestion state if the threshold is crossed and it is the root of congestion, i.e. the Port VL has available credits to output data. If the Port VL has no available credits, it is considered to be a victim of congestion and shall not enter the congestion state unless a specific *Victim_Mask* is set for the port. The *Victim_Mask* is typically set for a switch port connecting a CA. A CA that is not able to process received packets fast enough will not consider itself to be a root of congestion even if a congestion tree then builds up with the CA as the root. In this special case the Port VL at the switch connecting the CA should consider itself to be the root of congestion, even if it is actually a victim, and move into the congestion state.

When a Port VL is in the congestion state, its packets are eligible for FECN marking. A packet will then get the FECN bit set depending on two CC parameters at the switch, the *Packet_Size* and the *Marking_Rate*. Packets with a size smaller than the *Packet_Size* will not get the FECN bit set. The *Marking_Rate* sets the mean number of eligible packets sent between packets actually being marked. With both the *Packet_Size* and the *Marking_Rate* set to 0, all packets should get the FECN bit set while a Port VL

is in the congestion state.

IB CC features at a Channel Adapter:
When a destination CA receives a packet with a FECN bit set, the CA should as quickly as possible notify the source of the packet about the congestion[19]. As earlier mentioned, this is done by returning a packet with the BECN bit set back to the source. The packet with the BECN bit could either be an acknowledgement packet (ACK) for a reliable connection or an explicit *congestion notification packet* (CNP). In either case it is important that the ACK or the CNP is sent to the source as soon as possible to ensure a fast response to the congestion.

When a source CA receives a packet with the BECN bit set, the CA lowers the injection rate of the corresponding traffic flow. To determine how much and for how long the injection rate should be reduced, the CA uses a *Congestion Control Table* (CCT) and a set of CC parameters. The CCT holds *injection rate delay* (IRD) values that define the delay between consecutive packets sent by a particular flow (the IRD calculation being relative to the packet length). Each flow with CC activated holds an index into the CCT, the $CCTI$. When a new BECN arrives, the $CCTI$ of the flow is increased by $CCTI\_Increase$. The CCT is usually populated in such a way that a larger index yields a larger IRD. Then, consecutive BECNs increase the IRD which again decreases the injection rate. The upper bound of the $CCTI$ is given by $CCTI\_Limit$.

To increase the injection rate again, the CA relies on a $CCTI\_Timer$, maintained separately for each Service Level (SL) of a port. Each time the timer expires, the $CCTI$ is decremented by one for all associated flows. When the $CCTI$ of a flow reaches zero, the flow no longer experience any IRD. Furthermore, each SL also has a $CCTI\_Min$ parameter. By using the $CCTI\_Min$ it is possible to impose a minimum IRD to the SL, as the $CCTI$ should never be reduced below the $CCTI\_Min$.

The IB CC can operate either at the SL or at the Queue Pair[20] (QP) level at an CA. Any lowering of the injection rate as a result of BECN reception then affects the whole SL or the single QP depending on the level of CC operation. While operating at the SL level may require less resources at the CA than operating at the QP level, choosing the SL level will have an impact on both fairness and performance. The reason is that a single traffic flow contributing to congestion will lower the injection rate of all traffic flows within the same SL at the CA. This could include traffic flows not contributing to the congestion at all as they are not going through the root of the congestion tree, but headed for other parts of the network.

[19]There are three exceptions. The FECN bit in a multicast packet, acknowledgement packet or congestion notification packet should be ignored. That is, no congestion notification is sent back to the source in these three cases.

[20]A Queue Pair is an endpoint of a communication channel used by applications to communicate using InfiniBand's messaging service.

Chapter 3

Summary of Research Papers

This chapter presents the six research papers written as part of this thesis. An extended abstract is given for each paper, summarizing their contributions. The link between the papers and the research questions in chapter 1.1 is given below. Five of the papers have been published at peer-reviewed international conferences, while the last paper has been submitted to the IEEE Transactions on Parallel and Distributed Systems journal. Some ideas related to future work are presented at the end of the chapter.

Paper I [18] gives a detailed description of the simulator extensions developed by the author to support the research conducted as part of this thesis. As such, the paper is not directly connected to any of our research questions, but describes a tool used to study injection throttling based congestion management in the context of InfiniBand. In particular, the simulator has been the main tool used to produce the results presented in Paper II [19] and Paper IV [20].

In Paper II [19] we address research question two[1]: *How should congestion detection and notification be carried out at a forwarding node implementing an injection throttling based congestion management mechanism, like IB CC, to ensure fair treatment of the contributors to congestion?*. We show that the threshold mechanism used to detect congestion and initiate packet marking at a switch leads to unfair treatment of the contributors to congestion if the mechanism is implemented by a single threshold, even if the switch implements a generally fair arbitration algorithm. To solve this unfairness issue, we propose to add hysteresis to the congestion detection by implementing the threshold mechanism by the use of a lower and upper threshold. In the paper, we demonstrate how fairness among contributors to congestion is reintroduced by our proposed hysteresis. Furthermore, we demonstrate how injection throttling as specified for InfiniBand can solve *the parking lot problem* [19, 79] if the congestion control mechanism is properly configured.

Paper III [13] targets research question one: *How can the IB CC parameters be tuned to ensure efficient behavior of the IB CC mechanism for a given scenario, and how do different parameters impact the network performance?*. By systematically running experiments on a small InfiniBand cluster, we show how performance of a congested network varies as a function of IB CC parameter values. As part of this studies, we introduce the *treatment variation variable* to indicate unfairness among contributors to congestion.

[1]To maintain a logical ordering of the papers, we present the paper that address research question two before the paper that address research question one.

When a subset of IB CC parameter values is identified as leading to the desired IB CC behavior, we use these values to demonstrate that IB CC capable hardware is able to efficiently remove the congestion tree and avoid HoL blocking, while still utilizing the scarce resources at the root of the congestion tree in a fair way.

In Paper IV [20] we build on the knowledge from Paper III to address research question three: *What is the scope of an injection throttling based congestion management mechanism like IB CC, and how does such a mechanism behave depending on traffic dynamics in the network and the lifetime of the hotspots, and in the case of IB CC, how robust are the IB CC parameters with respect to such changing traffic dynamics?* By using a single set of IB CC parameter values we study how IB CC performs as the traffic dynamics increases in a fat-tree topology with 648 end nodes. An abstract classification scheme for congestion trees of varying degree of dynamics is introduced, and should be seen as one of the contributions in the paper. The paper shows that IB CC, even when using a single set of parameter values, outperforms a network running without CC for a large range of congested situations. However, as the lifetime of the hotspots in the network decreases, the benefit from enabling IB CC in the network becomes limited.

In Paper V [21] and Paper VI [22] we study how injection throttling can be combined with a HFDI technique to address research question four: *How can we extend a throttling based congestion management mechanism, like IB CC, to overcome the challenges imposed by the feedback control loop by taking immediate local action at a forwarding node, while we wait for the throttling mechanism to have effect?*. In [21] we propose the CCFIT congestion management mechanism. CCFIT combines an IB CC inspired injection throttling mechanism with a FBICM inspired HFDI technique for input buffered switches. In [22] we present EcoCC, a comprehensive redesign of the CCFIT mechanism to utilize switch architectures supporting virtual output queuing, VOQ_{sw}. The VOQ_{sw}'s ability to eliminate low-order HoL blocking allows EcoCC to focus solely on high-order HoL blocking. By extracting the best of two worlds, injection throttling and HFDI, without inheriting their respective weaknesses, CCFit and EcoCC are able to in an efficient, fair and scalable way conduct congestion management, outperforming injection throttling and HFDI techniques as separate approaches.

While the research Papers I, III and IV specifically targets the InfiniBand platform, research Paper II, addressing congestion detection and notification, should be seen as a contribution for general injection throttling based congestion management techniques. Thus, the knowledge from Paper II is also incorporated into the CCFIT and EcoCC mechanisms. CCFIT and EcoCC are proposed as general congestion management mechanisms for lossless interconnection networks, though inspired by IB CC and FBICM.

3.1 Paper I: InfiniBand Congestion Control, Modelling and Validation [18]

Access to large InfiniBand clusters with support for congestion control is not easily gained. In addition, a simulator model provides flexibility when it comes to experimenting with functionality yet not available or accessible in hardware. Therefore, a simulator model to study IB CC and gain knowledge about the behavior and characteristics of an injection throttling based CC mechanism in general would be a valuable tool. In this paper we present our CC capable IB model implemented in the OMNeT++ environment [17].

The CC capable IB model is based on the IB model made available to the OMNeT++ community by Mellanox Technologies Ltd in 2007/2008. We have ported the original IB model to the OMNeT++ 4 environment, made several bug fixes, and added some general extensions to make the model more suitable for our CC studies. Furthermore, the CC functionality specified for IB in release 1.2.1 [78] is carefully implemented by the use of the simple and compound module concepts of OMNeT++. In particular, support for all the IB CC parameters have been implemented according to the IB specification.

The high level of detail required to precisely capture all aspects of the IB CC mechanism and the corresponding dynamics imposed by injection throttling in a network, could be a challenge for the scalability and usefulness of a simulator. Our experience shows that the OMNeT++ environment and the IB model tackles the scalability issue quite well.

Furthermore, an important part of implementing a simulator is to validate it properly to ensure that the simulation results are trustworthy. Our IB CC model has been carefully validated against the hardware implementation of CC in the Mellanox ConnectX Host Channel Adapters [80] and the Mellanox InfiniScale IV switches [61]. Validation results presented in this paper shows that our CC capable IB model is able to closely resemble the CC capable hardware from Mellanox.

3.2 Paper II: On the Relation Between Congestion Control, Switch Arbitration and Fairness [19]

While the InfiniBand standard describes the functionality of IB CC, some freedom is provided when it comes to implementing the concept. Several design decisions are left to the hardware designer, as standards typically do. As we further elaborate on in this paper, one must be cautious when making these design decisions not to introduce unfairness. Fairness is an important property of any interconnection network. Different traffic flows having the same priority should all get equal access to shared resources in the network.

In this paper we study by simulation the relationship between injection throttling based congestion control, switch arbitration, and fairness. More specifically, we look at fairness in two different situations: First, we look at fairness among different traffic flows arriving at a hot spot switch on different input ports, as CC is turned on (Type I). We demonstrate that a straightforward implementation of an (IB) injection throttling mechanism using a single threshold results in unfairness, even if the switches implement a fair arbitration scheme like *round robin* [2]. Furthermore, we show that this unfairness

is unstable in the sense that the distribution of bandwidth to the different flows depends on the utilization of the different flows that just happened to prevail at the instance of time when congestion occurred.

The second type of fairness we study is the fairness among traffic flows at a switch where some flows are exclusive users of their input ports while other flows are sharing an input port (Type II). This fairness, or lack of it, is related to the parking lot problem; An arbiter at a switch will in general not differentiate between two separate traffic flows f_1 and f_2 arriving at the same input port ip_1, both headed for the output port op. When a third flow f_3 arrives at a different input port ip_2, but is headed for the same output port, op, a fair round robin arbitration will alternate between forwarding packets from ip_1 and ip_2. Such an arbitration, however, will give half the bandwidth of op to f_3, while f_1 and f_2 each will be left with only 1/4 of the op bandwidth. The three traffic flows are not treated in a fair way. The result of this type of unfairness could be dramatic in a network with a large diameter and/or if switches with many ports are used.

In this paper, we propose to solve Type I unfairness by adding hysteresis to the congestion detection mechanism, and we demonstrate that the needed hysteresis successfully can be implemented by means of two thresholds, one upper and one lower, where the distance between the two is at least one MTU (maximum transmission unit). Furthermore, the paper shows that Type II unfairness can be avoided if the above mentioned hysteresis is implemented and an aggressive marking rate for packets contributing to congestion is applied. That is, properly configured, IB CC is able to solve the parking lot problem.

3.3 Paper III: First Experiences with Congestion Control in InfiniBand Hardware [13]

The InfiniBand CC concept is rich in the way that it specifies a set of parameters that can be tuned in order to achieve effective CC. There is, however, limited experience with the InfiniBand CC mechanism in general, and in particular knowledge about how to tune the different CC parameters is needed. In this paper we present the first experiences with CC capable InfiniBand hardware in a small 2-switch/7-end-node cluster. By conducting extensive testing on a selection of the CC parameters, we explore the parameter space, and show that if properly configured, IB CC capable hardware is able to efficiently remove the congestion tree and avoid HoL blocking, while still utilizing the resources at the root of the congestion tree.

This paper consists of three main sections. First we study, by the use of an (by us) identified set of IB CC parameter values and synthetic traffic patterns, how different traffic flows are affected by properly configured IB CC. In particular we study how enabling IB CC affects both the victim and the contributors to congestion in a scenario where four contributors are creating a congestion tree causing HoL blocking of a single victim (Scenario 1). Experiments show that activating CC in this case results in an order of magnitude improvement in throughput for the victim flow, independent of packet size, and that the throughput achieved by the victim flow when CC is enabled coincides with the throughput the same flow achieves in a network with no congestion. The contributors to

congestion are still able to utilize the resources at the root of the congestion tree (and the parking lot problem is solved), but some oscillation is introduced as the contributors are trying to settle for their fair share of the bottleneck resource. Furthermore, a scenario with only contributors to congestion, and no potential victim flows, is investigated (Scenario 2). Note that in such a situation there is no potential benefit from enabling CC. In this scenario, the contributors to congestion experience a slight drop in performance (3.5%), while the oscillation introduced is evident.

In the next main section of the paper, we continue to study Scenario 1 as described above, but this time the victim flow is exchanged with traffic produced by the HPC Challenge benchmark [81, 82]. By using the benchmark together with a synthetically generated hotspot, we imitate the network conditions an application, here represented by the benchmark, might experience if the network is shared with another application creating congestion. The improvement in performance experienced by the benchmark as CC is enabled depends on the communication sensitivity of the different applications included in the benchmark. We observe performance improvement ranging from 1.7% to 248.1%. Again, the observed performance of the benchmark in a congested scenario with CC enabled is very close to the performance we observed in a scenario without congestion.

In the last section of the paper, we elaborate on how the IB CC parameter values we use in the paper were identified through extensive testing, with a particular focus on the *threshold* parameter and the relation between the *Marking_Rate* at a switch and the *CCTI_Timer* implemented by an end node. Furthermore, the *treatment variation variable* is introduced as a tool to indicate unfairness among contributors to congestion. Our studies show that even though the performance of IB CC is sensitive to the IB CC parameter values, it is possible to find a parameter value "sweet spot" for our test scenarios.

3.4 Paper IV: Exploring the Scope of the InfiniBand Congestion Control Mechanism [20]

Even though it has been shown that the InfiniBand CC mechanism, properly tuned, is able to improve both throughput and fairness in an interconnection network [13], it has been questioned whether the mechanism is fast enough to keep up with dynamic network traffic, and if a given set of parameter values for a topology is robust when it comes to different traffic patterns or if the parameters need to be tuned depending on the applications in use. Furthermore, uncertainty has lingered as to whether CC may actually be harmful in some scenarios. In this paper we address these questions by studying the performance of the IB CC mechanism, using a single set of IB CC parameter values, as the communication pattern in the network gradually changes from a static to a dynamic scenario in a fat-tree with 648 end nodes. The studies are conducted by means of simulation.

To be able to conduct a systematic study of how the IB CC mechanism performs as the traffic dynamic increases, we start by introducing an abstract classification scheme for congestion trees. According to this scheme, congestion trees are divided into three nonexclusive categories, based on the nature of the trees' (main) contributors and how dynamic they are. Starting with the least dynamic category, a congestion tree can be

silent, *windy*, or even *moving* while wind is blowing through the tree's crown.

A silent congestion tree is created by a set of static contributors to congestion permanently sending traffic to the same hotspot. In such a scenario, a congestion tree will grow with branches always following the same set of paths in the network. The tree's braches are not blowing in the wind, and as such the congestion tree is silent. On the other hand, contributors to a windy congestion tree is alternating between sending traffic to a permanent hotspot and sending traffic elsewhere in the network. Thus, depending on how fast the contributors alternate, the branches of the congestion tree may move (or grow and get pruned along different paths) by the traffic pattern itself, resembling wind blowing through the crown of the congestion tree. A moving congestion tree is a silent or windy congestion tree where the hotspot at given times move, that is, where the contributors to congestion at times change focus from one hotspot to another.

Even though the three categories of congestion trees are nonexclusive and a traffic pattern in a network naturally will create a diverse and stormy forest of congestion trees, by controlling the main creation of congestion trees we were able to in a systematic way measure the performance of the IB CC mechanism as the traffic dynamics increases in the network. Our studies show that IB CC, using a single set of IB CC parameter values, is quite capable of handling silent and windy congestion trees in a fat-tree topology with 648 end nodes, showing up to a seventeen-fold increase in throughput over a network where CC is not enabled. However, as the congestion trees start to move, the IB CC mechanism is challenged, and as the hotspot lifetime reaches 1ms the benefit from enabling IB CC in the network is more or less gone. Then again, the only adverse effect we ever registered when enabling IB CC was a negligible decrease in throughput for the contributors to congestion.

3.5 Paper V: Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks [21]

Existing congestion management mechanisms in interconnection networks can be divided into two general approaches. One approach is to throttle traffic injection at the sources that contribute to congestion, while the other approach is to isolate the congested traffic in specially designated resources. These two approaches have different, but non-overlapping weaknesses. An injection throttling mechanism is able to remove the congestion tree, and by that the introduced HoL blocking, but the mechanism has a challenge when it comes to reaction time. It operates behind schedule. A mechanism based on congested-flow isolation, on the other hand, reacts immediately and locally at a switch, but faces scalability issues as the number of congestion trees increases. A switch may run out of specially designated resources to handle congestion. In addition, there is another less obvious difference between the two mechanisms: A throttling mechanism has the potential of improving fairness in the network by solving the known parking lot problem.

Observing that the two approaches have non-overlapping weaknesses, we were inspired

to overcome the shortcoming of a throttling-based congestion management mechanism like IB CC by combining it with a congested-flow isolation mechanism like FBICM. At the same time, such a combination has the potential of overcoming the shortcoming of a congested-flow isolation mechanism, as the throttling could remove the congestion trees before the flow isolation mechanism runs out of the specially designated resources at the switches.

In this paper we present Combined Congested-Flow Isolation and Throttling (CCFIT), a novel mechanism which combines the two abovementioned approaches, injection throttling and flow isolation. The paper gives a detailed description of the switch and end node architecture, as well as their specific operation, to support CCFIT.

Through simulation studies we first demonstrate the respective flaws of injection throttling and flow isolation. Then, we show that CCFIT is able to overcome these flaws. More specifically, we show that CCFIT is able to quickly remove HoL-blocking, assure scalability by removing the congestion trees, improve fairness in the network, and last but not least, CCFIT achieves an allover higher throughput than injection throttling and congested-flow isolation do as standalone concepts.

3.6 Paper VI: Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks [22]

As shown in paper V [21], the CCFIT mechanism is able to provide efficient, scalable and fair congestion management, and is in particular able to outperform congestion management implemented by means of injection throttling or flow isolation alone. However, as CCFIT was design with input buffered switches in mind, the mechanism needs to handle both low- and high-order HoL blocking. On the contrary, state-of-the-art HPC switch architectures typically applies a VOQ-based buffer organization, e.g. implemented by the use of memory with several read ports at each switch port. A VOQ-based switch is then by its own internal structure able to eliminate low-order HoL blocking. This fact encouraged us to rethink the CCFIT mechanism in a context where the congestion management mechanism needs to consider high-order HoL blocking only. The result is the new congestion management mechanism presented in this paper.

In this paper we propose a new Efficient and cost-effective Congestion Control (EcoCC) mechanism. EcoCC combines injection throttling and congested-flow isolation in a way similar to CCFIT, but by taking advantage of a modern VOQ-based switch architecture several and significant improvements over CCFIT were made possible. In particular, as the VOQ scheme eliminates low-order HoL blocking, the specially designated resources to handle packets contributing to congestion can be reserved for packets contributing to high-order HoL blocking only. In addition, the VOQ characteristic of the switch makes it possible for EcoCC to do exact congestion detection, while CCFIT's detection mechanism is of probabilistic nature. Furthermore, the design of EcoCC makes it possible to detect congestion using only two thresholds, while five is needed for CCFIT, thereby

further reducing the implementation complexity. The paper includes a description of the architecture of a switch and end node supporting EcoCC.

The EcoCC mechanism is evaluated by simulating a combination of synthetic and trace-based traffic patterns in 3, 4, and 5 stage fat-tree topologies with 64, 256, and 1024 end nodes, respectively (in the case of trace-based traffic patterns, only the 4 stage fat-tree was used). The results show that the EcoCC mechanism is able to show significant improvements over injection throttling and flow isolation techniques as standalone concepts, and in particular that EcoCC operates very close to the theoretical maximum in most cases (represented by VOQ_{net}). In addition, in an indirect comparison between EcoCC and CCFIT, the improvements EcoCC achieves over the standalone concepts and a network without congestion management, are higher than those made by CCFIT in (almost) all cases, despite the more sophisticated switch architecture used when EcoCC is compared against the standalone concepts; VOQ switches avoiding low-order HoL blocking improves the base network performance. Thus, it can be concluded that EcoCC is able to more accurately locate congestion roots and more efficiently use resources to isolate contributors to congestion, compared to its counterpart CCFIT. Furthermore, EcoCC requires less and simpler hardware to implement than CCFIT.

3.7 Future Work

Throughout this thesis we have shed some light on the challenges related to congestion and congestion spreading in a lossless interconnection network, and new insight and new proposals in the context of congestion management in such networks have been put forward and evaluated, primarily in the attached research papers. Nevertheless, more interesting challenges lay ahead, whereof a few are listed below.

While we have shown that IB CC is able to successfully conduct congestion management in a large range of scenarios, and that promising discoveries have been made in relation to IB CC parameter value robustness and fat-trees, the IB CC mechanism is still not fully understood. It is of particular interest to extend the investigations started in this thesis with studies of IB CC applied in popular direct topologies like Tori and Meshes, and ultimately provide general guidelines on how to configure the IB CC mechanism. Furthermore, access to a large IB CC capable cluster to confirm our findings and extend our knowledge about IB CC's efficiency as a congestion management mechanism is still pursued.

IEEE have recently through the Data Center Bridging (DCB) task group supplemented the IEEE 802.1 family of standards with an 802.1Qau specification detailing support for congestion management in the context of Ethernet [72, 83]. The specification describes an injection throttling based congestion management mechanisms baring similarities to IB CC, but with two notable differences that could potentially improve the efficiency of the throttling mechanism: In the DCB mechanism the congestion notification messages are sent directly from a switch back to a contributing source. That is, the notifications are not sent via the destination node, as in IB CC, thus the feedback loop is shortened. Furthermore, the DCB notification message includes a quantized feedback about congestion,

and by that carries information about the severity of congestion back to the contributor. This information could be utilized by a contributor when it is adjusting its injection rate. It would be interesting to study how these two potential improvements of DCB throttling over IB CC actually play out in practice and how they influence the efficiency of the throttling mechanism in general.

Adaptive routing has desirable characteristics when it comes to e.g. load balancing and the flexibility to dynamically reroute around a faulty region in the network or a switch being switched off to save power. Then again, as we have shown in chapter 2.3, adaptive routing cannot always successfully conduct load balancing to alleviate congestion, but might in some cases actually make a situation of congestion worse. A proper congestion management mechanism is therefore still needed. However, just applying a well known HFDI or injection throttling based congestion management mechanism to a network using adapting routing will not work, as the adaptivity of the routing algorithm disturbs the fundamental assumptions made by the congestion management mechanisms. HFDI mechanisms, like FBICM and the corresponding parts of CCFIT and EcoCC, relies on a determinist routing algorithm to be able to decide at an upstream switch if a packet further downstream will pass through the root of a congestion tree, and by that, if the packet should be stored in a congested flow queue or in a normal flow queue. On the other hand, when it comes to injection throttling, it is not clear how the fact that adaptive routing detaches a packet from a given path influences the marking of packets and the injection throttling scheme; a source node will in general not differentiate between packets belonging to the same traffic flow where some of the packets are contributing to a congestion tree, and thus should have been throttled, while other packets are taking a different uncongested path through the network, and by that should not be throttled. In general, how to combine adaptive routing with a true congestion management mechanism is not well understood and needs to be studied.

Last, a switch implementing a HFDI-based congestion management technique needs to arbitrate between packets in normal flow queues and packets in congested flow queues. Packets in a normal flow queue needs to be given priority to avoid HoL blocking, while at the same time, starvation of the congested flow queues needs to be avoided. Ideally the arbitration scheme should be able to take into account the situation at the root of the congestion tree to find the right arbitration balance, avoiding both HoL blocking and starvation. We have some initial ideas for such an arbitration scheme to improve the general scheme used by, among others, FBICM, CCFIT and EcoCC. To refine, implement and validate our ideas, however, is left as future work.

Bibliography

[1] José Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks An Engineering Approach*. Morgan Kaufmann, revised edition, 2003.

[2] William J. Dally and Brian Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[3] Gregory F. Pfister and V. Alan Norton. "Hot Spot" Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, 34(10):943–948, 1985.

[4] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), 1965.

[5] InfiniBand Trade Association. *Infiniband Architecture Specification*, 1.2.1 edition, November 2007.

[6] Top 500 supercomputer sites. http://top500.org/, November 2012.

[7] Ron Brightwell, Kevin T. Pedretti, Keith D. Underwood, and Trammell Hudson. SeaStar Interconnect: Balanced Bandwidth for Scalable Performance. *IEEE Micro*, 26(3):41–57, 2006.

[8] Mark Crovella and Balachander Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. Wiley, John Wiley & Sons, Inc., 2006.

[9] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Computer Publishing, John Wiley & Sons, Inc., 1991.

[10] Jean-Yves Le Boudec and Patrick Thiran. Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, January 2001.

[11] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. John Wiley, New York, 3 edition, 1997.

[12] Karl Johan Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[13] Ernst Gunnar Gran, Magne Eimot, Sven-Arne Reinemo, Tor Skeie Olav Lysne, Lars Paul Huse, and Gilad Shainer. First Experiences with Congestion Control in InfiniBand Hardware. In *Proceedings of 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS 2010)*, pages 1–12, 2010.

[14] Mellanox Technologies. *Mellanox OFED for Linux User Manual*, 1.5.3 edition, February 2012.

[15] John S. Carson. Modeling and simulation worldviews. In *Proceedings of the 1993 Winter Simulation Conference*, WSC '93, pages 18–23. ACM, 1993.

[16] Thomas J. Schriber and Daniel T. Brunner. Inside discrete-event simulation software: how it works and why it matters. In *Proceedings of the 2002 Winter Simulation Conference*, volume 1, pages 97–107 vol.1, 2002.

[17] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, SIMUTools, pages 60:1–60:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[18] Ernst Gunnar Gran and Sven-Arne Reinemo. InfiniBand Congestion Control: Modelling and validation. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 390–397. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[19] Ernst Gunnar Gran, Eitan Zahavi, Sven-Arne Reinemo, Tor Skeie, Gilad Shainer, and Olav Lysne. On the Relation between Congestion Control, Switch Arbitration and Fairness. In *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 342–351, 2011.

[20] Ernst Gunnar Gran, Sven-Arne Reinemo, Olav Lysne, Tor Skeie, Eitan Zahavi, and Gilad Shainer. Exploring the Scope of the InfiniBand Congestion Control Mechanism. In *Proceedings of 2012 IEEE 26th International Parallel Distributed Processing Symposium (IPDPS 2012)*, pages 1131–1143, 2012.

[21] Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García, José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato. Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks. In *2011 International Conference on Parallel Processing (ICPP)*, pages 662–672, 2011.

[22] Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García, José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato. Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks. ***Submitted to*** *IEEE Transactions on Parallel and Distributed Systems*, 2013.

[23] William J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[24] The Titan Cray XK7. Location: Oak Ridge National Laboratory, United States. Local site: http://www.olcf.ornl.gov/titan/
Top500 system site: http://www.top500.org/system/177975, November 2012.

[25] The IBM Sequoia - BlueGene/Q. Location: Lawrence Livermore National Laboratory, United States. Local site: https://asc.llnl.gov/computing_resources/sequoia/
Top500 system site: http://www.top500.org/system/177556, November 2012.

[26] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34:892–901, 1985.

[27] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. Technical report, Dipartimento di Informatica, Universita di of Pisa, 1995.

[28] The IBM SuperMUC - iDataPlex DX360M4. Location: Leibniz Supercomputer Centre, Germany. Local site: http://www.lrz.de/services/compute/supermuc/
Top500 system site: http://top500.org/system/177719, November 2012.

[29] The Dell Stampede - PowerEdge C8220. Location: Texas Advanced Computing Center, United States. Local site: http://www.tacc.utexas.edu/stampede
Top500 system site: http://top500.org/system/177931, November 2012.

[30] The Tianhe-1A - NUDT YH MPP. Location: National Supercomputing Center in Tianjin, China. Top500 system site: http://top500.org/system/176929, November 2012.

[31] Min Xie, Yutong Lu, Kefei Wang, Lu Liu, Hongjia Cao, and Xuejun Yang. Tianhe-1A Interconnect and Message-Passing Services. *IEEE Micro*, 32(1):8–20, 2012.

[32] Edsger W. Dijkstra. Two starvation-free solutions of a general exclusion problem. EWD 625, Plataanstraat 5, 5671 AL Nuenen, The Netherlands.

[33] Kanianthra M. Chandy and Jayadev Misra. The Drinking Philosophers Problem. *ACM Transactions on Programming Languages and Systems*, 6(4):632–646, October 1984.

[34] Eitan Zahavi, Gregory Johnson, Darren J. Kerbyson, and Michael Lang. Optimized InfiniBand^{TM}Fat-tree Routing for Shift All-To-All Communication Patterns. *Concurrency and Computation: Practice and Experience*, 22(2):217–231, February 2010.

[35] Herbert Sullivan and Theodore R. Bashkow. A large scale, homogeneous, fully distributed parallel machine, I. *SIGARCH Computer Architecture News*, 5(7):105–117, 1977.

[36] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS 2002)*, pages 8 pp–, 2002.

[37] Larry L. Peterson and Bruce S. Davie. *Computer Networks, A Systems Approach*. Morgan Kaufmann, 1996.

[38] Parviz Kermani and Leonard Kleinrock. Virtual Cut-through: A New Computer Communication Switching Technique. *Computer Networks*, 3(4):267–286, September 1979.

[39] Charles Seitz et al. Wormhole Chip Project Report, Winter 1985.

[40] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transaction on Computers*, C-36(5):547–543, May 1987.

[41] Chris Eddington. InfiniBridge: An InfiniBand Channel Adapter with Integrated Switch. *IEEE Micro*, 22(2):48–56, 2002.

[42] InfiniBand Trade Association. *Infiniband Architecture Specification*, 1.2.1 edition, November 2007. (Chapter 7.9, Flow Control:212-216).

[43] Pedro J. García, José Flich, José Duato, Ian Johnson, Francisco J. Quiles, and Finbar Naven. Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture. In *High Performance Embedded Architectures and Compilers*, pages 266–285, 2005.

[44] William J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[45] Mu-Cheng Wang, Howard J. Siegel, Mark A. Nichols, and Seth Abraham. Using a Multipath Network for Reducing the Effects of Hot Spots. *IEEE Transactions on Parallel and Distributed Systems*, 6(3):252–268, 1995.

[46] Pen-Chung Yew, Nian-Feng Tzeng, and Duncan H. Lawrie. Distributing Hot-Spot Addressing in Large-Scale Multiprocessors. *IEEE Transactions on Computers*, C-36(4):388–395, 1987.

[47] Li-Shiuan Peh and William J. Dally. Flit-Reservation Flow Control. In *Proceedings of Sixth International Symposium on High-Performance Computer Architecture HPCA-6, 2000*, pages 73–84, 2000.

[48] Nikolaos I. Chrysos. Congestion Management for Non-Blocking Clos Networks. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, ANCS '07, pages 117–126. ACM, 2007.

[49] William J. Dally and Hiromichi Aoki. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[50] José Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[51] Mithuna Thottethodi, Alvin R. Lebeck, and Shubhendu S. Mukherjee. BLAM: A High-Performance Routing Algorithm for Virtual Cut-Through Networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003)*, pages 10 pp.–, 2003.

[52] Paul Gratz, Boris Grot, and Stephen W. Keckler. Regional Congestion Awareness for Load Balance in Networks-on-Chip. In *IEEE 14th International Symposium on High Performance Computer Architecture, HPCA 2008*, pages 203–214, 2008.

[53] Daniel Franco, Indhira Garcés, and Emilio Luque. A New Method to Make Communication Latency Uniform: Distributed Routing Balancing. In *Proceedings of the 13th international conference on Supercomputing*, ICS '99, pages 210–219, New York, NY, USA, 1999. ACM.

[54] Arjun Singh, William J. Dally, Brian Towles, and Amit K. Gupta. Globally Adaptive Load-Balanced Routing on Tori. *Computer Architecture Letters*, 3(1):2–2, 2004.

[55] Yuval Tamir and Gregory L. Frazier. High-performance multiqueue buffers for VLSI communication switches. In *Conference Proceedings of 15th Annual International Symposium on Computer Architecture, 1988*, pages 343–354, 1988.

[56] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High-Speed Switch Scheduling for Local-Area Networks. *ACM Transactions on Computer Systems*, 11(4):319–352, November 1993.

[57] Yuval Tamir and Gregory L. Frazier. Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches. *IEEE Transactions on Computers*, 41(6):725–737, 1992.

[58] Teresa Nachiondo, José Flich, and José Duato. Buffer Management Strategies to Reduce HoL Blocking. *IEEE Transactions on Parallel and Distributed Systems*, 21(6):739–753, June 2010.

[59] Wladek Olesinski, Hans Eberle, and Nils Gura. Scalable Alternatives to Virtual Output Queuing. In *Proceedings of IEEE International Conference on Communications, 2009. ICC'09*, pages 1–6, 2009.

[60] Jesús Escudero-Sahuquillo, Pedro J. García, Francisco J. Quiles, and José Duato. An Efficient Strategy for Reducing Head-of-Line Blocking in Fat-Trees. In *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II*, Euro-Par'10, pages 413–427, Berlin, Heidelberg, 2010. Springer-Verlag.

[61] Mellanox Technologies Ltd. *InfiniScale IV*, [Online, May 2013]. http://www.mellanox.com/related-docs/prod_silicon/PB_InfiniScale_IV.pdf.

[62] Wei Lin Guay, Bartosz Bogdanski, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS 2011)*, pages 197–208. IEEE Computer Society, 2011.

[63] Wei Lin Guay, Sven-Arne Reinemo, Olav Lysne, and Tor Skeie. dFtree: A Fat-Tree Routing Algorithm Using Dynamic Allocation of Virtual Lanes to Alleviate Congestion in InfiniBand Networks. In *Proceedings of the First International Workshop on Network-Aware Data Management*, NDM '11, pages 1–10. ACM, 2011.

[64] José Duato, Ian Johnson, José Flich, Finbar Naven, Pedro J. García, and Teresa Nachiondo. A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture, HPCA-11*, pages 108–119, 2005.

[65] Pedro J. García, Francisco J. Quiles, José Flich, José Duato, and Ian Johnson. RECN-DD: A Memory-Efficient Congestion Management Technique for Advanced Switching. In *Proceedings of International Conference on Parallel Processing, 2006. ICPP 2006*, pages 23–32, 2006.

[66] Gaspar Mora, Pedro J. García, José Flich, and José Duato. RECN-IQ: A Cost-Effective Input-Queued Switch Architecture with Congestion Management. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP, pages 74–. IEEE Computer Society, 2007.

[67] Jesús Escudero-Sahuquillo, Pedro J. García, Francisco J. Quiles, José Flich, and José Duato. FBICM: Efficient Congestion Management for High-Performance Networks Using Distributed Deterministic Routing. In *Proceedings of the 15th International Conference on High Performance Computing*, HiPC, pages 503–517. Springer-Verlag, 2008.

[68] Jesús Escudero-Sahuquillo, Pedro J. García, Francisco J. Quiles, José Flich, and José Duato. Cost-Effective Congestion Management for Interconnection Networks Using Distributed Deterministic Routing. In *Proceedings of the 2010 IEEE 16th International Conference on Parallel and Distributed Systems, ICPADS*, pages 355–364, 2010.

[69] Mithuna Thottethodi, Alvin R. Lebeck, and Shubhendu S. Mukherjee. Self-Tuned Congestion Control for Multiprocessor Networks. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture, HPCA-7*, pages 107–118, 2001.

[70] Elvira Baydal and Pedro López. A Robust Mechanism for Congestion Control: INC. In Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors, *Euro-Par 2003 Parallel Processing*, volume 2790 of *Lecture Notes in Computer Science*, pages 958–968. Springer Berlin Heidelberg, 2003.

[71] Jae H. Kim, Ziqiang Liu, and Andrew A. Chien. Compressionless Routing: A Framework for Adaptive and Fault-tolerant Routing. In *Computer Architecture, 1994., Proceedings the 21st Annual International Symposium on*, pages 289–300, 1994.

[72] IEEE 802 LAN/MAN Standards Committee. *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification.*, Available at http://www.ieee802.org/1 IEEE 802.1Qau-2010 edition.

[73] Joan-LLuís Ferrer, Elvira Baydal, Antonio Robles, Pedro López, and José Duato. Congestion Management in MINs Through Marked & Validated Packets. In *15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07)*, pages 260 – 7, 2007.

[74] Joan-LLuís Ferrer, Elvira Baydal, Antonio Robles, Pedro López, and José Duato. On the Influence of the Packet Marking and Injection Control Schemes in Congestion Management for MINs. In *Euro-Par 2008 Parallel Processing. 14th International Euro-Par Conference*, pages 930 – 9, 2008.

[75] Jose Renato Santos, Yoshio Turner, and G. (John) Janakiraman. Evaluation of Congestion Detection Mechanisms for InfiniBand Switches. In *Proceedings of the Global Telecommunications Conference, 2002. GLOBECOM '02*, pages 2276–2280, 2002.

[76] Jose Renato Santos, Yoshio Turner, and G. (John) Janakiraman. End-to-End Congestion Control for InfiniBand. In *Proceedings of Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. INFOCOM. IEEE Societies*, volume 2, pages 1123–1133, 2003.

[77] Gregory F. Pfister, Mitch Gusat, Wolfgang Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, Ronald P. Luijten, R. Krishnamurthy, and José Duato. Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control. Invited paper in *High Performance Interconnects for Distributed Computing*, july 2005.

[78] InfiniBand Trade Association. *Infiniband Architecture Specification*, 1.2.1 edition, November 2007. (Annex A10, Congestion Control:1650-1697).

[79] Mike Galles. Spider: A High-Speed Network Interconnect. *IEEE Micro*, 17(1):34–39, January 1997.

[80] Mellanox Technologies Ltd. *ConnectX*, [Online, May 2013]. http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX_Silicon.pdf.

[81] HPC Challenge Benchmark. http://icl.cs.utk.edu/hpcc/, [Online, May 2013].

[82] Piotr Luszczek, Jack J. Dongarra, David Koester, Rolf Rabenseifner, Bob Lucas, Jeremy Kepner, John McCalpin, David Bailey, and Daisuke Takahashi. Introduction to the HPC Challenge Benchmark Suite. Technical Report LBNL-57493, Lawrence Berkeley National Laboratory, April 2005.

[83] Sven-Arne Reinemo, Tor Skeie, and Manoj K. Wadekar. Ethernet for High-Performance Data centers: On the New IEEE Datacenter Bridging Standards. *IEEE Micro*, 30:42–51, 2010.

List of Appendices

Paper I	**InfiniBand Congestion Control, Modelling and Validation**
published at	The 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools2011, OMNeT++ 2011 Workshop)
authors	Ernst Gunnar Gran and Sven-Arne Reinemo
Paper II	**On the Relation Between Congestion Control, Switch Arbitration and Fairness**
published at	The 11th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2011)
authors	Ernst Gunnar Gran, Eitan Zahavi, Sven-Arne Reinemo, Tor Skeie, Gilad Shainer and Olav Lysne
Paper III	**First Experiences with Congestion Control in InfiniBand Hardware**
published at	The 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2010)
authors	Ernst Gunnar Gran, Magne Eimot, Sven-Arne Reinemo, Tor Skeie, Olav Lysne, Lars Paul Huse, and Gilad Shainer
Paper IV	**Exploring the Scope of the InfiniBand Congestion Control Mechanism**
published at	The 26th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2012)
authors	Ernst Gunnar Gran, Sven-Arne Reinemo, Olav Lysne, Tor Skeie, Eitan Zahavi, and Gilad Shainer
Paper V	**Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks**
published at	The 40th Annual Conference – International Conference on Parallel Processing (ICPP 2011)
authors	Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García, José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato
Paper VI	**Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks**
submitted to	IEEE Transactions on Parallel and Distributed Systems
authors	Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García, José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato

Paper I

InfiniBand Congestion Control
Modelling and Validation

Ernst Gunnar Gran and Sven-Arne Reinemo

InfiniBand Congestion Control

Modelling and validation

Ernst Gunnar Gran
Simula Research Laboratory
Martin Linges vei 17
1325 Lysaker, Norway
ernstgr@simula.no

Sven-Arne Reinemo
Simula Research Laboratory
Martin Linges vei 17
1325 Lysaker, Norway
svenar@simula.no

ABSTRACT

In a lossless interconnection network congestion may results in performance degradation if no countermeasure is taken. To relieve the consequences of congestion, and by that to achieve good utilization of networks resources even at high network load, congestion control (CC) has been added to the InfiniBand specification. The behavior of the InfiniBand CC is, however, governed by a set of CC parameters. Exactly how to set these parameters to ensure an all over efficient network is still not well understood. It is time consuming, costly and hard to explore the CC parameter space in a large scale cluster. Therefore, a simulation platform is needed. In this paper we present our CC capable IB model implemented in the OMNeT++ environment. We explain the basics of our model, and validate it against CC capable hardware to show its high accuracy.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations—*Network Management*; C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks—*High-speed*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; I.6.5 [**Simulation and Modeling**]: Model Development; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event*

General Terms

Simulation, validation, performance, design

Keywords

InfiniBand, congestion control, OMNeT++

1. INTRODUCTION

Congestion control (CC) is a hot topic in interconnection networks for large high performance computing (HPC) clusters because the interconnection networks used in HPC are lossless, i.e. they use flow control to avoid packet loss caused by buffer overflows. In lossless networks congestion may spread and result in severe performance degradation if no countermeasures are taken[3, 5, 6, 10].

Congestion control was added to version 1.2.1 of the InfiniBand (IB) standard [7], and hardware with support for CC has just recently appeared, even though the CC features are not yet generally available. A major challenge with the congestion control mechanism found in IB is how to properly use it, i.e. how to configure the congestion control parameters properly for a given system. This problem is not yet understood, and it is time consuming, costly, and difficult to explore the parameter space in a large scale cluster. Another related question is how CC affects performance in scenarios with and without congestion, which is equally hard to understand. Therefore, a simulation model would be a valuable tool in the search for a better understanding on how to properly configure IB CC in various topologies of different sizes, and useful to study the effect of congestion control in different application scenarios.

Our contribution in this paper consists of an implementation of the IB CC mechanism in OMNeT++ [12] and a validation of this model against the hardware implementation of CC in the Mellanox ConnectX Host Channel Adapters [8] and Mellanox InfiniScale IV switches [9].

The paper is structured as follows: In section 2 we give an overview of the IB CC mechanism, followed by a detailed description of the simulation model in section 3. Then we describe the hardware test-bed we have used to validate the simulation model in section 4. The validation scenarios and results are described and analysed in section 5 and 6, respectively. Finally, we conclude in section 7.

2. THE CC CONCEPT IN INFINIBAND

In this section we give an overview of the IB CC mechanism as specified in the InfiniBand Architecture Specification release 1.2.1 [7]. The IB CC mechanism is based on a closed loop feedback control system where a switch detecting congestion marks packets contributing to the congestion by setting a specific bit in the packet headers, the *Forward Explicit Congestion Notification* (FECN) bit (fig. 1 (1)). The congestion notification is carried through to the destination by this bit. The destination registers the FECN bit, and returns a packet with the *Backward Explicit Congestion Notification* (BECN) bit set to the source (fig. 1 (2)). The source then temporarily reduces the injection rate to resolve congestion (fig. 1 (3)).

OMNeT++ 2011, March 21, Barcelona, Spain

DOI 10.4108/icst.simutools.2011.245509

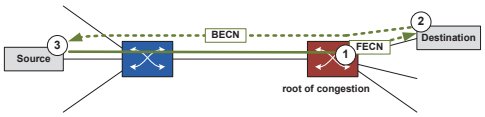


Figure 1: Congestion control in InfiniBand.

The exact behaviour of the IB CC mechanism depends upon the values of a set of CC parameters governed by a *Congestion Control Manager*. These parameters determine characteristics like when switches detect congestion, at what rate the switches will notify destination nodes using the FECN bit, and how much and for how long a source node contributing to congestion will reduce its injection rate. Appropriately set, these parameters should enable the network to resolve congestion by avoiding head-of-line blocking [10], while still utilizing the network resources efficiently.

2.1 Congestion Control at a Switch

The switches are responsible for detecting congestion and notifying the destination nodes using the FECN bit. A switch detects congestion on a given *port* and a given *Virtual Lane* (Port VL) depending on a *threshold* parameter. If the threshold is crossed, a port may enter the Port VL congestion state, which again may lead to FECN marking of packets.

The *threshold*, represented by a weight ranging from 0 to 15 in value, is the same for all VLs on a given port, but could be set to a different level for each port. A weight of 0 indicates that no packets should be marked, while the values 1 through 15 represent a uniformly decreasing value of the threshold. That is, a value of 1 indicates a high threshold with high possibility of congestion spreading, caused by Port VLs moving into the congestion state too late. A value of 15 on the other hand indicates a low threshold with a corresponding low possibility of congestion spreading, but at the cost of a higher probability for a Port VL to move into the congestion state even when the switch is not really congested. The exact implementation of the threshold depends on the switch architecture and is left to the designer of the switch.

A Port VL enters the congestion state if the threshold is crossed and it is the root of congestion, i.e. the Port VL has available *credits*[1] to output data. If the Port VL has no available credits, it is considered to be a victim of congestion and shall not enter the congestion state unless a specific $Victim\_Mask$ is set for the port. The $Victim\_Mask$ is typically set for ports connecting a *channel adapter* (CA) to the switch. A CA that is not able to process received packets fast enough will not consider itself to be a root of congestion even if a congestion tree[6] then builds up with the CA as the root. In this special case the Port VL at the switch connecting the CA should consider itself to be the root of congestion, even if it is actually a victim, and move into the congestion state.

When a Port VL is in the congestion state its packets are eligible for FECN marking. A packet will then get the FECN bit set depending on two CC parameters at the switch, the $Packet\_Size$ and the $Marking\_Rate$. Packets with a size smaller than the $Packet\_Size$ will not get the FECN bit set. The $Marking\_Rate$ sets the mean number of eligible packets sent between packets actually being marked. With both the $Packet\_Size$ and the $Marking\_Rate$ set to 0, all packets should get the FECN bit set while a Port VL is in the congestion state.

2.2 Congestion Control at a Channel Adapter

When a destination CA receives a packet with a FECN bit, the CA should as quickly as possible notify the source of the packet about the congestion[2]. As earlier mentioned, this is done by returning a packet with the BECN bit set back to the source. The packet with the BECN bit could either be an acknowledgement packet (ACK) for a reliable connection or an explicit *congestion notification packet* (CNP). In either case it is important that the ACK or the CNP is sent to the source as soon as possible to ensure a fast response to the congestion.

When a source CA receives a packet with the BECN bit set, the CA lowers the injection rate of the corresponding traffic flow. That is, the injection rate of either the related *queue pair* (QP) or the corresponding *service layer* (SL) will be reduced. Congestion control at a CA port operates either at the QP or at the SL level, exclusively. To determine how much and for how long the injection rate should be reduced, the CA uses a *Congestion Control Table* (CCT) and a set of CC parameters. The CCT, consisting of at least 128 entries, holds *injection rate delay* (IRD) values that define the delay between consecutive packets sent by a particular flow (QP or SL). Each flow with CC activated holds an index into the CCT, the $CCTI$. When a new BECN arrives, the $CCTI$ of the flow is increased by $CCTI\_Increase$. The CCT is usually populated in such a way that a larger index yields a larger IRD. Then consecutive BECNs increase the IRD which again decreases the injection rate. The upper bound of the $CCTI$ is given by $CCTI\_Limit$.

To increase the injection rate again, the CA relies on a $CCTI\_Timer$, maintained separately for each SL of a port. Each time the timer expires the $CCTI$ is decremented by one for all flows associated with the corresponding port SL. When the $CCTI$ of a flow reaches zero, the flow now longer experience any IRD. Each port SL also has a $CCTI\_Min$ parameter. Using the $CCTI\_Min$ it is possible to impose a minimum IRD to the port SL, as the $CCTI$ should never be reduced below the $CCTI\_Min$.

3. THE SIMULATION MODEL

Our IB CC implementation is based on the IB model made available to the OMNeT++ community by Mellanox Technologies Ltd in 2007/2008. We have ported this model to the OMNeT++ 4 environment, made several bug fixes, and added some general extensions to the IB model to make it more suitable for our CC studies (e.g. support for interval logging of network statistics like bandwidth and buffer occupancy, and detailed control of the startup time for the traffic

[1]The number of *credits* at a port determines how much traffic the port is allowed to send to the next port downstream. The credit value corresponds to the available buffer space at the downstream port.

[2]There are three exceptions. The FECN bit in a multicast packet, acknowledgement packet or congestion notification packet should be ignored. That is, no congestion notification is sent back to the source in these three cases.

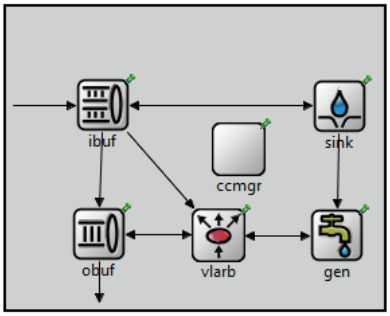


Figure 2: The HCA compound module of the IB model.

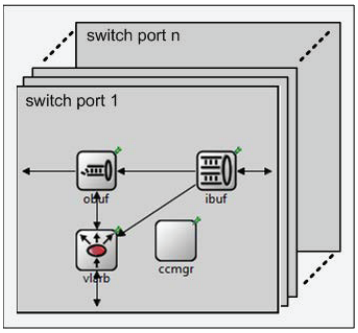


Figure 3: The switch compound module of the IB model.

generators). Below we start by giving a brief overview of the IB model, before we give a more detailed explanation of the IB CC extensions to the model in the next section.

3.1 The IB Model

The IB model consists of a set of simple and compound modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, and routing using linear forwarding tables.

The two building blocks for creating networks using the IB model are the *Host Channel Adapter* (HCA) compound module and the *Switch* compound module, shown in figure 2 and 3 correspondingly. The *Switch* consists of a set of *SwitchPort*s, which are by themselves compound modules. During a simulation, an HCA represents both a traffic injector and a traffic sink in the network, while a *Switch* acts as a forwarding node. The HCAs and *switches* are connected using *gates*, corresponding to links in the network.

The HCAs and the *SwitchPort*s consist of a set of common simple modules *ibuf*, *obuf*, and *vlarb* (the *ccmgr* is part of the IB CC extension), while the simple modules *gen* and *sink* are exclusive to the HCAs. The *ibuf* represents an input buffer with support for virtual lanes, virtual output queuing (VoQ) and virtual cut through switching. The *obuf* represents a simple output buffer, while the *vlarb* implements round robin arbitration over the different VLs and multiple input ports (if the module is part of a *SwitchPort*). The *gen* implements traffic generation in a HCA, while the *sink* is the part of the HCA responsible for removing traffic from the network. The *gen* module supports several traffic generation schemes, e.g. varying the injection rate, the packet size and the destination node distribution.

In general, the *gen* module at an HCA generates traffic that is forwarded through the *vlarb* to the *obuf* of the HCA. From there it is sent out into the network. At a *Switch*(*Port*) the *ibuf* receives the traffic, does the routing decision and moves the traffic into the corresponding VoQ. Here the traffic waits until the *vlarb* of the given output port grants access to the corresponding *obuf*. At an HCA the *ibuf* receives traffic and forwards it to the *sink*. The IB flow control is managed by the *ibuf*s and the *obuf*s, exchanging flow control messages.

3.2 The IB CC Extensions

The InfiniBand Congestion Control mechanism is implemented by the simple module *ccmgr*. The *ccmgr* is included in the compound modules for the HCA and the *SwitchPort*, and manages everything related to congestion control in these modules, with the help of the other simple modules therein. In particular, all CC parameters specified in the InfiniBand Architecture Specification release 1.2.1 [7] are supported by the *ccmgr* module.

3.2.1 The Switch Model

At a *Switch*, the *ccmgr* is responsible for detection and notification of congestion. The *ibuf* at a given *SwitchPort* reports every change in the fill ratio of one of its VoQs to the *ccmgr* of the *SwitchPort* corresponding to the VoQ in question. This reporting is done at the VL level. Notice that the mentioned *ibuf* and *ccmgr* in most situations will be located in different *SwitchPort*s, as the traffic usually leaves and enters a switch on different ports. Now, using the updates from the different *ibuf*s, the *ccmgr* keeps track of the fill ratios of all VoQs headed for the corresponding *obuf*, and decides whether the corresponding Port VL should be considered to be in the congestion state or not. The decision is based upon the value of the *threshold*, a consideration of the $Victim\_Mask$, as well as how the fill ratio is evaluated against the *threshold*. This evaluation of the *threshold* against the fill ratios of the VoQs, is an example of a design decision that is left to the switch designer. Our IB CC extension supports three different ways of doing this mapping: comparing the fill ratio of each individual VoQ to the *threshold*, comparing the sum of the fill ratios of the VoQs to the *threshold*, and last comparing the sum of the fill ratios of the VoQs to a *threshold* divided by the number of contributing input ports. At the *obuf*, each time the module wants to send a new packet into the network, the *ccmgr* is asked to update the FECN bit of the packet before it is sent. When doing this, the *ccmgr* considers if the corresponding Port VL is in the congestion state, as well as if the two CC parameters $Marking\_Rate$ and $Packet\_Size$ qualify for setting the FECN bit.

3.2.2 The HCA Model

At an HCA, the *ccmgr* inspects every packet entering the

ibuf to check if the FECN or BECN bit is set (if both bits are set, the FECN bit is ignored). Upon detection of a FECN bit, the *ccmgr* creates a CNP to send a BECN back to the source of the FECN marked packet; a contributor to congestion. The CNP is placed in a special CNP queue at the *ccmgr*. The *obuf* of the HCA gives priority to the CNP queue over other traffic to make sure that the CNPs are sent as soon as possible.

The CCT is contained in the *ccmgr* of an HCA. Upon reception of a CNP with the BECN bit set, the *ccmgr* updates the *CCTI* of the corresponding SL or QP - depending on the level of congestion control operation - by the value of *CCTI_Increase*. At the same time, the related timer is updated. Each timer corresponding to a SL or a QP is in our IB CC extension implemented as a *CCTI_Timer* delayed message sent from the *ccmgr* to itself. If the message is not canceled, the return of the message will cause the *CCTI* identified by the message to be decreased (and a new timer message is sent, if needed).

When the *vlarb* at the HCA arbitrates between flows from the *gen* requesting access to the *obuf*, the *ccmgr* calculates the needed IRD values from the CCT to assist in the arbitration process. The *vlarb* uses the feedback from the *ccmgr* during arbitration to make sure that only traffic flows contributing to congestion is held back. The calculation of the IRD values follows the guidelines given in the IB specification[7].

3.3 Scalability

The level of detail required from the simulation model to be able to accurately simulate the IB CC mechanism and its parameter space, could be a challenge for the scalability and the usefulness of the simulator. Our experience shows that the OMNeT++ environment and the IB model scales quite well, and there are no specific limitations regarding topology or network size in the model itself. In addition to simulations of network sizes similar to the examples given in this paper, we have run extensive CC simulations of the Sun Datacenter InfiniBand Switch 648[11]. This is an IB 1.2 compliant QDR capable switch with 648 ports. The internal topology of the switch is a three-stage full non-blocking Clos network. Even for such a complex network structure, with all CC features enabled and 648 active end nodes, the memory requirement for a simulation is less than 1.5GB. Simulating 0.5 seconds of real time network traffic, corresponding to approximately 100GB of data transferred in the network, takes 1-2 days to complete depending on the CC parameters and if the CC operates at the SL or QP level. This is when running on an Intel Q6600 CPU, using only *one* out of four cores. We plan to parallelize the model in the future in order to reduce the simulation time.

4. THE HARDWARE TEST BED

The hardware test bed used to measure the behavior of the IB CC is shown in figure 4. Seven Sun Fire X2200 M2 hosts (H1-H7) are connected to two InfiniScale IV Mellanox switches (IS4). The host-to-switch links have a capacity of 20 Gbit/s each, while the switch-to-switch link has a capacity of 40 Gbit/s.

The Mellanox ConnectX HCAs and IS4 switches are the latest generation of IB hardware from Mellanox Technologies, and both the HCAs and switches include hardware support for the IB CC mechanism[3].

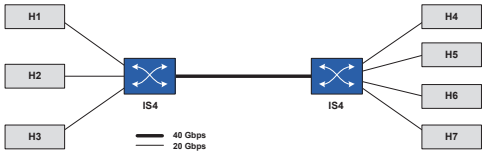


Figure 4: The test bed.

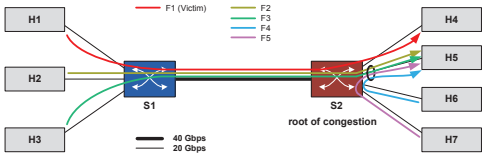


Figure 5: Flow configuration in scenario 1.

The compute nodes in our test bed consists of seven Sun Fire X2200 M2 servers that are connected as hosts H1-H7 in figure 4. Each host has a dual port DDR HCA fitted in a 8x PCIe 1.1 slot, one dual core AMD Opteron 2210 CPU, and 2GB of RAM. All hosts run Ubuntu Linux 8.04 x86_64 with kernel version 2.6.24-24-generic and OFED 1.4.1. The PCIe 1.1 8x slots in these machines have a signalling rate of 20 Gbit/s, which equals a theoretical bandwidth of 16 Gbit/s when counting for the 8b/10b encoding overhead. The achievable bandwidth is further reduced by PCIe protocol overhead, the speed of other system components etc.

To generate traffic on the hosts we used several different tools. *Netpipe* [2], which measures bandwidth and latency for different packet sizes, is used to get some basic performance numbers. To be able to study congestion in a controlled manner we have implemented some changes to the *perftest* application suite (part of OFED) to support regular bandwidth reporting and continuously sending traffic at full capacity. The modified *perftest* is used to both create congestion in the network and to measure the impact of congestion.

5. THE VALIDATION SCENARIOS

In order to validate the performance of the simulation model we have defined two scenarios as described below.

5.1 Scenario 1

The purpose of communication scenario 1 is twofold. First, it illustrates the negative effect that congestion can have on a flow not contributing to congestion, a *victim* flow (flow 1 from *H1* to *H4* in fig. 5). Second, it illustrates how this negative effect can be avoided by using congestion control.

In this scenario we use the following communication pattern (fig. 5): Flow 1 (F1) from *H1* to *H4*, and flow 2 - 5 (F2-F5) where *H2*, *H3*, *H6*, and *H7* all send to *H5*. Communication starts with only F1 active, then F2 - F5 are ac-

[3]To enable congestion control, custom firmware is required for both switches and HCAs. This is not yet generally available.

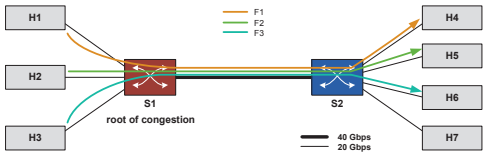


Figure 6: Flow configuration in scenario 2.

tivated one by one with one second intervals. When a flow is active it tries to send at maximum speed, using a reliable connection.

5.2 Scenario 2

The purpose of communication scenario 2 is to study how congestion control performs when there is no victim present, and by that no HOL blocking to reduce in order to potentially improve overall performance.

In this scenario we use the following communication pattern (fig. 6): Flow 1 (F1) from *H1* to *H4*, flow 2 (F2) from *H2* to *H5*, and flow 3 (F3) from *H3* to *H6*. Communication starts with only F1 active, then F2 and F3 are activated one by one with one second intervals. As before, when a flow is active, it tries to send at maximum speed, using a reliable connection. In this scenario there is no victim flow, but there is contention for bandwidth on the link between *S1* and *S2* that is shared by all three flows.

6. THE VALIDATION RESULTS

The figures 7, 8 and 9 show the results from our validation studies of scenario 1 and 2, comparing hardware experiments with simulation results.

6.1 Scenario 1

We start by studying the hardware measurements from scenario 1 with congestion control turned *off*. In figure 7(a) we see the presence of both head of line (HOL) blocking and the parking lot problem[4]: As soon as the third flow, $F3$, is added after about 2.5 seconds, we see a drop in performance for all three flows down to half the bandwidth. The link from the switch $S2$ to the end node $H5$ has become a bottleneck, creating a congestion tree growing towards the contributors to congestion, $H2$ and $H3$. The congestion tree causes HOL blocking, hindering the flow $F1$ from progressing any faster between $S1$ and $S2$ than $F2$ and $F3$. As we add the flows $F4$ and $F5$, the HOL blocking continues. In addition, due to the fact that the switch $S2$ during (round robin) arbitration considers the flows $F2$ and $F3$ as a single flow, we see an unfairness in the amount of access the different traffic flows are given to the bottleneck link. The two locally connected flows, $F4$ and $F5$, are each given the same access to the bottleneck link as the two flows $F2$ and $F3$ combined; the parking lot problem is evident.

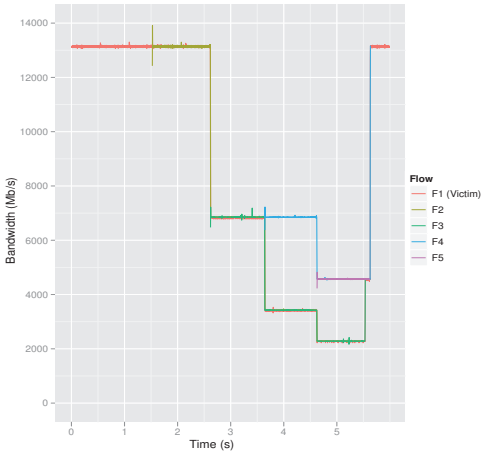
If we turn *on* congestion control in the hardware and repeat our scenario 1 experiment, we got the results presented in figure 7(b). Now, both the HOL blocking and the parking lot problem are removed. The flow $F1$ is sent through the network independently of the other flows, and all the contributors to congestion each got a fair share of the scarce resources at the root of the congestion tree. This was studied in-depth in [6].

Let us now turn our attention towards the simulator to see how it compares to hardware. Figure 7(c) and 7(d) show simulation results for scenario 1 with congestion control turned off and on, respectively. As we can see from figure 7(c), comparing it to figure 7(a), the simulator adheres to the hardware quite accurately when congestion control is turned off. The traffic flows are all experiencing the same throughput in the simulator as in the hardware, and both the HOL blocking and the parking lot problem is present. When congestion control is turned on, the resemblance between the hardware and the simulator is still quite good, as figure 7(b) and 7(d) show. The HOL blocking and the parking lot problem are both removed at the same cost in both the hardware and the simulator: increased oscillation among the contributors to congestion as they are constantly trying to adjust their injection rate to their fair share of the bottleneck link. Figure 9 shows how the fairness has improved for both the hardware measurements and the simulation results, when all four contributors are active. Each colored area represents the fraction of the total throughput given to the corresponding flow in each case. It is clear from the figure that the parking lot problem is removed, despite the oscillation observed for all the contributors.

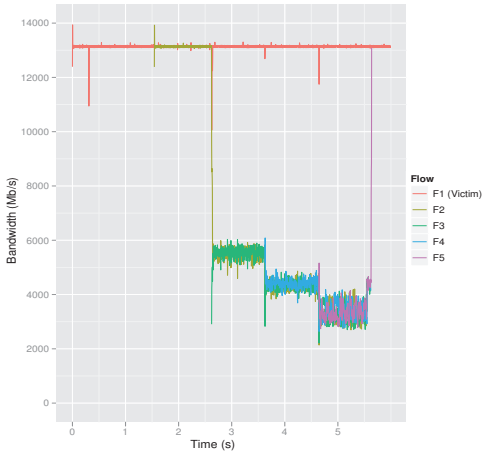
There is one noticeable difference in the two figures 7(b) and 7(d) though: In the one second time interval from 2s to 3s, the flow $F1$ experiences some oscillation during a simulation, an oscillation that is not present in the hardware experiment. This difference in behavior is related to the aggressiveness of the two contributors to congestion and the utilization of the bottleneck link. The hardware is not able to fully utilize the bottleneck link during the time where only two contributors to congestion are active. This is clearly visible in figure 7(b). The two contributors together achieve a throughput of approximately 11Gbps on average, while the capacity of the bottleneck link (or actually the end node connected to it) is just above 13Gbps. In the simulator, during the same time interval, the two contributors to congestion achieve approximately 13Gbps. This increased aggressiveness makes it possible to fully utilize the bottleneck link. Furthermore, it also results in a greater demand for resources at the left switch, $S1$. More specific, if the two contributors are too aggressive they will, together with flow $F1$, request more resource from the switch-to-switch link than the link can handle, and by that create a congestion tree with the root of the tree at $S1$, and not $S2$. When this happens, the flow $F1$ is actually a contributor to congestion itself, and should lower its injection rate just like any other contributor. This is exactly what is happening when we see a drop in performance for the flow $F1$ in the time interval between 2s and 3s during a simulation. The $F1$ drop in performance is not due to any HOL blocking, but a result of proper congestion control behavior from $H1$ when the switch $S1$ experiences congestion. This behavior is never seen in hardware as the contributors are less aggressive (resulting in an underutilization of the bottleneck link). When the flows $F4$ and $F5$ are added during a simulation, the congestion at the right switch, $S2$, is more severe, which again means that the injection rates of the flows $F2$ and $F3$ are never high enough to create congestion at the switch $S1$.

6.2 Scenario 2

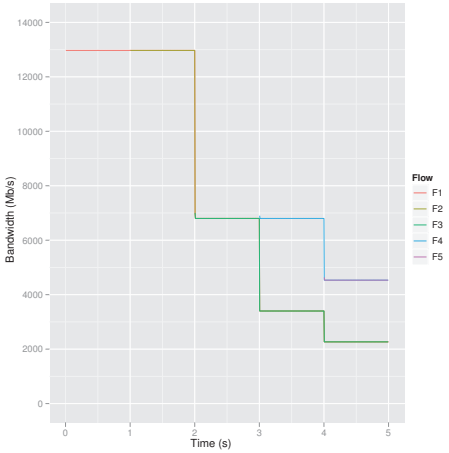
In figure 8(a) and 8(b) we see the results from the hard-



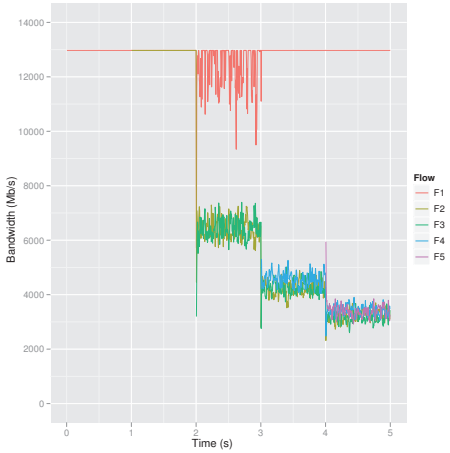
(a) Hardware. Congestion Control turned OFF.



(b) Hardware. Congestion Control turned ON.

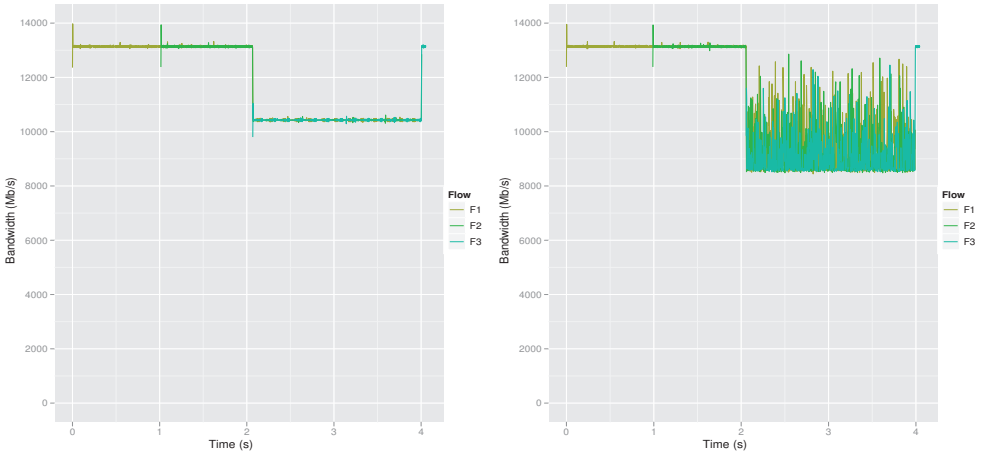


(c) Simulation. Congestion Control turned OFF.



(d) Simulation. Congestion Control turned ON.

Figure 7: Throughput for flows in scenario 1 from both the hardware measurements and simulations.



(a) Hardware. Congestion Control turned OFF.

(b) Hardware. Congestion Control turned ON.

(c) Simulation. Congestion Control turned OFF.

(d) Simulation. Congestion Control turned ON.

Figure 8: Measured throughput for flows in scenario 2.

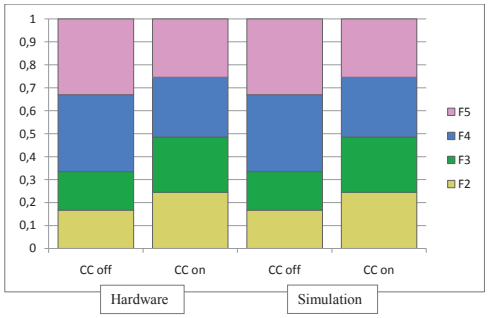


Figure 9: Fairness between the flows contributing to congestion.

ware experiments of scenario 2 (fig. 6), with congestion control turned *off* and *on*, respectively. The link between the two switches becomes the bottleneck in both experiments as soon as we add the third flow after two seconds. At this point the switch $S1$ becomes congested. When CC is turned off(fig. 8(a)), the three flows all experience the same drop in performance down to approximately 11.4 Gbps. The situation is both fair and stable. When CC is turned on, however, all three flows will constantly try to adjust their injection rates as a result of the congested situation at $S1$. The effect is seen as oscillation in figure 8(b) as soon as the third flow is added. In addition, the average throughput of the three flows drops to approximately 9Gbps.

Figure 8(c) and 8(d) show the simulation results for scenario 2. When the CC is turned *off* (fig. 8(c)), we see a drop in throughput as soon as the third flow is added, just as we experienced in hardware (fig. 8(a)). The simulator and the hardware show the same behavior and the same performance. If we turn the CC *on*(fig. 8(d)), the oscillation becomes evident again at the time 2s as the three contributors to congestion are constantly trying to adjust their injection rates. The characteristics of the flows in the simulator are similar to the ones in hardware, even though the oscillation is less dominant in the simulator.

Overall, the figures 7, 8 and 9 show a good resemblance between our simulator and the hardware. Both the throughput and the traffic characteristics are very close for both the scenario 1 and the scenario 2 cases, especially considered the complex dynamics of a closed loop feedback control system like the IB CC mechanism. We are simulating a "black box" where several design decisions are left to the hardware designer. The exact behavior and performance of the IB CC mechanism is therefore likely to vary among switches and HCAs from different vendors – depending on the chosen design. Our aim is to catch the general IB CC characteristics to be able to study them. The simulator does this very well. It is able to catch the effect of the HOL blocking as well as the performance improvement possible with IB CC.

7. CONCLUSIONS

Congestion control is an important topic in interconnection network research because congestion can severely degrade performance if no countermeasures are taken. In this paper we described a new simulation model for OMNeT++ that accurately simulates IB congestion control and that can be used to study the effect of, and how to configure, IB CC. Moreover, we presented measurements from a small real cluster that validates our simulation results. We expect the model to be a useful tool for further research, and we already have several works in progress using the simulation model.

8. ACKNOWLEDGMENTS

We would like to thank the HPC Advisory Council [1] for the support throughout the development and validation activities and for the contribution of the switches and firmware that were used for the validation. We would also like to thank Magne Eimot for his contributions to the development of the simulator, and last but not least, a thank you to Eitan Zahavi and Mellanox Technologies for the original implementation of the IB model and for sharing it with the OMNeT++ community.

9. REFERENCES

[1] High Performance Computing - Advisory Council. `http://www.hpcadvisorycouncil.com/`.

[2] NetPIPE - Network Protocol Independent Performance Evaluator, Sept. 2009. `http://www.scl.ameslab.gov/netpipe/`.

[3] W. J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3:194–205, Mar. 1992.

[4] W. J. Dally and B. Towles. *Principles and practices of interconnection networks*, chapter 15.4.1, pages 294–295. Morgan Kaufmann, 2004.

[5] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F.Naven. Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture. In *High Performance Embedded Architectures and Compilers*, pages 266–285, 2005.

[6] E. G. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. P. Huse, and G. Shainer. First Experiences with Congestion Control in InfiniBand Hardware. In *Proceeding of the 24th IEEE International Parallel & Distributed Processing Symposium*, pages 1–12, 2010.

[7] InfiniBand Trade Association. *Infiniband architecture specification*, 1.2.1 edition, November 2007. `http://www.infinibandta.org/`.

[8] Mellanox Technologies Ltd. *ConnectX*, 2009. `http://www.mellanox.com/related-docs/prod_silicon/PB_ConnectX_Silicon.pdf`.

[9] Mellanox Technologies Ltd. *InfiniScale IV*, 2009. `http://www.mellanox.com/related-docs/prod_silicon/PB_InfiniScale_IV.pdf`.

[10] G. F. Pfister and V. A. Norton. "Hot Spot" Contention and Combining in Multistage Interconnection Networks. *IEEE Trans. Computers*, 34:943–948, 1985.

[11] Sun Microsystems. Sun Datacenter Infiniband Switch 648 Architecture And Deployment, 2009. `http://www.oracle.com/us/sun/`.

[12] A. Varga. Using the OMNeT++ discrete event simulation system in education. *IEEE Transactions on Education*, 42(4):11 pp., Nov. 1999.

Paper II

On the Relation Between Congestion Control, Switch Arbitration and Fairness

Ernst Gunnar Gran, Eitan Zahavi, Sven-Arne Reinemo,
Tor Skeie, Gilad Shainer, and Olav Lysne

2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

On the Relation Between Congestion Control, Switch Arbitration and Fairness

Ernst Gunnar Gran*, Eitan Zahavi†, Sven-Arne Reinemo*, Tor Skeie*, Gilad Shainer†, Olav Lysne*
*Simula Research Laboratory, Fornebu, Norway. Email: {ernstgr, svenar, tskeie, olavly}@simula.no
†Mellanox Technologies, Israel/USA. Email: eitan@mellanox.co.il, Shainer@Mellanox.com

***Abstract*—In lossless interconnection networks such as InfiniBand, congestion control (CC) can be an effective mechanism to achieve high performance and good utilization of network resources. The InfiniBand standard describes CC functionality for detecting and resolving congestion, but the design decisions on how to implement this functionallity is left to the hardware designer. One must be cautious when making these design decisions not to introduce fairness problems, as our study shows.**

In this paper we study the relationship between congestion control, switch arbitration, and fairness. Specifically, we look at fairness among different traffic flows arriving at a hot spot switch on different input ports, as CC is turned on. In addition we study the fairness among traffic flows at a switch where some flows are exclusive users of their input ports while other flows are sharing an input port (the parking lot problem).

Our results show that the implementation of congestion control in a switch is vulnerable to unfairness if care is not taken. In detail, we found that a threshold hysteresis of more than one MTU is needed to resolve arbitration unfairness. Furthermore, to fully solve the parking lot problem, proper configuration of the CC parameters are required.

I. Introduction

Traffic congestion in interconnection networks may degrade the network and the compute system performance severely if no countermeasures are taken[1], [2], [3]. Congestion is simply a result of high load of traffic fed into a network link, exceeding the link capacity at that point. Hot spot traffic patterns, network burstiness, re-routing around faulty regions, and conducting link frequency/voltage scaling (lowering the link speed in order to save power), can all lead to congestion. If all these factors are known in advance, the network administrator may alleviate the consequences by effective load balancing of the traffic, but typically this is not the case. Furthermore, in cases where multiple nodes send more data to a single destination than the node can handle, no dynamic re-routing can be done to avoid network congestion. It becomes even more severe when a parallel computer is running multiple different jobs as an on-demand service (e.g. cloud computing), where the resulting traffic pattern becomes totally unpredictable.

Congestion control (CC) as a countermeasure for relieving the consequences of congestion has been widely studied in the literature. In particular, this problem is well understood and solved by dropping network packets in traditional lossy networks such as local area networks (LANs) and wide area networks (WANs). In these environments packet loss and increased latency are indications of network congestion. Herein it is mainly TCP that implements end-to-end congestion control, either by a traditional window control mechanism [4] for detecting dropped packets or through changes in latency [5], [6]. Very often those networks are also over-provisioned in order to avoid congestion.

In high performance computing (HPC) data centers low latency is crucial, and packet dropping and retransmission are not allowed under regular circumstances, contrary to LANs and WANs, due to the loss of performance that is associated with packet drops. Lossless behavior is achieved with credit based link-level flow control, which prevents a node or a switch from transmitting packets if the downstream node or switch lacks buffer space to receive them.

Typically, when congestion occurs in a switch, a congestion tree starts to build up due to the backpressure effect of the link-level flow control. The switch where the congestion starts will be the root of a congestion tree that grows towards the source nodes contributing to the congestion. This effect is known as congestion spreading. The tree grows because buffers fill up through the switches as the switches run out of flow control credits (not necessarily in the root). As the congestion tree grows, it introduces head-of-line (HOL) blocking[7] and slows down packet forwarding that also affects flows which are not contributing to the congestion, severely degrading the entire network performance. The HOL blocked flows become victims of congestion[7].

Congestion control for link-level flow controlled networks cannot be based on a traditional window control mechanism as deployed by TCP, though it effectively limits the amount of buffer space that a flow can occupy in the network[8]. The reason for this is the relatively small bandwidth-delay product in this environment, where even a small window size may saturate the network [7]. A rate control based CC mechanism is more appropriate for link-level flow controlled networks, since it increases the range of control compared to a window based system. The mechanism relies on the switches to detect congestion, and inform the sources that contribute to the congestion that they must reduce their corresponding injection rates. There are basically two ways to inform the source nodes in such an explicit congestion notification scheme. Either the switches can mark the packets contributing to congestion in order to notify the destinations about the situation which subsequently notifies the sources (the forward explicit notification approach), or the switches

978-0-7695-4395-6/11 $26.00 © 2011 IEEE
DOI 10.1109/CCGrid.2011.67

can themselves generate notification packets that are sent directly to the source nodes (the backward explicit notification approach). The InfiniBand (IB) [9] network applies the former approach, while the emerging Data Center Bridging standard [10] is implementing the latter.

There is a body of work that propose different strategies for congestion notification and marking, e.g. a congested packet can be marked both in the input and output buffer as well as being tagged with information about the severity of the congestion. Furthermore, there are several different approaches to the design of the source response function, i.e. the actions taken to reduce the injection rate, later followed by an increase in the rate when congestion is resolved [8], [11], [12], [13].

There are also congestion control mechanisms targeting link-level flow controlled networks that take a completely different approach. Instead of removing the congestion tree itself, these approaches strive to relieve the unfortunate side effects the congestion tree has on flows not contributing to the congestion. That is, they try to remove the HOL blocking by using special set aside queues for contributors to congestion, effectively making it possible for victim flows to bypass the contributors to congestion without actually removing the congestion tree[14], [15]. Such an approach has the advantage of being able to react immediately and locally at each switch, at the cost of the extra buffers needed for the set aside queues and the added complexity in the switch to manage them. The real cause of the problem, sources injecting too much traffic into the network, is though left untouched. Furthermore, such a CC mechanism is not directly applicable in InfiniBand, the interconnection network technology we will use as a basis for our congestion control and fairness studies in this paper.

InfiniBand was standardized in October 2000 and over the years it has increased its marked share, when referring to the Top500 list [16], to 42% of the HPC market. Furthermore, 4 out of 7 Petaflop systems in the world are using InfiniBand as the system interconnect. Congestion control was added in release 1.2 of the InfiniBand specification and is to some extent based on the work done by Santos et. al. [8].

InfiniBand hardware with support for CC has been available since June 2008 [17], but the firmware required for using CC has just been released. Recently Gran et. al. presented the first experiences with CC in IB hardware, where they showed that the IB CC mechanism effectively resolves congestion and improves fairness by solving the parking lot problem, if the CC parameters are appropriately set[7]. Another significant contribution is the work done by Pfister et. al. [18], where they studied (through simulations) how well IB CC can solve certain hot spot traffic scenarios in fat tree networks.

The IB standard provides some freedom in the implementation of its CC concept; several design decisions are left to the implementer (as standards typically do). Care must be taken, however, regarding the implementation and the use of both the *threshold* and the *marking rate* CC parameters, in order to be able to resolve congestion and achieve fairness. Fairness is an important property of any interconnection network. Different traffic flows, having the same priority, should all get equal access to shared resources in the network. InfiniBand as well as other interconnection networks uses a round robin arbitration scheme[19] for selecting the next packet to be sent over a switch output port.

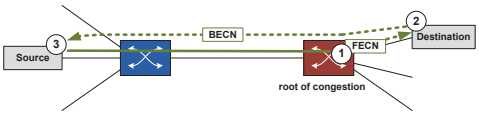


Figure 1. Congestion control in InfiniBand.

In this paper we study the relation between CC, switch arbitration, and fairness. Specifically, we look at fairness at a switch in two types of situations: I) Fairness among different traffic flows arriving at a hot spot switch on different input ports (solving the congestion problem without introducing unfairness), and II) Fairness among traffic flows where some flows are exclusive users of their input ports while other flows are sharing an input port (essentially solving the parking lot problem[20]).

We outline the parameter use and implementation of IB CC around a simulation model. In addition we present results from experiments with hardware to confirm that there is a good correlation between our simulator and the hardware behavior.

The reminder of the paper is organized as follows: Section II gives an overview of the CC mechanism supported by IB. In section III we describe our simulation model, before we in section IV and V study fairness of a IB CC capable switch in the two types of situations explained above. In section VI we show the correlation between our simulator and the hardware, before we conclude in section VII.

II. Congestion Control in InfiniBand

The IB CC mechanism, specified in the InfiniBand Architecture Specification release 1.2.1[9], is based on a closed loop feedback control systems where a switch detecting congestion marks packets contributing to the congestion by setting a specific bit in the packet headers, the *Forward Explicit Congestion Notification* (FECN) bit (fig. 1 (1)). The congestion notification is carried through to the destination by this bit. The destination registers the FECN bit, and returns a packet with the *Backward Explicit Congestion Notification* (BECN) bit set to the source (fig. 1 (2)). The source then temporarily reduces the injection rate to resolve congestion (fig. 1 (3)).

The exact behaviour of the IB CC mechanism depends upon the values of a set of CC parameters governed by a *Congestion Control Manager*. These parameters determine characteristics like when switches detect congestion, at what rate the switches will notify destination nodes using the FECN bit, and how much and for how long a source node contributing to congestion will reduce its injection rate. Appropriately set, these parameters should enable the network to resolve congestion, avoiding HOL blocking, while still utilizing the network resources efficiently.

1) Switch Features: The switches are responsible for detecting congestion and notifying the destination nodes using the FECN bit. A switch detects congestion on a given *port* and a given *Virtual Lane* (Port VL) depending on a $threshold$ parameter. If the threshold is crossed, a port may enter the Port VL congestion state, which again may lead to FECN marking of packets.

The $threshold$, represented by a weight ranging from 0 to 15 in value, is the same for all VLs on a given port, but could be set to a different level for each port. A weight of 0 indicates that no packets should be marked, while the values 1 through 15 represent a uniformly decreasing value of the threshold. That is, a value of 1 indicates a high threshold with high possibility of congestion spreading, caused by Port VLs moving into the congestion state too late. A value of 15 on the other hand indicates a low threshold with a corresponding low possibility of congestion spreading, but at the cost of a higher probability for a Port VL to move into the congestion state even when the switch is not really congested. The exact implementation of the threshold depends on the switch architecture and is left to the designer of the switch. We will in section IV see that precaution needs to be taken not to violate the fairness of the switch when implementing the threshold.

A Port VL may enter the congestion state if the threshold is crossed and it is the root of congestion, i.e. the Port VL has available credits to output data. If the Port VL has no available credits, it is considered to be a victim of congestion and shall not enter the congestion state[1]. When a Port VL is in the congestion state its packets are eligible for FECN marking. A packet will then get the FECN bit set depending on two CC parameters at the switch, the $Packet\_Size$ and the $Marking\_Rate$. Packets with a size smaller than the $Packet\_Size$ will not get the FECN bit set. The $Marking\_Rate$ sets the mean number of eligible packets sent between packets actually being marked. With both the $Packet\_Size$ and the $Marking\_Rate$ set to 0, all packets should get the FECN bit set while a Port VL is in the congestion state.

[1]If the $Victim\_Mask$ is set for the port, then the switch will move the Port VL into the congestion state independently of the number of available credits. The $Victim\_Mask$ is typically set for switch ports connection HCAs to the switch as an HCA will never detect congestion itself.

2) Channel Adapter Features: When a destination CA receives a packet with a FECN bit, the CA should as quickly as possible notify the source of the packet about the congestion. This is done by returning a packet with the BECN bit set back to the source. The packet with the BECN bit could either be an acknowledgement packet (ACK) for a reliable connection or an explicit *congestion notification packet* (CNP). In either case it is important that the ACK or the CNP is sent to the source as soon as possible to ensure a fast response to the congestion.

When a source CA receives a packet with the BECN bit set, the CA lowers the injection rate of the corresponding traffic flow. To determine how much and for how long the injection rate should be reduced, the CA uses a *Congestion Control Table* (CCT) and a set of CC parameters. The CCT holds *injection rate delay* (IRD) values that define the delay between consecutive packets sent by a particular flow (the IRD calculation being relative to the packet length). Each flow with CC activated holds an index into the CCT, the $CCTI$. When a new BECN arrives, the $CCTI$ of the flow is increased by $CCTI\_Increase$. The CCT is usually populated in such a way that a larger index yields a larger IRD. Then consecutive BECNs increase the IRD which again decreases the injection rate. The upper bound of the $CCTI$ is given by $CCTI\_Limit$.

To increase the injection rate again, the CA relies on a $CCTI\_Timer$, maintained separately for each SL of a port. Each time the timer expires, the $CCTI$ is decremented by one for all associated flows. When the $CCTI$ of a flow reaches zero, the flow no longer experience any IRD.

The IB CC can operate either at the Service Level (SL) or at the Queue Pair (QP) level at an HCA. Any lowering of the injection rate as a result of BECN reception, then affects the whole SL or the single QP depending on the level of CC operation. While operating at the SL level may require less resources at the HCA than operating at the QP level, choosing the SL level will have a negative impact on both fairness and performance. The reason is that a single traffic flow contributing to congestion will lower the injection rate of all traffic flows within the same SL at the HCA. This could include traffic flows not contributing to the congestion at all as they are not going through the root of the congestion tree, but headed for other parts of the network. This type of unfairness is avoided if the CC operates at the QP level. We will not consider this type of unfairness any further in this paper, as we focus on the fairness provided by switches running CC.

III. The Simulation Model

Our network simulator and switch model is built on the OMNet++ platform[21]. It is based on the IB model made available to the OMNeT++ community by Mellanox Technologies Ltd in 2007/2008. We have ported this model

to the OMNeT++ 4 environment, made several bug fixes, implemented CC support, and added some general extensions to it to make it more suitable for our studies.

Below we give a brief overview of the features and the detail level of our simulation model. A more detailed description of the simulator is given in [22].

A. The IB Model

The IB model consists of a set of modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear forwarding tables.

The two building blocks for creating networks using the IB model are the *Host Channel Adapter* (HCA) module and the *Switch* module. During a simulation an HCA represents both a traffic generator, traffic injector and a traffic sink in the network, while a *Switch* acts as a forwarding node. The HCA supports several traffic generation schemes, e.g. varying the injection rate, the packet size and the destination node distribution. The *Switch* is modelled as an input buffer architecture with support for virtual lanes, virtual output queuing (VoQ) and virtual cut through switching. It uses round robin arbitration over the different VLs and multiple input ports.

The InfiniBand Congestion Control mechanism is implemented by the *Congestion Control ManaGeR* (CCMGR) module. The CCMGR is part of the HCA and the *Switch*, and manages everything related to congestion control in these modules in compliance with IB CC as described in the previous section.

IV. Fairness - Type I

In the type I situation we look at the fairness among traffic flows arriving at a switch on different input ports, creating congestion as they are headed for the same output port. In an IB CC capable network, the switch will detect congestion as soon as the CC threshold is crossed, and mark the contributing packets to tell the sources about the contention at the switch.

Round robin (RR)[23], [19] is thought to be a fair arbitration scheme for the type I situation. All input ports at a switch are served in a round robin fashion. An input port currently accessing an output port, will not get access to the same output port again until all other input ports requesting the same output port has been granted access. We will use an example to illustrate the fairness of the RR scheme. Figure 2 shows a switch connecting seven end nodes, where all links have the same bandwidth. Now, let us add traffic flows to the network as indicated by the arrows in the figure, and study the throughput of the different flows. The flows are added one by one, with one second intervals, until all five flows are active. The startup sequence follows the numeric ordering, starting with *H1*. Notice that one flow, the one from *H1*, is headed for the node *H4*, while the four other flows are all headed for the same destination, *H5*.

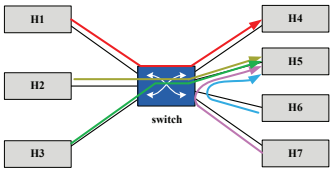


Figure 2. Topology and traffic flows.

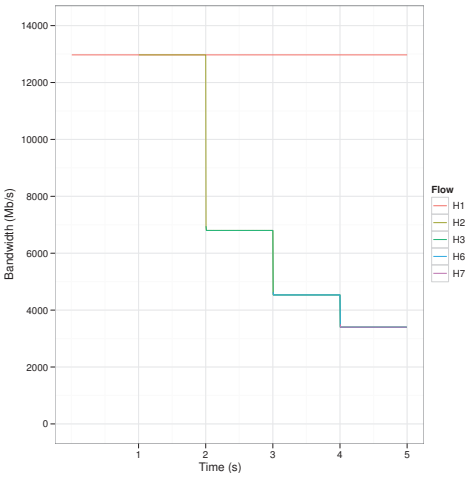


Figure 3. Throughput - No congestion control.

Figure 3 shows the throughput of the five traffic flows during a simulation. The flow from *H1* to *H4* achieves a steady throughput of 13Gbps during the whole simulation. This is as expected as this traffic flow, being the only one headed for *H4*, is independent of all the other traffic flows. The throughput of this flow will serve as a reference as we now add the four flows headed for *H5*. After one second we add the first one, that is, the one originating from *H2*. This flow also maintains a 13Gpbs throughput for one second. Then, at 2s, we add the second traffic flow towards *H5*. Now the two flows headed for *H5* each experience a drop in performance down to half the capacity of the bottleneck link, the link from the switch to *H5*. Each flow is given the same access to this link, that is, the arbitration scheme is fair. As we add the third and fourth flow headed for *H5*, we see a corresponding drop in performance for each new additional flow. However, the available bandwidth at the bottleneck link is evenly shared among the flows, that is, the RR scheme is fair.

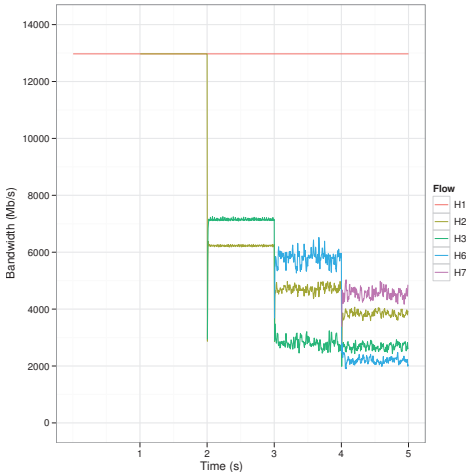


Figure 4. Throughput - Congestion control enabled.

A. Congestion Control Using One Threshold

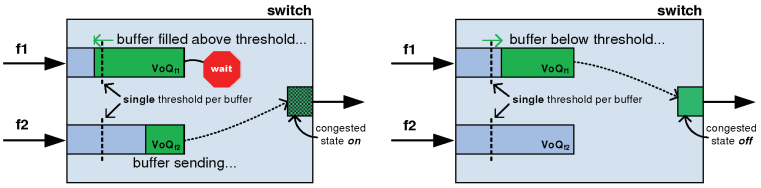
Now, let see what happens to fairness when we enable congestion control (CC) (this small topology could be part of a greater network where it has been found reasonable to use CC). Figure 4 shows the throughput of the five traffic flows running the same simulation as before, but this time with CC turned on. As we can see, even if the RR arbitration scheme is fair, the CC has a negative impact on the fairness. When we at 2s add the flow from *H3*, this flow is able to stabilize at a throughput of a little more than 7Gbps. This is about 1Gbps higher than the other flow going to *H5*, the flow from *H2*. Then, when we add *H6* at 3s we see that the unfairness is even more dramatic. This time the amount of traffic the flow from *H6* is able to get through the bottleneck link is twice as much as what the flow *H3* is able to get through. The flow *H2* settles in between at a little less than 5Gpbs. Enabling CC has introduced unfairness in the network, even if the arbitration scheme in the switch itself is fair. When we add the last flow, *H7*, at 4s the situation continues; the unfairness is still present.

The reason for the observed unfairness in figure 4 has been identified to be the use of a single threshold to detect congestion in the switch. Let us have a closer look at what is happening inside the switch as different traffic flows headed for *H5* are active. First, at any time when only one of the flows is active, no matter how much traffic is sent, the buffers will not fill and no congestion control marking occurs (assuming that all links have the same capacity, and that the end node *H5* is able to remove the traffic from the network as soon as it arrives). When a second flows is active, however, the one flow with the highest injection rate will fill its virtual output queue (VoQ) buffer first[2]. This situation is shown in figure 5a. The figure shows two different input ports of a switch, and the virtual output queues of these two ports, VoQ_{f1} and VoQ_{f2}, corresponding to the shown output port (the one connecting *H5* to the switch in our scenario). The threshold is shown as a vertical dotted line. Here *f1* is the flow with the highest injection rate, and hence the flow filling its VoQ the fastest. Naturally, the fill ratio of VoQ_{f1} crosses the threshold at a time where the other flow is given access to the output port. The crossing of the threshold moves the output port into the congested state, and by that, packets being forwarded on this output port will be marked as contributing to congestion. While in this state, the RR arbitration in the switch will alternate between giving the two flows access to the output port, resulting in each flow experiencing approximately the same amount of congestion control marking. The two flows will lower their injection rates correspondingly. As the flow *f1* initially was the most aggressive flow, and both flows receives approximately the same amount of congestion control feedback, the flow *f2* is actually likely to empty VoQ_{f2} before the fill ratio of VoQ_{f1} is below the threshold. When this happens, *f1* is given sole access to the output port. Now having an injection rate lower than the maximum capacity of the link, the VoQ_{f1} will be emptied below the single threshold and the output port will be moved out of the congestion state, as figure 5b shows.

All in all, the situation is somewhat controlled by the most aggressive flow, *f1*. Exactly how many packets that will be marked as contributing to congestion from each of the two flows *f1* and *f2*, depends on the traffic flow characteristics, packet and buffer sizes, and the threshold chosen. Our simulation studies show, however, that in the common situation the different flows receive approximately the same amount of congestion control information. Keeping this in mind, also notice that two different traffic flows governed by congestion control might very well stabilize at different injection rate levels, even if they receive the same amount of congestion control information. This could happen if one of the flows, when congestion occurs, is initially sending at a higher injection rate than the other one. Then, as both flows are throttled by the same amount of congestion control information, the net result is that the two flows are stabilizing at different injection rates after the throttling as well.

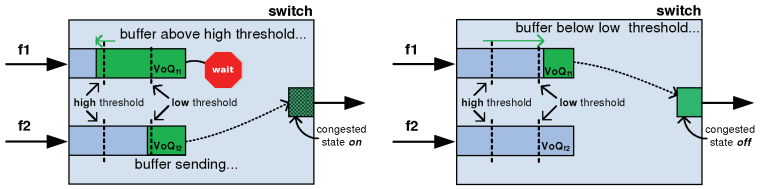
Now, look at figure 4 again. Each time we add a new flow contributing to congestion, the new flow starts out with a higher injection rate than the already contributing flows. This is exactly the situation explained in the previous section, and the result is quite visible in the figure: the flow given the largest share of the bottleneck link is always the last flow added - a flow that started its injection at full bandwidth.

[2]In general, different end nodes may very well inject traffic into the network at different injection rates.



(a) Output port moved into congested state. (b) Output port released from congested state.

Figure 5. Congestion control using a single threshold.



(a) Output port moved into congested state. (b) Output port released from congested state.

Figure 6. Congestion control using two thresholds.

More generally, an aggressive flow always being the last one to empty its VoQ below the threshold, could benefit from this behavior by constantly getting more than its fair share of access to the bottleneck link. This is the behavior observed in figure 4. One way to avoid this unfortunate situation is to introduce a second threshold.

B. Congestion Control Using Two Thresholds

Let us now study the situation in figure 2 again, this time with two thresholds per VoQ. Introducing a second threshold, each VoQ now has a lower and upper threshold, as shown in figure 6. When the fill ratio climbs above the high threshold the corresponding output port is moved into the congested state, as shown in figure 6a. The output port is then not released from this state until the fill ratio is lower than the low threshold, figure 6b. By using these two thresholds we ensure that congestion control information continue to be sent for an extended period of time, even after the fill ratio is below the upper threshold. It is no longer possible for a flow to first trigger congestion marking, and then immediately release congestion again as soon as it itself is being granted access to the output port. In particular, an aggressive flow will experience congestion control marking for an extended period of time, even when it has sole access to the output port.

Figure 7 shows the same simulation as before, this time using two thresholds instead of one. As we can see from the figure, fairness is now restored. Each time a new contributor to congestion is added, all contributors quickly settle for their fair share of the congested link, even if the newly added contributor initially was injecting traffic at a much

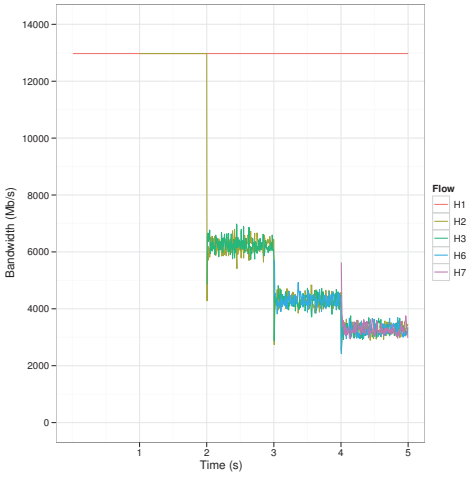


Figure 7. Throughput - Congestion control using two thresholds.

higher rate than the others. The use of two thresholds ensures that any aggressive flow triggering congestion at a VoQ in a switch, will receive a correspondingly high amount of congestion control information, that is, BECNs.

Simulations have shown that an hysteresis of more than one MTU (maximum transmission unit) is needed to resolve unfairness. The distance between the upper and lower threshold used during the simulations shown in figure 7 was a little

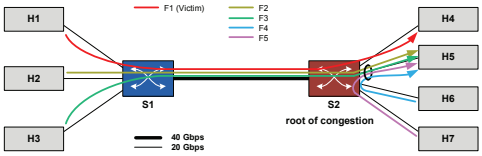


Figure 8. Topology and traffic flows (II).

more than 3 MTUs.

C. A common set of thresholds

For simplicity, each VoQ in figure 6 is shown to have its own set of thresholds. Then CC is triggered as soon as the threshold is crossed in at least one of the VoQs. Another approach is to use a common threshold for all VoQs related to a given output port. Then this common threshold is compared against the sum of the buffer occupancy of all corresponding VoQs. Using a common threshold, there is no differentiation between a single flow occupying a large part of its VoQ, and by that solely triggering congestion marking, and several flows filling smaller parts of their VoQ, but together occupying enough buffer space to trigger congestion marking. Our simulations shows that the behavior of the two approaches are quite similar. What matters is that the threshold parameter is mapped to a fill ratio of the VoQ(s) that ensures fast and proper congestion detection. Special care must be taken when implementing the common threshold approach, to make sure that the threshold maps to a low enough fill ratio to allow for one single VoQ to trigger congestion by its own. This is important as a single flow or VoQ could create congestion alone in certain situations. E.g. a source node injecting traffic into the network faster than the destination node can handle, could singlehandedly create a congestion tree with the root of the tree at the last switch prior to the destination. A single VoQ could also very easily create congestion alone in a network where different links have different bandwidths. We will see an example of this in the next section.

V. Fairness - Type II

Let us turn our attention towards the type II situation. In this scenario some traffic flows are sharing an input port, while other flows are the exclusive users of their input ports. By extending our topology from figure 2 with a second switch, we can exemplify such a situation by adding the traffic flows shown in figure 8. Notice that the switch to switch link has twice the capacity of the other links. Now, at the switch $S2$, the two flows from $H2$ and $H3$ headed for $H5$ are sharing an input port and the VoQ corresponding to the link towards $H5$, while the two traffic flows from $H6$ and $H7$ are exclusive users of their input ports. The Round Robin arbitration scheme provides each input port with the same fraction of the output bandwidth, resulting in an unfair arbitration for the flows heading to $H5$. The two flows from $H2$ and $H3$ will by $S2$ be seen as one flow since they share the same VoQ. Then, implementing RR arbitration, the switch $S2$ will grant access to the link towards $H5$ as if there were only three traffic flows requesting access to it instead of four. The result is that the flows from $H6$ and $H7$ each get 1/3 of the total bandwidth of the link between $S2$ and $H5$, while the last 1/3 is shared by the two flows from $H2$ and $H3$ giving them 1/6 of the bandwidth each. This unfairness, due to the sharing of input ports and buffers by *some* flows at an RR based switch, is often referred to as *the parking lot problem*[24], [20].

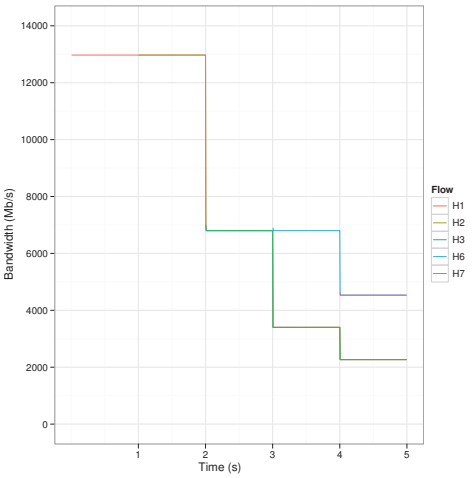


Figure 9. Throughput - No congestion control (II).

Figure 9 shows the throughput achieved by the different traffic flows when simulating the situation of figure 8. As before, a new traffic flow is added to the network each second, starting with the flow from $H1$. The unfairness caused by the parking lot problem is clearly visible as soon as we add the flows from $H6$ and $H7$ at 3s and 4s respectively. During the time period from 3s to 4s, $H6$ achieves the same throughout as $H2$ and $H3$ combined. When $H7$ is added at the time 4s, both $H6$ and $H7$ achieves twice the throughput of $H2$ and $H3$. The parking lot problem is evident. Notice also that HOL blocking is present. As soon as we add the flow from $H3$, a congestion tree builds from $S2$ through $S1$ towards the sources, and the flow from $H1$ is not able to progress any faster than the contributors to congestion $H2$ and $H3$[7].

Most interconnection networks like InfiniBand are using Round Robin arbitration between inputs of the switch. The worst possible unfairness happens when all nodes send

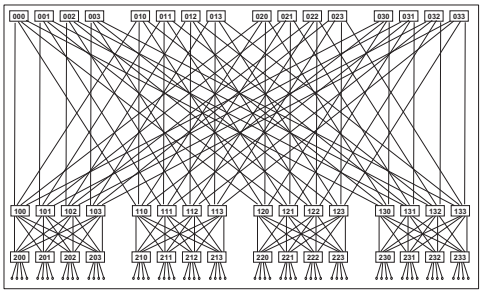


Figure 10. An example 4-ary 3-tree.

data to a single node. Formally, assuming traffic is flowing through M out of the $N-1$ input ports towards an output port, the RR arbitration will provide $1/M$ of the output bandwidth to each input. In the single switch scenario with a network diameter of two, this is fair. In a larger network, however, the M_{s_i} flows from a switch S_i sharing the output port towards the switch S_j, will at S_j together only be assigned $1/M_{s_j}$ of the capacity of the next output port (given that they are all headed for the same output port at S_j as well). That is, in the worst case scenario the unfairness increases by M_{s_x} for each additional switch on the path from the source towards the destination. For large MIN networks where M approaches $N-1$, the worst possible unfairness in bandwidth allocation is: $(N-1)^{D-2}$ where D is the network diameter. For network topologies like k-ary n-trees[25], the exact ratio depends on the routing used. For D-Mod-K routing [26], [27], each down link in the network carries traffic to a single destination. In the example 4-ary 3-tree provided in figure 10, there are 3 first level neighbors to the destination, $4^2-4=12$ second level neighbors and $4^3-4^2=48$ third level neighbors. With RR arbitration, the bandwidth fraction provided is $1/4$ and $1/4/4/4=1/64$ for the first and second level neighbors, respectively. For the third level neighbors, as they all go through the same spine port, the bandwidth fraction for each source is $1/4/4/48=1/768$. More formally, for a k-ary n-tree the furthest sources will in the worst case scenario receive only $1/k^{n-1}/(k^n-k^{n-1})=1/(k^{2n-1}-k^{2n-2})$ of the bandwidth of the last link to the destination, while a first level neighbor will receive $1/k$. It is imminent that with high radix switches the unfairness is so high that it may actually cause transport timeouts in cases of many to one communication. Similarly, timeouts could happen due to unfairness in a network even with low radix switches if the diameter is high.

Some alternatives to RR arbitration, that could solve the demonstrated unfairness, rely on the number of different flows through an input port as a weight for the arbitration.

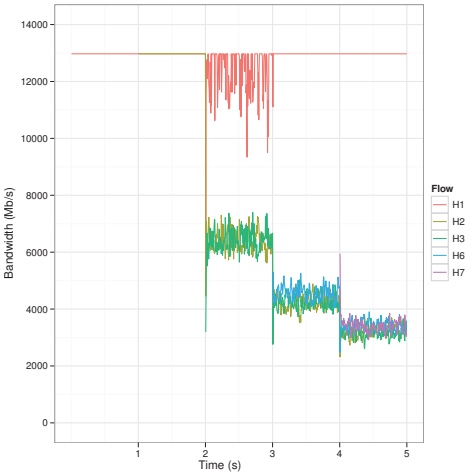


Figure 11. Throughput - Congestion control turned on (II).

The incurred cost of such a solution presented by [28] makes it impractical for Interconnection Networks - where the number of different flows is $O(N^2)$.

The topology in figure 8 is suitable for studying the characteristics of CC in a controlled manner. At the same time, recognize that this topology could be part of a larger topology similar to the 4-ary 3-tree in figure 10. If we now rerun our simulation of the scenario shown in figure 8 with CC enabled, we get the results shown in figure 11. The four traffic flows contributing to congestion are now equalized. The HOL blocking is gone and the parking lot problem has been resolved. Recall from section IV that any aggressive flow will experience more congestion marking than less aggressive flows, given that the CC mechanism is properly implemented using hysteresis, that is, two thresholds. Then, with the introduction of CC, using a marking rate of 1[3], the close neighbors are punished as they are able to pass more traffic through the congested port. The result is that, on average, all flows will be throttled to provide the same bandwidth, as can be seen from figure 11.

While the introduction of IB CC has the potential of solving the parking lot problem, the degree of success in solving this problem is not only related to the hardware implementation of the IB CC, but also the values chosen for the IB CC parameters. E.g. choosing an unfortunate marking rate or $CCTI\_Timer$ could result in the contributors to congestion being slow in settling for their fair share of the bottleneck link. An example is given in figure 12. Here

[3]The marking rate rule applied marks every packet going through a congested link with probability $P(1/(marking\_rate+1))$. This adheres to the IB CC specification.

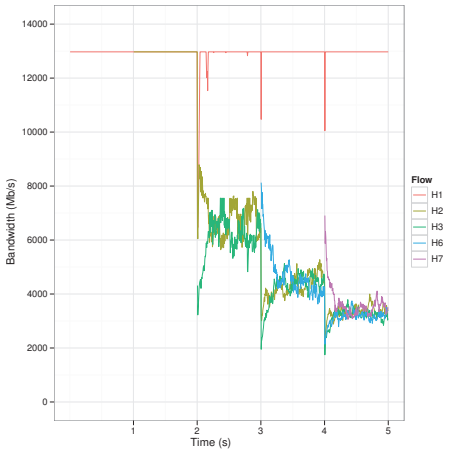


Figure 12. Throughput - marking rate 10.

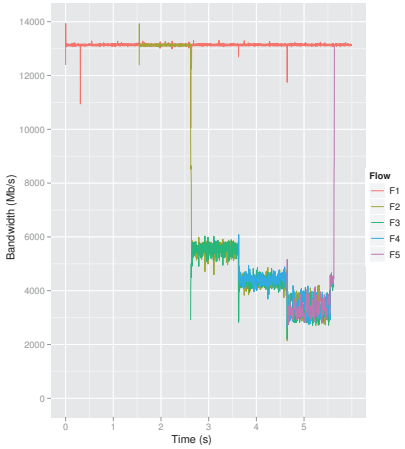


Figure 13. Throughput from hardware - Congestion control turned on.

the marking rate is set to 10, while the $CCTI\_Timer$ is set 10 times as high as before to compensate for the new marking rate. While the parking lot problem is still to some degree solved, an unfairness is present in the network for an extended period of time each time a new contributor is added. How to set the IB CC parameters, possibly independent of topology and traffic patterns, is a subject of ongoing research. Both simulation studies and hardware experiments[7] indicate, however, that the marking rate should be kept low, preferable at 1, to ensure high utilization of network resources and fairness.

VI. A Comparison with Hardware Experiments

In the studies we have presented in this paper we have been using a simulation model to examine the relation between CC, arbitration and fairness. The use of a simulation model was necessary because no hardware that we know of gives us the same flexibility as simulations when it comes to changing the internal behavior of the switch, as done in our study. When using simulations, however, it is important to validate, to the extent possible, the correctness of the model with its real world equvivalent. We have done a general validation of our simulation model against real hardware as documented in[22]. Furthermore, we have compared the simulation results from the threshold implementation using hysteresis (two common threshold values for all VoQs corresponding to a given output port), to hardware experiments using IB CC capable hardware from Mellanox Technologies and Sun Microsystems, now Oracle. A hardware test bed corresponding to the topology showed in figure 8 was put together using two Mellanox InfiniScale IV switches and seven Mellanox ConnectX Host Channel Adapters equipped in seven Sun Fire X2200 M2 hosts. Figure 13 shows the hardware experiment results corresponding to the simulation results shown in figure 11. As the figures show there is a strong correlation between our simulation results and the hardware results, which confirms our confidence in the simulation model.

VII. Conclusions

Switch arbitration and its relation to fair utilization of link bandwidth has been studied for decades. Congestion control in interconnection networks on the other hand is far less understood. This is particularly true for its relation to various fairness aspects.

Ideally, the local switch-fairness provided by well functioning switch arbitration should work independently of the more global stream fairness provided by end-to-end congestion control. In this paper we have demonstrated and explained why this is not always the case. Through simulations calibrated with hardware measurements, we have shown that a straightforward implementation of IB congestion control leads to unfairness even in a simple one switch scenario. Furthermore this unfairness is unstable in the sense that the distribution of bandwidth to the different flows depends on the utilization of the different flows that just happened to prevail at the instance of time when congestion occurred.

We have demonstrated that a good solution to the above problem is to introduce two congestion control marking thresholds. We have also shown that this technique solves the parking lot problem[20] and is quite robust with regards to parameter settings. The detailed relation between our results and CC-parameters settings is still under investigation. Preliminary results indicate that our main results prevail,

but that the marking rate influences the speed by which the system converges to a stable state after a change of traffic pattern. A full investigation of this is, however, left for future work.

REFERENCES

[1] G. F. Pfister and V. A. Norton, ""Hot Spot" contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. 34, no. 10, pp. 943–948, 1985.

[2] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[3] P. García, J. Flich, J. Duato, I. Johnson, F. Quiles, and F.Naven, "Dynamic evolution of congestion trees: Analysis and impact on switch architecture," in *High Performance Embedded Architectures and Compilers*, 2005, pp. 266–285.

[4] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*. ACM, 1988, pp. 314–329.

[5] L. S. Brakmo and L. L. Peterson, "TCP vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465–1480, 1995.

[6] C. Parsa and J. Garcia-Luna-Aceves, "Improving TCP congestion control over internets with heterogeneous transmission media," in *7th International Conferance on Network Protocols (ICNP99)*. IEEE Computer Society, 1999, pp. 213–221.

[7] E. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1–12.

[8] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-end congestion control for InfiniBand," in *INFOCOM*, 2003.

[9] *Infiniband architecture specification*, 1st ed., InfiniBand Trade Association, November 2007.

[10] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification.*, IEEE 802.1Qau-2010 ed., IEEE 802 LAN/MAN Standards Committee, 2010. [Online]. Available: http://www.ieee802.org/1

[11] J. R. Santos, Y. Turner, and G. J. Janakiraman, "Evaluation of congestion detection mechanisms for InfiniBand switches," in *IEEE GLOBECOM – High-Speed Networks Symposium*, 2002.

[12] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Congestion management in MINs through marked and validated packets," in *PDP*, 2007, pp. 254–261.

[13] ——, "On the influence of the packet marking and injection control schemes in congestion management for MINs," in *Euro-Par*, 2008, pp. 930–939.

[14] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 108–119.

[15] J. Escudero-Sahuquillo, P. García, F. Quiles, J. Flich, and J. Duato, "FBICM: Efficient congestion management for high-performance networks using distributed deterministic routing," in *High Performance Computing - HiPC 2008*, ser. Lecture Notes in Computer Science.

[16] "Top 500 supercomputer sites," http://top500.org/, Nov. 2010.

[17] Mellanox Technologies, "Mellanox announces availability of industry's first 40Gb/s InfiniBand switch silicon device and product reference platforms," Press release, June 2008, http://www.mellanox.com/content/pages.php?pg=press_release_item&rec_id=214.

[18] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using InfiniBand architecture congestion control," Invited paper in High Performance Interconnects for Distributed Computing, july 2005.

[19] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[20] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004, ch. 15.4.1, pp. 294–295.

[21] "Omnet++ network simulation framework," http://www.omnetpp.org/.

[22] E. G. Gran and S.-A. Reinemo, "InfiniBand congestion control, modelling and validation." OMNeT++ 2011, Barcelona, Spain.

[23] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[24] M. Galles, "Spider: A high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, pp. 34–39, 1997.

[25] S. Ohring, M. Ibel, S. Das, and M. Kumar, "On generalized fat trees," in *Parallel Processing Symposium, 1995. Proceedings., 9th International*, Apr. 1995, pp. 37–44.

[26] C. Gomez, F. Gilabert, M. Gomez, P. Lopez, and J. Duato, "Deterministic versus adaptive routing in fat-trees," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1 –8.

[27] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, "Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns," in Concurrency and Computation: Practice and Experience, vol. 22, no.2, pp. 217 –231, 2009.

[28] P. Gratz, B. Grot, and S. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, 2008, pp. 203 –214.

Paper III

First Experiences with Congestion Control in InfiniBand Hardware

Ernst Gunnar Gran, Magne Eimot, Sven-Arne Reinemo,
Tor Skeie, Olav Lysne, Lars Paul Huse, and Gilad Shainer

First Experiences with Congestion Control in InfiniBand Hardware

Ernst Gunnar Gran,
Magne Eimot, Sven-Arne Reinemo,
Tor Skeie, Olav Lysne *Member, IEEE*
Simula Research Laboratory
Fornebu, Norway
Email: {ernstgr, magneei, svenar, tskeie, olavly}@simula.no

Lars Paul Huse
Sun Microsystems
Email: Lars.Paul.Huse@sun.com

Gilad Shainer
Mellanox Technologies
Email: Shainer@Mellanox.com

***Abstract*—In lossless interconnection networks congestion control (CC) can be an effective mechanism to achieve high performance and good utilization of network resources. Without CC, congestion in one node may grow into a congestion tree that can degrade the performance severely. This degradation can affect not only contributors to the congestion, but also throttles innocent traffic flows in the network. The InfiniBand standard describes CC functionality for detecting and resolving congestion. The InfiniBand CC concept is rich in the way that it specifies a set of parameters that can be tuned in order to achieve effective CC. There is, however, limited experience with the InfiniBand CC mechanism. To the best of our knowledge, only a few simulation studies exist. Recently, InfiniBand CC has been implemented in hardware, and in this paper we present the first experiences with such equipment. We show that the implemented InfiniBand CC mechanism effectively resolves congestion and improves fairness by solving the parking lot problem, if the CC parameters are appropriately set. By conducting extensive testing on a selection of the CC parameters, we have explored the parameter space and found a subset of parameter values that leads to efficient CC for our test scenarios. Furthermore, we show that the InfiniBand CC increases the performance of the well known *HPC Challenge* benchmark in a congested network.**

I. Introduction

Congestion in interconnection networks may degrade performance severely if no countermeasures are taken[1], [2], [3]. Congestion is simply a result of too much traffic fed into a network link, exceeding link capacity at this point. Hot spot traffic patterns, rerouting around faulty regions, and conducting link frequency/voltage scaling (lowering the link speed) in order to save power, can all lead to congestion. If all these factors are known in advance, the network administrator might alleviate the consequences by effective load balancing of the traffic, but typically this is not the case. It becomes even more difficult when a parallel computer is running multiple different jobs as an on-demand service (embedding virtual servers), where the resulting traffic pattern becomes unpredictable.

Congestion control (CC) as a countermeasure for relieving the consequences of congestion has been widely studied and debated in the literature. In particular, this problem is well understood and solved in traditional lossy networks such as local area (LANs) and wide area networks (WANs). In these environments packet loss and increased latency are indications of network congestion. Herein it is mainly TCP that implements end-to-end congestion control, either by a traditional window control mechanism [4] for detecting dropped packets or through changes in latency [5], [6]. Very often those networks are also overprovisioned in order to avoid congestion.

In high performance computing (HPC) low latency is crucial and packet dropping and retransmission are not allowed under regular circumstances, contrary to LANs and WANs. Lossless behaviour is achieved with credit based link-level flow control, which prevents a switch from transmitting packets if the downstream switch lacks buffer space to receive them.

Typically, when congestion occurs in a switch, a congestion tree starts to build up due to the back pressure effect of the link-level flow control. The switch where the congestion starts will be the root in a congestion tree that grows towards the source nodes contributing to the congestion. This effect is known as congestion spreading. The tree grows because buffers fill up through the switches as the switches run out of credits (not necessarily in the root). As the congestion tree grows, it introduces head-of-line (HOL) blocking and slows down packet forwarding, also affecting flows not contributing to the congestion, severely degrading the network performance. Figure 1 shows how three flows destined for the node H5 create congestion at switch S2. A congestion tree builds up from S2 (fig.2, solid arrows). The flow headed for H4 (fig.1) is blocked, even if that flow is not requesting the congested link from S2 to H5. This HOL blocking not only limits the transmission rate of the flow destined to a non-congested link, but also makes the congestion tree grow further (fig. 2, dotted arrow). The HOL blocked flow has become a *victim* of congestion.

Congestion control for link-level flow controlled networks cannot be based on a traditional window control mechanism as deployed by TCP, though it effectively limits the amount

This work is in part financed by Sun Microsystems, Inc.

978-1-4244-6443-2/10/$26.00 ©2010 IEEE

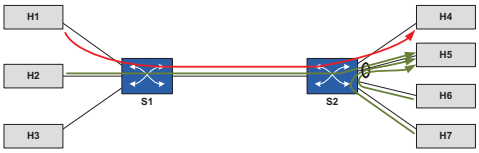


Figure 1. Congestion in an interconnection network.

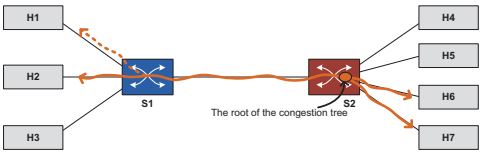


Figure 2. A congestion tree in an interconnection network.

of buffer space that a flow can occupy in the network (and otherwise offers the benefit that packet injection is self-clocked), as discussed by [7]. The reason for this is the relatively small bandwidth-delay product in this environment. If we assume a network with 1 Gigabyte/sec links, 64 ns switch forwarding delay, and a diameter of 32 switches we get a bandwidth-delay product of 2048-bytes, which might be just one single packet [7]. This means that a flow (of 2048-byte packets) limited to the window size of one packet can roughly use all the bandwidth through the network, and a window size of two will saturate the network. For link-level flow controlled networks a rate control mechanism is more appropriate, since it increases the range of control compared to a window based system. The mechanism relies on the switches to detect congestion (the root of the congestion tree) and inform the sources that contribute to the congestion that they must reduce the injection rate. There are basically two ways to inform the source nodes. Either the switches mark the packets contributing to congestion in order to notify the destinations about the situation which subsequently notifies the sources (the forward explicit notification approach), or the switches themselves generate a notification packet that is sent directly to the source nodes (the backward explicit notification approach). InfiniBand (IB) [8] applies the former, while the emerging Data Centre Bridging standard [9] (Ethernet) seemingly is to implement the latter. There is a body of work that propose different strategies for congestion notification and marking, e.g. a congested packet can be marked both in the input and output buffer as well as being tagged with information about the severity of the congestion. Furthermore, there are several different approaches to the design of the source response function, i.e. the actions taken to reduce the injection rate, later followed by an increase in the rate when congestion is resolved [7], [10], [11], [12]. In this paper we will confirm to the congestion control strategy specified by InfiniBand.

InfiniBand [8] was standardised in October 2000 and over the years it has increased its marked share, when referring to the Top500 list [13], to 30% of the market. For the top 20 super computers 45% is based on IB. Congestion control was added in release 1.2 of the IB specification and is to some extent based on the work done by Santos et. al. [7]. Only a few contributions have assessed the effect of the IB CC and how to use the various CC parameters. The most significant contribution is the work done by Pfister et. al. [14], where they studied (through simulations) how well IB CC can solve certain hot spot traffic scenarios in fat trees.

InfiniBand hardware with support for CC has been available since June 2008 [15], [16], but the firmware required for using CC is still not generally available. To the best of our knowledge there are no published results on experience with such hardware. In this paper we present experimental results with CC on the latest generation of IB hardware. Moreover, we add insight on how to use the CC parameters by exploring a large set of parameter values. We will also show how CC can benefit the well known HPCC test benchmark. The reminder of the paper is organised as follows: Section II gives an overview of the CC mechanism supported by IB. In section III we describe our test bed and the hardware and software used, while Section IV gives a detailed description of our experiment set ups. Our results are presented in Section V, VI, and VII. Section V presentes our results from using CC to reduce the negative impact of congestion, while Section VI presents similar results for the HPCC benchmark. Section VII presents our results from a study of the IB CC parameter value space and how to select optimal values for these parameters. Finally, in Section VIII we give our conclusions.

II. The CC Concept in InfiniBand

In this section we give an overview of the IB CC mechanism as specified in the InfiniBand Architecture Specification release 1.2.1 [8]. As our studies focus on CC capable equipment only, the parts of the specification defining credit starvation to support legacy devices will not be covered[1].

The IB CC mechanism is based on a closed loop feedback control systems where a switch detecting congestion marks packets contributing to the congestion by setting a specific bit in the packet headers, the *Forward Explicit Congestion Notification* (FECN) bit (fig. 3 (1)). The congestion notification is carried through to the destination by this bit. The destination registers the FECN bit, and returns a packet with the *Backward Explicit Congestion Notification* (BECN) bit

[1][8] also specifies functionality and parameters to perform monitoring and logging of the congestion control mechanism in the IBA, but as these features have not been extensively used during our experiments we will not touch upon them any further in this section.

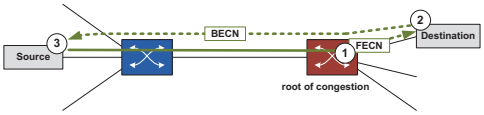


Figure 3. Congestion control in InfiniBand.

set to the source (fig. 3 (2)). The source then temporarily reduces the injection rate to resolve congestion (fig. 3 (3)).

The exact behaviour of the IB CC mechanism depends upon the values of a set of CC parameters governed by a *Congestion Control Manager*. These parameters determine characteristics like when switches detect congestion, at what rate the switches will notify destination nodes using the FECN bit, and how much and for how long a source node contributing to congestion will reduce its injection rate. Appropriately set, these parameters should enable the network to resolve congestion, avoiding head-of-line blocking, while still utilizing the network resources efficiently.

A. Congestion Control at a Switch

The switches are responsible for detecting congestion and notifying the destination nodes using the FECN bit. A switch detects congestion on a given *port* and a given *Virtual Lane* (Port VL) depending on a $threshold$ parameter. If the threshold is crossed, a port may enter the Port VL congestion state, which again may lead to FECN marking of packets.

The $threshold$, represented by a weight ranging from 0 to 15 in value, is the same for all VLs on a given port, but could be set to a different level for each port. A weight of 0 indicates that no packets should be marked, while the values 1 through 15 represent a uniformly decreasing value of the threshold. That is, a value of 1 indicates a high threshold with high possibility of congestion spreading, caused by Port VLs moving into the congestion state too late. A value of 15 on the other hand indicates a low threshold with a corresponding low possibility of congestion spreading, but at the cost of a higher probability for a Port VL to move into the congestion state even when the switch is not really congested. The exact implementation of the threshold depends on the switch architecture and is left to the designer of the switch.

A Port VL enters the congestion state if the threshold is crossed and it is the root of congestion, i.e. the Port VL has available credits to output data. If the Port VL has no available credits, it is considered to be a victim of congestion and shall not enter the congestion state unless a specific $Victim\_Mask$ is set for the port. The $Victim\_Mask$ is typically set for ports connecting a *channel adapter* (CA) to the switch. A CA that is not able to process received packets fast enough will not consider itself to be a root of congestion even if a congestion tree then builds up with the CA as the root. In this special case the Port VL at the switch connecting the CA should consider itself to be the root of congestion, even if it is actually a victim, and move into the congestion state.

When a Port VL is in the congestion state its packets are eligible for FECN marking. A packet will then get the FECN bit set depending on two CC parameters at the switch, the $Packet\_Size$ and the $Marking\_Rate$. Packets with a size smaller than the $Packet\_Size$ will not get the FECN bit set. The $Marking\_Rate$ sets the mean number of eligible packets sent between packets actually being marked. With both the $Packet\_Size$ and the $Marking\_Rate$ set to 0, all packets should get the FECN bit set while a Port VL is in the congestion state.

B. Congestion Control at a Channel Adapter

When a destination CA receives a packet with a FECN bit, the CA should as quickly as possible notify the source of the packet about the congestion[2]. As earlier mentioned, this is done by returning a packet with the BECN bit set back to the source. The packet with the BECN bit could either be an acknowledgement packet (ACK) for a reliable connection or an explicit *congestion notification packet* (CNP). In either case it is important that the ACK or the CNP is sent to the source as soon as possible to ensure a fast response to the congestion.

When a source CA receives a packet with the BECN bit set, the CA lowers the injection rate of the corresponding traffic flow. That is, the injection rate of either the related *queue pair* (QP) or the corresponding *service layer* (SL) will be reduced. Congestion control at a CA port operates either at the QP or at the SL level, exclusively. To determine how much and for how long the injection rate should be reduced, the CA uses a *Congestion Control Table* (CCT) and a set of CC parameters. The CCT, consisting of at least 128 entries, holds *injection rate delay* (IRD) values that define the delay between consecutive packets sent by a particular flow (QP or SL). Each flow with CC activated holds an index into the CCT, the $CCTI$. When a new BECN arrives, the $CCTI$ of the flow is increased by $CCTI\_Increase$. The CCT is usually populated in such a way that a larger index yields a larger IRD. Then consecutive BECNs increase the IRD which again decreases the injection rate. The upper bound of the $CCTI$ is given by $CCTI\_Limit$.

To increase the injection rate again, the CA relies on a $CCTI\_Timer$, maintained separately for each SL of a port. Each time the timer expires the $CCTI$ is decremented by one for all flows associated with the corresponding port SL. When the $CCTI$ of a flow reaches zero, the flow now longer

[2]There are three exceptions. The FECN bit in a multicast packet, acknowledgement packet or congestion notification packet should be ignored. That is, no congestion notification is sent back to the source in these three cases.

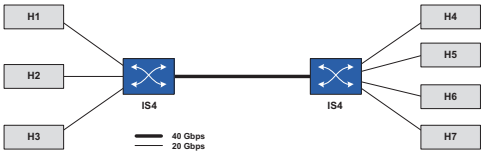


Figure 4. The test bed.

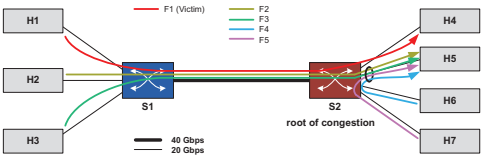


Figure 5. Flow configuration in scenario 1.

experience any IRD. Each port SL also has a $CCTI\_Min$ parameter. Using the $CCTI\_Min$ it is possible to impose a minimum IRD to the port SL, as the $CCTI$ should never be reduced below the $CCTI\_Min$.

III. The Test Bed

In this section we describe the hardware and software used in our test bed, shown in fig. 4.

A. The Mellanox Switches and Adapters

Mellanox ConnectX InfiniBand adapters and InfiniScale® IV switches (IS4 in fig. 4) are the latest generation of IB solutions from Mellanox Technologies that have been designed for HPC clustering technology. The ConnectX HCAs and InfiniScale IV switches deliver up to 40 Gbit/s of bandwidth between servers and up to 120 Gbit/s between switches. This is matched with ultra-low application latency of 1 μs, and switch latencies of 100 ns.

Mellanox ConnectX HCAs and InfiniScale IV switches both include support for the InfiniBand CC mechanism, and at the moment are the only end-to-end solutions to provide this capability[3]. Furthermore, the adapters and switches also include other critical capabilities for efficient high-performance computing networking, such as adaptive routing and application offload. Adaptive routing helps to eliminate network congestion due to point-to-point communications that share the same path, while application offload reduce the CPU overhead of networking processes. Adaptive routing is out of scope for this paper, where we focus on the IB CC capabilities and how it might eliminate congestion that occur due to multiple traffic initiators and a single target.

B. Compute Nodes

The compute nodes in our test bed consists of seven Sun Fire X2200 M2 servers that are connected as hosts H1-H7 in figure 4. Each host has a dual port Mellanox ConnectX DDR HCA fitted in a 8x PCIe 1.1 slot, one dual core AMD Opteron 2210 CPU, and 2GB of RAM. All hosts run Ubuntu Linux 8.04 x86_64 with kernel version 2.6.24-24-generic and OFED 1.4.1. The PCIe 1.1 8x slots in these machines has a signalling rate of 20 Gbit/s, which equals a theoretical bandwidth of 16 Gbit/s when counting for the 8b/10b encoding overhead. The achievable bandwidth is further reduced by PCIe protocol overhead, the speed of other system components etc.

To generate traffic on the hosts we use several different tools. *Netpipe* [17], which measures bandwidth and latency for different packet sizes, is used to get some basic performance numbers. To be able to study congestion in a controlled manner we have in addition implemented some changes to the *perftest*[18] application suite to support regular bandwidth reporting and continuously sending traffic at full capacity. The modified *perftest* is used to both create congestion in the network and to measure the impact of congestion. We also used the HPC Challenge [19] benchmark to study the impact of congestion on a few well know HPC applications.

IV. Experiment Scenarios

In this section we describe the two communication scenarios we have used to investigate the behaviour of CC in our test bed.

A. Scenario 1

The purpose of communication scenario 1 is twofold. First, it illustrates the negative effect that congestion has on a victim flow (flow 1 from *H1* to *H4* in fig. 5). Second, it illustrates how this can be avoided by using congestion control.

In this scenario we use the following communication pattern (fig. 5): Flow 1 (F1) from *H1* to *H4*, and flow 2 - 5 (F2-F5) where *H2*, *H3*, *H6*, and *H7* all send to *H5*. Communication starts with only F1 active, then F2 - F5 are activated one by one with one second intervals. When a flow is active it tries to send at maximum speed, using a reliable connection.

B. Scenario 2

The purpose of communication scenario 2 is to study how congestion control performs when there is no victim present, and by that no HOL blocking to reduce in order to potentially improve overall performance.

In this scenario we use the following communication pattern (fig. 6): Flow 1 (F1) from *H1* to *H4*, flow 2 (F2) from

[3]Custom firmware is required both for switches and HCAs to enable congestion control.

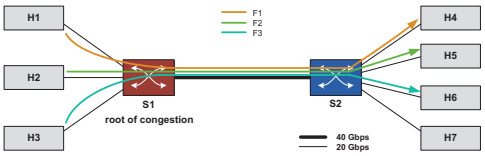


Figure 6. Flow configuration in scenario 2.

Parameter	Value
Threshold	15
Marking_Rate	1
Packet_Size	8
CCTI_Increase	1
CCTI_Limit	127
CCTI_Min	0
CCTI_Timer	150

Table I
CC PARAMETER VALUES FOR SCENARIO 1 AND 2.

H2 to *H5*, and flow 3 (F3) from *H3* to *H6*. Communication starts with only F1 active, then F2 and F3 are activated one by one with one second intervals. As before, when a flow is active, it tries to send at maximum speed, using a reliable connection. In this scenario there is no victim flow, but there is contention for bandwidth on the link between *S1* and *S2* that is shared by all three flows.

V. EXPERIMENT RESULTS

In this section we present and analyze the results obtained from our scenario 1 and scenario 2 experiments, staring with scenario 1. At the end we give a brief summary of our most important findings.

A. Results from Scenario 1

Figure 7(a) shows the individual throughput of the five traffic flows from scenario 1 (fig. 5) running without flow control. For the first 1.5 seconds the flow from H1 to H4 (*F1*) is the only active flow in the network. During this period the average throughput of this flow is 13 Gbit/s. This is as expected with our hardware configuration, where the throughput is limited by the PCIe capacity at the hosts[20], [21]. Then we progressively add one new flow each second until the four sources H2, H3, H6 and H7 are active (flows *F2-F5* in fig. 5, respectively), all with H5 as the destination.

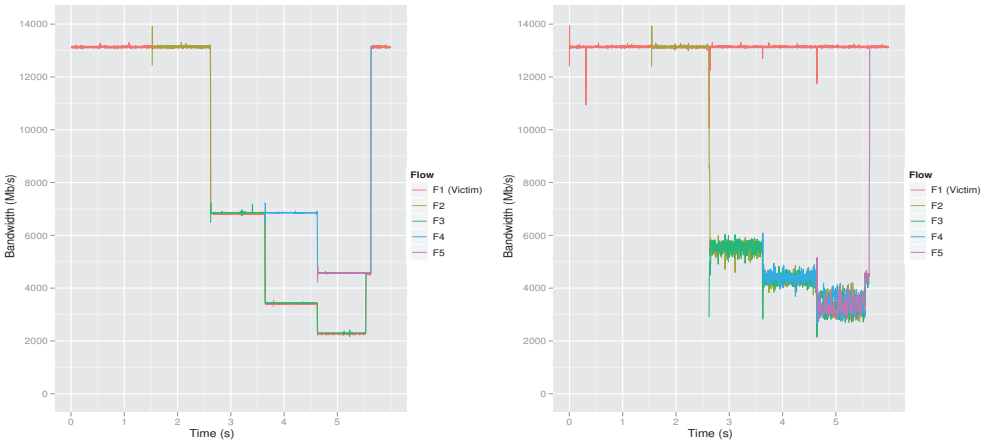
The addition of flow *F2* does not affect *F1* as the two flows only share the link between the two switches, a link with twice the capacity of the switch-to-host links. Therefore they both achieve a throughput of 13 Gbit/s. Now adding the flow *F3*, we observe a major drop in throughput for all three flows, leaving them at just below 7 Gbit/s each. This happens because the link from switch S2 to H5 has become a bottleneck, causing a congestion tree to be built from S2 towards the sources. Due to HOL blocking, *F1* becomes a *victim* flow which is also affected, even if that flow is not requesting the congested resources at S2. *F1* gets the same share of the switch-to-switch link as *F2* and *F3*, a share determined by the individual access *F2* and *F3* get to the bottleneck link. The growth of the congestion tree has led to an underutilization of the switch-to-switch link, wasting resources in the network.

Adding flow *F4* (blue flow in fig. 7), the flows *F1* (victim), *F2* and *F3* experience a new drop in performance, roughly halving their throughput once more. Now, *F1* suffers even more due to the HOL blocking. Notice that *F4* gets more than its fair share of the bottleneck link, achieving a throughput of almost 7 Gbit/s. This is an example of the well known *parking lot problem* [22], [23]. As the flows *F2* and *F3* (and *F1*) from S1 are all treated by S2 as one traffic flow, the flows *F2* and *F3* together only get access to the same amount of the congested resources as the flow *F4* does alone.

Adding the last flow, *F5*, the same pattern repeats. *F1* (victim), *F2* and *F3* is reduced to approximately 2 Gbit/s, while *F4* and *F5* is reduced to approximately 4.5 Gbit/s. Again, *F1* suffers even more from the HOL blocking, while we still see an unfairness among the flows headed for H5.

Figure 7(b) shows the scenario 1 experiment (fig. 5) repeated with CC enabled. As we can see from the figure, the CC mechanism is able to completely remove the HOL blocking of the victim flow *F1*, giving the flow a more or less constant throughput of 13 Gbit/s independent of the other traffic flows. CC is activated at switch S2 as soon as we add the flows *F3*, *F4* and *F5*. Then some oscillations occur among all the flows contributing to congestion as they are constantly adjusting their injection rates depending on the congestion notifications received at the sources. This oscillating behaviour corresponds well to the simulation results provided by [14]. When the flows *F2* and *F3* are the sole contributors to congestion, they both experience a small penalty in average throughput caused by the activation of the CC. This penalty is, however, removed with the introduction of the flows *F4* and *F5*. Both the degree of oscillation and the penalty in throughput caused by the activation of the CC, depend on the CC parameter values being used. We will explore the CC parameter space further in section VII. Table I shows the parameter values used for our scenario 1 and scenario 2 experiments.

An interesting observation is that the activation of CC to resolve congestion also solves the parking lot problem in our test scenario. As mentioned earlier, the switch S2 treats *F2* and *F3* as a single flow when providing access to the congested link. This gives the flows *F4* and *F5* an unwanted advantage. *F4* and *F5* both get access to 1/3 of the capacity of the link towards the node H5, while *F2* and *F3* have to



(a) Congestion Control turned OFF.

(b) Congestion Control turned ON.

Figure 7. Measured throughput for flows in scenario 1.

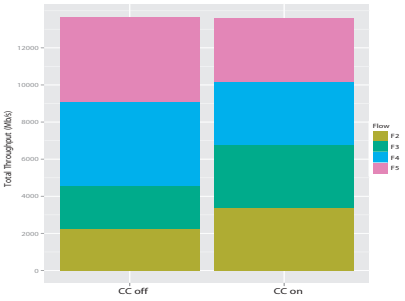


Figure 8. Fairness between the flows contribution to congestion.

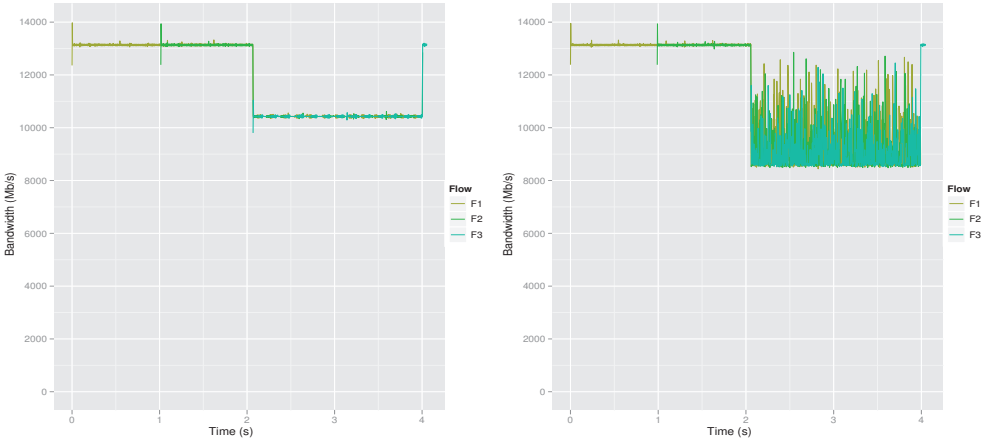
share the last 1/3 of the capacity. This skew is shown in the first bar in fig. 8. The CC treats all contributors to congestion in a fair way. If one contributor occupies more than its fair share of the congested resources at the root of the congestion tree, it will receive a correspondingly high share of the congestion notifications, and by that throttle the injection rate more than contributors occupying less resources at the root of the tree. For the four contributors to congestion in our experiment, the result is even access to the congested link, effectively solving the parking lot problem. Figure 8, second bar, illustrates this, showing how all four flows share the congested link equally.

1) Packet size: All results presented this far are gathered from experiments run with a packet size of 65536 bytes and a MTU of 2048 bytes. The potential benefit from CC is, however, by no means limited to this packet size. Figure 9 shows how a victim of HOL blocking, the *F1* flow from the last section, benefits from the CC, depending on the packet size being used. As we can see from the graph, activating CC results in an order of magnitude improvement in throughput for this flow, independent of the packet size. The throughput that the victim flow *F1* achieves with CC enabled, coincides with the throughput the same flow achieves when there is no congestion in the network, while a congested network without CC yields inferior results.

B. Results from Scenario 2

In scenario 2 (fig. 6) we turn our attention towards the possible penalty of enabling CC in a network. We do this by focusing on a scenario where only contributors to congestion are present. In particular we have removed the HOL blocked traffic flow that experienced a performance gain when we enabled CC in scenario 1. In addition we have moved the root of the congestion tree from S2 to S1, to maximize the length the congestion notifications have to travel in our test bed. Now the three sources H1, H2 and H3 sends traffic to the three destinations H4, H5 and H6, respectively. We denote the three flows *F1* , *F2* and *F3* (fig. 6)

Figure 10(a) shows the throughput of the three flows *F1* to *F3* when the CC is turned off. As in scenario 1, we progressively add one flow each second to see what impact each new flow has on the network performance. When the third flow, *F3*, is added after approximately 2 seconds, we observe that the link based flow control throttles the three sources. The switch-to-switch link has now reached



(a) Congestion Control turned OFF.

(b) Congestion Control turned ON.

Figure 10. Measured throughput for flows in scenario 2.

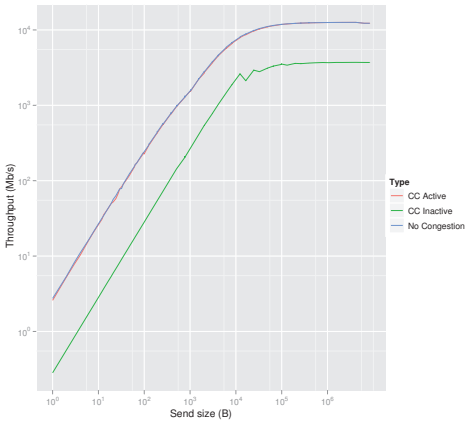


Figure 9. Throughput of the victim flow as a function of packet size.

its capacity (remember that this link has twice the bandwidth of the switch-to-host links). A congestion tree has grown towards the sources, constantly supplying the switch with traffic to forward. All three flows get their fair share of the bandwidth. This is as expected, given a fair switch.

If we enable the CC, the behavior in the network changes when we add the third traffic flow, *F3*, (fig 10(b)). Now CC is triggered at switch S1 as soon as the switch-to-switch link becomes congested. The CC introduces oscillation caused by the sources constantly trying to adjust their injection rate to resolve congestion. The behavior is the same as we observed for the contributors to congestion during scenario 1. The throughput jitter experienced by the three traffic flows are increased by more than an order of magnitude (table II). While the oscillation is clearly visible, the average throughput experienced by each of the three flows is, however, only reduced by 3.5%. All three flows are treated fairly.

It is evident that enabling the congestion control with the CC parameters from scenario 1 decreases throughput and increases oscillation, as seen in Figure 10 and table II. Notice though, that this is a worst case scenario with no possible benefit from enabling the CC.

(a) Congestion Control turned OFF.

	H1	**H2**	**H3**	**All**
Mean	10427.69	10427.82	10427.41	10427.64
SD	28.38508	42.83686	51.56237	42.02744
Min	10319.09	10339.89	9831.57	9831.57
Max	10503.43	10615.34	11076.06	11076.06

(b) Congestion Control turned ON.

	H1	**H2**	**H3**	**All**
Mean	10065.51	10124.61	9986.035	10058.55
SD	1773.68	1743.891	1793.207	1770.445
Min	8504.688	8487.411	8496.641	8487.411
Max	13210.3	13206.15	13902.36	13902.36

Table II
THROUGHPUT STATISTICS FOR FLOWS IN SCENARIO 2.

C. Results Summary

Our results so far has shown that congestion can have a negative impact on flows not contributing to congestion and that IB CC is able to remove the negative impact congestion has on a victim flow. Furthermore, we have seen that the penalty of using IB CC is low even in a worst case scenario where there is no victim present and by that no HOL blocking to reduce in order to potentially improve overall performance. Finally we have seen that IB CC have an unforeseen positive side effect that gives fairness to flows that would otherwise be treated unfairly due to the parking lot problem.

VI. The Impact of CC on the HPC Challenge Benchmark

In the previous section we presented results from measurements on two configurations where both the congested flows and the victim flow used a synthetic traffic pattern. In this section we present results from measurements where the victim flow is replaced with a victim flow running the *HPC Challenge* (HPCC) benchmark [19], [24], [25]. The congested flows are still synthetically generated, but when combined with the HPCC benchmark this resembles the network conditions that HPCC type applications would experience when a hot spot is present, but not created by the HPCC applications itself. E.g. a hot spot created by a combination of I/O traffic and one or more concurrently running applications. This configuration allows us to study how congestion impacts the type of traffic generated by the HPCC benchmark.

Table III shows the main results from the HPCC benchmark when performed *a*) without both congestion and congestion control; *b*) with congestion and without congestion control; *c*) with both congestion and congestion control.

For the *Randomly ordered ring* (ROR) test the observed latency is increased by 544.6%, from 2036 μs to 11088 μs, when comparing the non-congested and congested scenario. When activating CC in the congested scenario the observed latency is reduced by 81.3%, from 11088 μs to 2073 μs. The observed difference between the uncongested scenario and the scenario with congestion and CC active is less than 1.8%, showing that the CC is able to resolve the congestion effectively. The other latency tests show similar behaviour.

The observed throughput for the ROR test is reduced by 48.8%, from 684.667 MByte/s to 350.357 MByte/s, when congestion is present. When activating CC the observed throughput is increased again by 95.1%, from 350.357 MByte/s to 683.452 MByte/s, which is very close to the observed throughput without congestion. The same behaviour can be seen for the other bandwidth tests. These results from the latency and bandwidth tests are as expected and they correspond well with what we saw for the synthetic traffic in the previous section.

The remaining application benchmarks illustrates how the network performance affects the application performance. To what extent they are affected depends on how communication sensitive the application is. E.g. the *Linpack* test only see a 2.1% improvement in performance when congestion is present and CC is active, compared to the congested scenario without CC. On the other hand the more communication sensitive *PTrans* test see an improvement in performance by 76%. The *RandomAccess* and *FFT* tests show a 20.5% and 39.3% improvement in performance, respectively. Again, the observed performance in a congested scenario with CC enabled is very close to what we observe in the scenario without congestion.

These results clearly shows how applications are negatively affected by congestion and how IB CC can be used to reduce, and sometimes remove completely, the negative effect of congestion.

Host Channel Adapter		**Switch**
$CongestionControlTable$	(CCT)	$Threshold$
$CCTI$	(CCT index)	$Marking\_Rate$
$CCTI\_Increase$		$Packet\_Size$
$CCTI\_Limit$		$Victim\_Mask$
$CCTI\_Min$		
$CCTI\_Timer$		

Table IV
InfiniBand CC host and switch parameters.

VII. Exploring the Effect of CC Parameters

As explained in Section II, there are several CC parameters that can be configured at the switches and the hosts. While analysing our results in the previous sections, we briefly mentioned the CC parameters, postponing the discussion concerning what parameter values to use. Now, we turn our attention towards the CC parameter space itself. We explore the space through reasoning and experiments, adding insight into the impact the parameter space has on performance, and by that how effective the corresponding instance of the CC mechanism is. We have used the traffic pattern from Scenario 1 as the main basis for our studies, as that scenario contains both a victim flow and several contributors to congestion.

A. Switch CC Parameters

Table IV summarizes the parameters introduced in Section II. Starting with the switch, the four parameters are $threshold$, $Marking\_Rate$, $Packet\_Size$, and $Victim\_Mask$, where $threshold$ and $Marking\_Rate$ proved to be the most interesting. The $Victim\_Mask$ is only used to ensure proper congestion detection when a host

[4]Shows the percentage decrease for latency and percentage increase for throughput between column c) and b).

Network Lat. And BW	a) No cong.	b) Cong, CC off	c) Cong, CC on	Impr.[4]
Min Ping Pong Lat. (ms)	0.001132	0.001192	0.001172	1.7%
Avg Ping Pong Lat. (ms)	0.001678	0.012385	0.001729	86.0%
Max Ping Pong Lat. (ms)	0.001957	0.018001	0.002056	88.6%
Naturally Ordered Ring Lat. (ms)	0.002193	0.011396	0.002098	81.6%
Randomly Ordered Ring Lat. (ms)	0.002036	0.011088	0.002073	81.3%
Min Ping Pong BW (MB/s)	880.463	663.235927	876.049	32.1%
Avg Ping Pong BW (MB/s)	1354.021	733.159	1360.26	85.5%
Max Ping Pong BW (MB/s)	1590.559	879.125	1611.025	83.3%
Naturally Ordered Ring BW (MB/s)	742.469675	213.687109	743.769828	248.1%
Randomly Ordered Ring BW (MB/s)	684.66655	350.356751	683.451954	95.1%
Other HPCC Benchmarks	**a) No cong.**	**b) Cong, CC off**	**c) Cong, CC on**	**Impr.[4]**
PTRANS GB/s	0.755254	0.347585	0.611816	76.0%
HPLinpack 2.0 Gflops	1.819	1.79	1.827	2.1%
MPIRandomAccess Updates GUP/s	0.015118991	0.01195898	0.014409549	20.5%
MPIFFT Gflops/s	1.3768	0.982365	1.36891	39.3%

Table III
RESULTS FROM THE HPC CHALLENGE BENCHMARK.

is connected to a switch. Varying the $Packet\_Size$ had little impact, easily explained by the fact that the packet size of the traffic flows within most of our experiments do not vary. Given a constant packet size, it is only important to keep the $Packet\_Size$ below the size of the packets actually being sent to keep the CC working. Differentiated FECN marking depending on the packet size is not an issue when all packet sizes are the same.

The $threshold$ parameter indicates how aggressive a switch shall be when deciding if a packet is experiencing congestion. Its value affects how early a switch signals congestion to a source. How fast a source is able to react is, however, also affected by the distance the congestion notifications have to travel, first going to the destination, and then back to the source. On one hand, the $threshold$ needs to be aggressive enough to signal sources early while the switch still has room for any in flight packets headed for the contested resources at the switch. If the $threshold$ is not aggressive enough a congestion tree might grow. On the other hand, being too aggressive, the switch might tell the sources to cease sending packets too early, leaving the contested resources at the switch idle as the buffers are emptied. Furthermore, if the $threshold$ is too aggressive, the switch is more likely to detect congestion based on small, temporary peaks in traffic, causing an unneeded reduction of injection rates at the sources.

We have found that the CC works best with the $threshold$ at its maximum value for our test bed. It is good to signal congestion early when there is still buffer space available, as it prevents a congestion tree from forming. We expect this to be true for larger networks as well, as the path between the root of the congestion and the sources is then generally longer, which implies a longer reaction time in order to quench the sources. It is worth noting though, that we have a deterministic traffic pattern in our experiments, where an aggressive $threshold$ never leads to a wrong guess about congestion. The deterministic contributors to congestion make sure that a guess about congestion is always right, with a certain amount of upstream traffic on its way to the contested resource. A more dynamic traffic pattern than what we have studied so far might increase the impact of the $threshold$ parameter.

The $Marking\_Rate$ parameter dictates the mean number of packets eligible for FECN marking sent between packets actually being marked at the switch. The $Marking\_Rate$ acts as a filter on top of the $threshold$ with regards to the number of FECNs sent. How many BECNs a source receives when congestion occurs, and by that how much the injection rate is reduced, is therefore closely related to the value of the $Marking\_Rate$ parameter. Figure 11 shows how the average throughput of our *victim* flow depends on the $Marking\_Rate$ (and the $CCTI\_Timer$ of a channel adapter). The plot is from Scenario 1 with all five flows active. Even though the throughput for a given $Marking\_Rate$ is obviously not independent of the other parameter, the $CCTI\_Timer$, it is evident that keeping the $Marking\_Rate$ low generally yields the best throughput. Keeping it low becomes particularly important when using low values of the $CCTI\_Timer$. We will get back to the correlation between the $Marking\_Rate$ and the $CCTI\_Timer$ in the next section, where we will discuss fig. 11 in more detail.

B. Channel Adaptor CC Parameters

The CC at a CA is centred around the *Congestion Control Table* (CCT) and the set of related parameters given in table IV. As explained in Section II, the CCT contains an array of increasing IRD values used to control the injection rate of a host, and thereby its contribution to congestion. A low IRD means a high injection rate and vice versa. The CCT size is given by the value $CCTI\_Limit$ and has a minimum value of 128. The $CCTI\_Limit$ serves as a upper bound for the $CCTI\_Index$ parameter. The

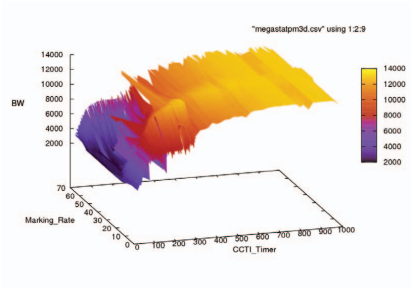


Figure 11. Average throughput of the *victim* flow as a function of the $CCTI\_Timer$ and the $Marking\_Rate$.

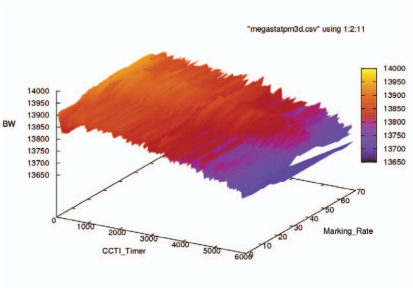


Figure 12. Average throughput of the four contributors to congestion as a function of $CCTI\_Timer$ and $Marking\_Rate$.

$CCTI\_Index$ refers to a given entry in the CCT for a given flow (QP or SL), and is used to select an IRD whenever a host should increase or decrease its injection rate for this particular flow. The $CCTI\_Index$ is increased by $CCTI\_Increase$ steps whenever a BECN is received, and decreased by one whenever the $CCTI\_Timer$ expires. The lower bound for the $CCTI\_Index$ is given by $CCTI\_Min$. To summarize, the CAs reaction to congestion depends on the size and the population of the CCT, and how the $CCTI\_Index$ moves inside this table. The movement of the $CCTI\_Index$ is mainly determined by the $CCTI\_Increase$, the $CCTI\_Timer$, and the $Marking\_Rate$ in the switch (discussed in the previous section).

A large CCT implies more IRD values and makes it possible to increase or decrease the injection rate in smaller steps than in a smaller table. In our test bed, the minimum size of 128 entries proved to give a granularity good enough to ensure efficient CC[5]. We used the values $CCTI\_Min = 0$ and $CCTI\_Limit = 127$ to utilize the whole CCT. The table was then populated by the formula $cct[i] = i^2 * 7/106^2$ (μs). This formula is a small adjustment of the default formula provided by the switch manufacturer ($cct[i] = i^2 * 7/95^2$). The adjustment of the IRDs was shown through experiments to give a little less oscillation in our test scenarios, while keeping the same average throughput. The exact impact of the IRD values might be different though in a larger network with a more dynamic traffic pattern, and needs to be further investigated in such environments.

How the $CCTI\_Index$ moves in the CCT, and by that how much traffic a contributor to congestion injects into the network, is as mentioned mainly determined by the $CCTI\_Increase$, the $CCTI\_Timer$, and the $Marking\_Rate$. Looking at fig. 11, we see the average throughput of the *victim* flow shown as a function of the $CCTI\_Timer$ and the $Marking\_Rate$, while all five flows are active. The $CCTI\_Increase$ is kept at the default value 1. Then, if the $CCTI\_Timer$ is too low, here below approximately 150 μs, the contributors to congestion increase the injection rate too early after receiving a BECN, effectively allowing the congestion tree to form, no matter the value of the $Marking\_Rate$. The *victim* suffers due to HOL blocking. The corresponding low throughput of the *victim* flow can be seen as the purple area of the surface in fig. 11. When the $CCTI\_Timer$ increases, the contributors keep a low injection rate for a longer period of time, removing the congestion tree and the corresponding HOL blocking. It is, however, important to keep the $Marking\_Rate$ low to supply the contributors with a high frequency of BECNs. As the $CCTI\_Timer$ increases, the throughput of the *victim* becomes less sensitive to the $Marking\_Rate$. The contributors decrease the injection rate for a longer period of time when throttled, and are able to remove the HOL blocking even if they receive less BECNs. The *victim* suffers from less HOL blocking in the area where the surface is first turning orange, and later yellow. In the bright yellow area the throughput of the *victim* is limited by the PCIe bus capacity of the hosts.

One could suspect the CC to be too aggressive in the yellow area of the surface in fig 11, underutilizing the contested resources in the network. Figure 12 shows, however, that the average throughput of the four contributors to congestion vary very little in this area (the surface has been rotated to increase readability). The four contributors are able to utilize the congested link, even when using a $CCTI\_Timer$ value as high as 2000 μs. This surface does, however, hide an important aspect of the CC mechanism; how fast the contributors are able to settle for a fair distribution of the

[5]Others has found that the minimum size is sufficient for larger topologies as well [14].

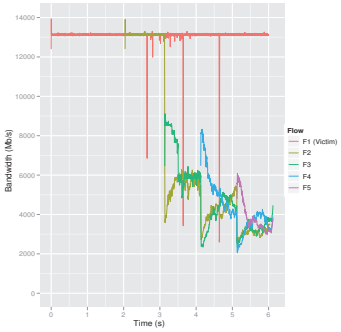


Figure 13. The scenario 1 experiment with a $CCTI\_Timer$ value of 2000 μs.

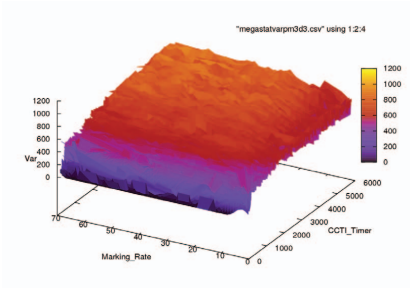


Figure 14. The *treatment variation variable* Var as a function of $CCTI\_Timer$ and $Marking\_Rate$.

contested resource at the root of the congestion tree when congestion occurs.

Figure 13 shows our scenario 1 experiment repeated with the $CCTI\_Timer$ set to 2000 μs. Comparing this figure to fig. 7(b), we clearly see how the contributors now experience unfairness among each other for an extended period of time, each time a new contributor is added. The CC mechanism is not able to stabilize the contributors, with regard to fairness between these flows, during the one second interval between adding new flows. To further investigate how fast the contributors stabilize we need to study the treatment of the contributors during congestion. We do this by defining a *treatment variation variable* (Var) as a function of the $CCTI\_Timer$ and the $Marking\_Rate$ parameters. For each point in time where all four contributors are active in scenario 1, we subtract the lowest throughput of any of the four flows from the throughput of the flow with the highest throughput. This results in an array of delta throughput values for the time period where all four flows are active. Then, calculating the variation of this delta array, we get the Var value indicating how fast the CC mechanism is able stabilize and give the four flows a fair treatment when congestion occurs. In fig. 14 Var is plotted as a function of the $CCTI\_Timer$ and the $Marking\_Rate$. Now we clearly see how a large part of the parameter space, the orange part of the surface, will result in unfairness and instability among the contributors. With regard to fairness, the CC mechanism performs best when the $CCTI\_Timer$ is kept low.

Based on our experiment results and the insight from fig. 11, fig. 12 and fig. 14, we have observed that we achieve the best performance of the *victim* flow when both the $threshold$ and the $CCTI\_Timer$ is high, while the $Marking\_Rate$ has limited impact on performance. The situation is opposite for the flows contributing to congestion. Here the best performance is achived when the $CCTI\_Timer$ is kept low. Moreover, we see that the set of parameter values that gives good performance is small. Based on these observations we were able to narrow down the CC parameter space to the values given in table I, used during our scenario 1 and scenario 2 experiments.

VIII. Conclusions

Congestion control has been an important subject for researchers in interconnection networks for many years. Still, hardware implementation of CC mechanisms have only recently materialized. To the best of our knowledge, this paper contains the first results on the behavior of congestion control mechanisms in InfiniBand implemented in hardware.

Our main findings are the following:

- Without CC, the problems of flows being victims of congestion without contributing to it is easy to provoke. This problem is severe, as it makes the bandwidth of links in the congestion tree lie idle, even if this bandwidth is needed by a victim flow.
- The parking lot problem, where flows get uneven shares of a congested link, is severe when CC is not active.
- Infiniband CC can alleviate both above problems. Our results show that Infiniband CC can equally save the victims of congestion, and give fairness to flows that share a congested link.
- The cost of having Infiniband CC turned on can be made small. In a scenario where there is congestion, but no victims to save, the parameters can be set so that there is a negligible penalty in throughput.
- The above results of Infiniband CC on synthetic flows translate into highly significant improvements in the performance of the more realistic traffic scenarioes of the HPC Challenge benchmark.
- Even if the performance of Infiniband CC is sensitive

to parameter setting, we were able to find a sweet spot for our test scenarios.

This first study of Infiniband CC in hardware has given encouraging results. Still, there is further work to be done before the mechanism is fully understood. Open questions include how these results scale to bigger topologies, and to what extent optimal tuning of the parameter setting is stable over varying topologies, traffic patterns and applications. These questions will be addressed in further work, that will include the combination of hardware measurements in limited topologies, and calibrated simulation tests in large topologies.

ACKNOWLEDGEMENTS

We would like to thank the HPC Advisory Council [26] for the support throughout the research and testing activities and for the contribution of the switches that were used for the testing. The HPC Advisory Council is a world-wide HPC organization that support education, outreach and research activities on high performance computing solutions.

REFERENCES

[1] G. F. Pfister and V. A. Norton, ""hot spot" contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. 34, no. 10, pp. 943–948, 1985.

[2] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[3] P. García, J. Flich, J. Duato, I. Johnson, F. Quiles, and F.Naven, "Dynamic evolution of congestion trees: Analysis and impact on switch architecture," in *High Performance Embedded Architectures and Compilers*, 2005, pp. 266–285.

[4] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*. ACM, 1988, pp. 314–329.

[5] L. S. Brakmo and L. L. Peterson, "Tcp vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465–1480, 1995.

[6] C. Parsa and J. Garcia-Luna-Aceves, "Improving tcp congestion control over internets with heterogeneous transmission media," in *7th International Conferance on Network Protocols (ICNP99)*. IEEE Computer Society, 1999, pp. 213–221.

[7] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-end congestion control for infiniband," in *INFOCOM*, 2003.

[8] *Infiniband architecture specification*, 1st ed., InfiniBand Trade Association, November 2007.

[9] *Data Center Bridging standard*, Ieee 802.1qau ed., IEEE 802 LAN/MAN Standards Committee. [Online]. Available: http://www.ieee802.org/1/

[10] J. R. Santos, Y. Turner, and G. J. Janakiraman, "Evaluation of congestion detection mechanisms for infiniband switches," in *IEEE GLOBECOM – High-Speed Networks Symposium*, 2002.

[11] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Congestion management in mins through marked and validated packets," in *PDP*, 2007, pp. 254–261.

[12] ——, "On the influence of the packet marking and injection control schemes in congestion management for mins," in *Euro-Par*, 2008, pp. 930–939.

[13] "Top 500 supercomputer sites," http://top500.org/, June 2009.

[14] G. Pfister, M. Guzat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using infiniband architecture congestion control," Invited paper in High Performance Interconnects for Distributed Computing, july 2005. [Online]. Available: http://www.cercs.gatech.edu/hpidc2005/presentations/GregPfister.pdf

[15] M. Technologies, "Mellanox infiniscale iv switch architecture provides massively scaleable 40gb/s server and storage connectivity," Press release, November 2007, http://www.mellanox.com/content/pages.php?pg=press_release_item&rec_id=34.

[16] ——, "Mellanox announces availability of industry's first 40gb/s infiniband switch silicon device and product reference platforms," Press release, June 2008, http://www.mellanox.com/content/pages.php?pg=press_release_item&rec_id=214.

[17] "Netpipe - network protocol independent performance evaluator," http://www.scl.ameslab.gov/netpipe/, September 2009.

[18] "Perftest - performance testing framework," http://perftest.sourceforge.net/, September 2009.

[19] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the hpc challenge benchmark suite," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-57493, april 2005, http://repositories.cdlib.org/lbl/LBNL-57493.

[20] J. Liu, A. Mamidala, A. Vishnu, and D. K. Panda, "Evaluating infiniband performance with pci express," *IEEE Micro*, vol. 25, no. 1, pp. 20–29, 2005.

[21] M. J. Koop, W. Huang, K. Gopalakrishnan, and D. K. Panda, "Performance analysis and evaluation of pcie 2.0 and quad-data rate infiniband," *High-Performance Interconnects, Symposium on*, vol. 0, pp. 85–92, 2008.

[22] M. Galles, "Spider: A high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, pp. 34–39, 1997.

[23] W. J. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004, ch. 15.4.1, pp. 294–295.

[24] "Hpc challenge benchmark," http://icl.cs.utk.edu/hpcc/.

[25] R. Rabenseifner, S. Tiyyagura, and M. Muller, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer-Verlag, october 2005, vol. 3666, ch. Network Bandwidth Measurements and Ratio Analysis with the HPC Challenge Benchmark Suite (HPCC), pp. 368–378.

[26] "Hpc advisory council," http://www.hpcadvisorycouncil.com/.

Paper IV

Exploring the Scope of the InfiniBand Congestion Control Mechanism

Ernst Gunnar Gran, Sven-Arne Reinemo, Olav Lysne,
Tor Skeie, Eitan Zahavi, and Gilad Shainer

2012 IEEE 26th International Parallel and Distributed Processing Symposium

Exploring the Scope of the InfiniBand Congestion Control Mechanism

Ernst Gunnar Gran, Sven-Arne Reinemo,
Olav Lysne, Tor Skeie
Simula Research Laboratory
Fornebu, Norway
Email: {ernstgr, svenar, olavly, tskeie}@simula.no

Eitan Zahavi, Gilad Shainer
Mellanox Technologies
Israel/USA
Email: eitan@mellanox.co.il,
Shainer@Mellanox.com

***Abstract*—In a lossless interconnection network, network congestion needs to be detected and resolved to ensure high performance and good utilization of network resources at high network load. If no countermeasure is taken, congestion at a node in the network will stimulate the growth of a congestion tree that not only affects contributors to congestion, but also other traffic flows in the network. Left untouched, the congestion tree will block traffic flows, lead to underutilization of network resources and result in a severe drop in network performance.**

The InfiniBand standard specifies a congestion control (CC) mechanism to detect and resolve congestion before a congestion tree is able to grow and, by that, hamper the network performance. The InfiniBand CC mechanism includes a rich set of parameters that can be tuned in order to achieve effective CC. Even though it has been shown that the CC mechanism, properly tuned, is able to improve both throughput and fairness in an interconnection network, it has been questioned whether the mechanism is fast enough to keep up with dynamic network traffic, and if a given set of parameter values for a topology is robust when it comes to different traffic patterns, or if the parameters need to be tuned depending on the applications in use. In this paper we address both these questions. Using the three-stage fat-tree topology from the Sun Datacenter InfiniBand Switch 648 as a basis, and a simulator tuned against CC capable InfiniBand hardware, we conduct a systematic study of the efficiency of the InfiniBand CC mechanism as the network traffic becomes increasingly more dynamic.

Our studies show that the InfiniBand CC, even when using a single set of parameter values, performs very well as the traffic patterns becomes increasingly more dynamic, outperforming a network without CC in all cases. Our results show throughput increases varying from a few percent, to a seventeen-fold increase.

I. Introduction

Traffic congestion in interconnection networks may degrade the network and the compute system performance severely if no countermeasures are taken [1], [2], [3]. Congestion is simply a result of high load of traffic fed into a network link, exceeding the link capacity at that point. Hot spot traffic patterns, network burstiness, re-routing around faulty regions, and conducting link frequency/voltage scaling (lowering the link speed in order to save power), can all lead to congestion. If all these factors are known in advance, the network administrator may alleviate the consequences by effective load balancing of the traffic, but typically this is not the case. Furthermore, in cases where multiple nodes send more data to a single destination than the node can handle, no dynamic re-routing can be done to avoid network congestion. It becomes even more severe when a parallel computer is running multiple different jobs as an on-demand service (e.g. cloud computing), where the resulting traffic pattern becomes unpredictable.

Congestion control (CC) as a countermeasure for relieving the consequences of congestion has been widely studied in the literature. In particular, this problem is well understood and solved by dropping network packets in traditional lossy networks such as local area networks (LANs) and wide area networks (WANs). In these environments packet loss and increased latency are indications of network congestion. Herein it is mainly TCP that implements end-to-end congestion control, either by a traditional window control mechanism [4] for detecting dropped packets or through changes in latency [5], [6]. Very often those networks are also over-provisioned in order to avoid congestion.

In high performance computing (HPC) data centers low latency is crucial, and packet dropping and retransmission are not allowed under regular circumstances, contrary to LANs and WANs, due to the loss of performance that is associated with packet drops. Lossless behavior is achieved with credit based link-level flow control, which prevents a node or a switch from transmitting packets if the downstream node or switch lacks buffer space to receive them.

Typically, when congestion occurs in a switch in a network with link-level flow control, a congestion tree starts to build up due to the backpressure effect of the flow control mechanism. The switch where the congestion starts will be the root of a congestion tree that grows towards the source nodes contributing to the congestion. This effect is known as congestion spreading. The tree grows because buffers fill up through the switches as the switches run out of flow control credits (not necessarily in the root). As the congestion tree grows, it introduces head-of-line (HOL) blocking [7] and slows down packet forwarding that also affects flows which are not contributing to the congestion, severely degrading the entire network performance. The HOL blocked flows become victims of congestion [7].

Congestion control for link-level flow controlled networks

1530-2075/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPS.2012.104

cannot be based on a traditional window control mechanism as deployed by TCP, even though it effectively limits the amount of buffer space that a flow can occupy in the network [8]. The reason for this is the relatively small bandwidth-delay product in this environment, where even a small window size may saturate the network [7]. A rate control based CC mechanism is more appropriate for link-level flow controlled networks, since it increases the range of control compared to a window based system. The mechanism relies on the switches to detect congestion, and inform the sources that contribute to the congestion that they must reduce their corresponding injection rates. There are basically two ways to inform the source nodes in such an explicit congestion notification scheme. Either the switches can mark the packets contributing to congestion in order to notify the destinations about the situation which subsequently notifies the sources (the forward explicit notification approach), or the switches can themselves generate notification packets that are sent directly to the source nodes (the backward explicit notification approach). The InfiniBand [9] architecture applies the former approach[1], while the emerging Data Center Bridging standard [10] is implementing the latter.

There is a body of work that propose different strategies for congestion notification and marking, e.g. a congested packet can be marked both in the input and output buffer as well as being tagged with information about the severity of the congestion. Furthermore, there are several different approaches to the design of the source response function, i.e. the actions taken to reduce the injection rate, later followed by an increase in the rate when congestion is resolved [8], [11], [12], [13].

There are also congestion control mechanisms targeting link-level flow controlled networks that take a completely different approach. Instead of removing the congestion tree itself, these approaches strive to relieve the unfortunate side effects the congestion tree has on flows not contributing to the congestion. That is, they try to remove the HOL blocking by using special set aside queues for contributors to congestion, effectively making it possible for victim flows to bypass the contributors to congestion without actually removing the congestion tree [14], [15]. Such an approach has the advantage of being able to react immediately and locally at each switch, at the cost of the extra buffers needed for the set aside queues and the added complexity in the switch to manage them. The real cause of the problem, sources injecting too much traffic into the network, is left untouched.

Adaptive routing (AR) could be used as a mechanism to alleviate congestion by spreading the traffic in the network onto otherwise idle resources when congestion occurs. Notice, though, that AR as a sole solution to congestion might not be effective. When there is no possible route around an area of congestion (e.g. end node congestion), trying to reroute around the problem will only make the branches of the congestion tree spread out and cause more HOL blocking than when using deterministic routing. To be effective AR needs to work in conjunction with another CC mechanism. Furthermore, the InfiniBand specification does not yet support AR.

InfiniBand (IB) was standardized in October 2000 and over the years it has increased its market share, when referring to the Top500 list [16], to 42% of the HPC market. If we only look at the top 100 supercomputers in the world, the number increases to 62%, while 5 out of 10 Petaflop systems in the world are using IB as the system interconnect. The vast majority of the IB installations are fat-trees with the most notable exceptions being Pleiades (11D hypercube) and RedSky (3D torus).

In 2010 Gran et. al. presented the first experiences with CC in IB hardware, where they showed that the IB CC mechanism effectively resolves congestion and improves fairness by solving the parking lot problem, if the CC parameters are appropriately set [7] and the switch arbitration properly designed [17]. Another significant contribution is the work done by Pfister et. al. [18], where they studied (through simulations) how well IB CC can solve certain hot spot traffic scenarios in fat-tree networks.

The IB standard provides a large degree of freedom, but little guidance, when it comes to configuring the CC mechanism. Care must be taken when configuring the CC parameters because a bad configuration can result in low performance and instability in the network [7].

In this paper we explore the scope of the IB CC mechanism with regards to the robustness and performance of a given parameter set when the communication pattern gradually changes from a static to a dynamic scenario. We describe a set of communication patterns designed to stress the IB CC mechanism and we present a large set of simulation results showing that the IB CC mechanism is both robust and increases performance in all the scenarios studied independent of the communication pattern.

The reminder of the paper is organized as follows: Section II gives an overview of the CC mechanism supported by IB. In section III we describe different types of congestion trees and the communication patterns used in our simulations.Then the simulation model is described in section IV, before we in section V discuss the simulation results. Finally, we conclude in section VI.

II. Congestion Control in InfiniBand

The IB CC mechanism, specified in the InfiniBand Architecture Specification release 1.2.1 [9], is based on a closed loop feedback control systems where a switch detecting congestion marks packets contributing to the congestion by

[1]Congestion control was added in release 1.2 of the InfiniBand specification and is to some extent based on the work done by Santos et. al. [8].

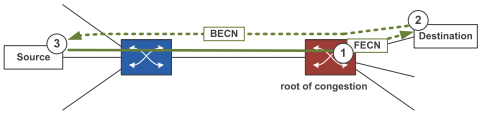


Figure 1. Congestion control in InfiniBand.

setting a specific bit in the packet headers, the *Forward Explicit Congestion Notification* (FECN) bit (fig. 1 (1)). The congestion notification is carried through to the destination by this bit. The destination registers the FECN bit, and returns a packet with the *Backward Explicit Congestion Notification* (BECN) bit set to the source (fig. 1 (2)). The source then temporarily reduces the injection rate to resolve congestion (fig. 1 (3)).

The exact behaviour of the IB CC mechanism depends upon the values of a set of CC parameters governed by a *Congestion Control Manager*. These parameters determine characteristics like when switches detect congestion, at what rate the switches will notify destination nodes using the FECN bit, and how much and for how long a source node contributing to congestion will reduce its injection rate. Appropriately set, these parameters should enable the network to resolve congestion, avoiding HOL blocking, while still utilizing the network resources efficiently.

1) Switch Features: The switches are responsible for detecting congestion and notifying the destination nodes using the FECN bit. A switch detects congestion on a given *port* and a given *Virtual Lane* (Port VL) depending on a $threshold$ parameter. If the threshold is crossed, a port may enter the Port VL congestion state, which again may lead to FECN marking of packets.

The $threshold$, represented by a weight ranging from 0 to 15 in value, is the same for all VLs on a given port, but could be set to a different level for each port. A weight of 0 indicates that no packets should be marked, while the values 1 through 15 represent a uniformly decreasing value of the threshold. That is, a value of 1 indicates a high threshold with high possibility of congestion spreading, caused by Port VLs moving into the congestion state too late. A value of 15 on the other hand indicates a low threshold with a corresponding low possibility of congestion spreading, but at the cost of a higher probability for a Port VL to move into the congestion state even when the switch is not really congested. The exact implementation of the threshold depends on the switch architecture and is left to the designer of the switch.

A Port VL may enter the congestion state if the threshold is crossed and it is the root of congestion, i.e. the Port VL has available credits to output data. If the Port VL has no available credits, it is considered to be a victim of congestion and shall not enter the congestion state[2]. When a Port VL is in the congestion state its packets are eligible for FECN marking. A packet will then get the FECN bit set depending on two CC parameters at the switch, the $Packet\_Size$ and the $Marking\_Rate$. Packets with a size smaller than the $Packet\_Size$ will not get the FECN bit set. The $Marking\_Rate$ sets the mean number of eligible packets sent between packets actually being marked. With both the $Packet\_Size$ and the $Marking\_Rate$ set to 0, all packets should get the FECN bit set while a Port VL is in the congestion state.

2) Channel Adapter Features: When a destination CA receives a packet with a FECN bit set, the CA should as quickly as possible notify the source of the packet about the congestion. This is done by returning a packet with the BECN bit set back to the source. The packet with the BECN bit could either be an acknowledgement packet (ACK) for a reliable connection or an explicit *congestion notification packet* (CNP). In either case it is important that the ACK or the CNP is sent to the source as soon as possible to ensure a fast response to the congestion.

When a source CA receives a packet with the BECN bit set, the CA lowers the injection rate of the corresponding traffic flow. To determine how much and for how long the injection rate should be reduced, the CA uses a *Congestion Control Table* (CCT) and a set of CC parameters. The CCT holds *injection rate delay* (IRD) values that define the delay between consecutive packets sent by a particular flow (the IRD calculation being relative to the packet length). Each flow with CC activated holds an index into the CCT, the $CCTI$. When a new BECN arrives, the $CCTI$ of the flow is increased by $CCTI\_Increase$. The CCT is usually populated in such a way that a larger index yields a larger IRD. Then consecutive BECNs increase the IRD which again decreases the injection rate. The upper bound of the $CCTI$ is given by $CCTI\_Limit$.

To increase the injection rate again, the CA relies on a $CCTI\_Timer$, maintained separately for each SL of a port. Each time the timer expires, the $CCTI$ is decremented by one for all associated flows. When the $CCTI$ of a flow reaches zero, the flow no longer experience any IRD.

The IB CC can operate either at the Service Level (SL) or at the Queue Pair (QP) level at an HCA. Any lowering of the injection rate as a result of BECN reception, then affects the whole SL or the single QP depending on the level of CC operation. Choosing the SL level will have a negative impact on both fairness and performance. The reason is that a single traffic flow contributing to congestion will lower the injection rate of all traffic flows within the same SL at the HCA. This could include traffic flows not contributing

[2]If the $Victim\_Mask$ is set for the port, then the switch will move the Port VL into the congestion state independently of the number of available credits. The $Victim\_Mask$ is typically set for switch ports connection HCAs to the switch as an HCA will never detect congestion itself.

to the congestion at all as they are not going through the root of the congestion tree, but headed for other parts of the network. In this paper we only consider CC operating at the QP level.

III. Congestion Trees

When contention leads to congestion in a lossless interconnection network, a congestion tree will form, as explained in section I. Branches of the tree will grow from the root of the tree, backwards towards the sources contributing to congestion, as traffic competing for the contested resources at the root starts to pile up. Buffers and links will be occupied along each branch by traffic headed for the root of the tree. Eventually, a branch might grow all the way back into a source node. The HOL blocking caused by the branches not only results in underutilization of network resources, but will also stimulate further growth of the tree towards nodes initially being victims of congestion. The role of a throttling based congestion control mechanism like IB CC, is to prune the branches of the congestion tree just enough to remove any HOL blocking, while still utilizing the bottleneck resources at the root of the tree.

Exactly how all the branches of a congestion tree grow and get pruned, and by that how the congestion tree itself develops and how much HOL blocking it causes, depends not only on the actions taken by the congestion control mechanism, but obviously also to a great extend on the traffic patterns in the network. A highly dynamic traffic pattern with lively contributors to congestion could lead to a vastly dynamic congestion tree, which again could pose a great challenge to the congestion control mechanism, compared to a more stable tree having a permanent set of contributors. At the same time, a dynamic traffic pattern could by its own dynamic nature, relieve the effects of congestion as the traffic pattern itself will hinder the formation of large congestion trees and any extensive HOL blocking. To be able to conduct a systematic study of the effectiveness of the IB CC mechanism under different traffic scenarios, we start by dividing congestion trees into three nonexclusive categories, based on the nature of their (main) contributors and how dynamic they are. Starting with the least dynamic category, the congestion trees can be *silent*, *windy*, or even *moving* while wind is blowing through their crowns.

A. The Silent Forest of Congestion Trees

The most basic congestion tree forms when a subset, C, of the nodes, N, in the network constantly injects traffic to a permanent hotspot, while the rest of the nodes, V, injects traffic to other destinations[3]. Then we have $N = C \cup V$, where C are the contributors to congestion, while V are the potential victims suffering from HOL blocking. In such a situation, a stable congestion tree will grow branches from the root of the congestion, ultimately the last link towards the hotspot, along the backward paths towards each of the nodes in C. In an IB network with CC enabled, the branches in such a congestion tree will grow and get pruned as the throttling mechanism constantly tries to adjust the injection rates of the nodes in C to their fair share of the resources at the root of the tree. Notice, however, that the branches will not move. The growing and shortening of a branch always happens along the same path. We refer to a congestion tree with such branches as a *silent* tree, as the tree's branches are *not blowing in the wind*. Figure 2 shows an artificial network topology where a silent congestion tree, represented by the red lines, has grown all the way from the root of the tree to the five source nodes contributing to congestion. We have somewhat artificially gathered all the source nodes in the network at the top, and the destination nodes at the bottom, to make the congestion tree visually more appealing and the characteristics of the tree easier to comprehend.

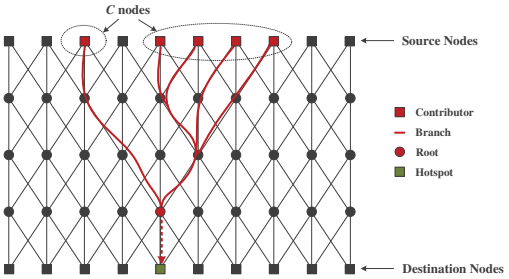


Figure 2. A silent congestion tree.

If C is divided into subsets $C_1, C_2, ..., C_n$ where each subset sends to a different hotspot, the corresponding network will grow a forest of silent congestion trees. Such a forest may resemble a set of nodes sending large virtual image files back to file servers in a network, or a set of sensor nodes continuously transferring large amounts of data back to a set of collector nodes.

The task of the IB CC mechanism will, as always, be to prune the branches of the trees all the way down to their roots, without causing under utilisation of the bottleneck resources. As the number of contributors is constant and the hotspots are stable, the IB CC should have a relatively easy task, given enough time, to adjust the injection rate of each contributor to its fair share of the bottleneck resources. The throttling mechanism at a source node does not need to keep track of different injection rates related to different destinations or traffic flows, and the recovery process after a period of congestion has ended is not really challenged as the hotspots are always present.

[3]Notice that in general, a node can be a contributor to one congestion tree and a victim of another - at the same time - depending on the communication patterns in the network.

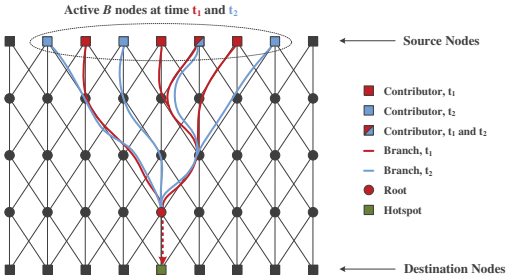


Figure 3. A windy congestion tree.

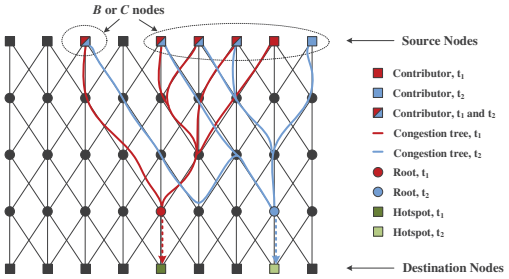


Figure 4. A moving congestion tree.

B. The Windy Forest of Congestion Trees

Now consider a network where the nodes are not divided into pure contributors to and possible victims of congestion, like in the previous section, but instead consists of a different type of nodes, B, that sends a certain percentage p of its traffic to a hotspot, and then the rest $1 - p$ percent to other nodes in the network. A B node can then be both a contributor to and a victim of congestion, depending on the time period we are looking at. In a network of B nodes, the traffic creates a congestion tree, given that p is not too low, but it will be different from a silent one. The root of the congestion tree will as before be permanently related to the hotspot, but the branches will now be more dynamic. No longer will the branches only grow and shrink along a specific set of paths, but the paths themselves change as the nodes actually sending to the hotspot at any given time changes. The branches of the congestion tree will be moving like the branches of a tree blowing in the wind. We have created a *windy* congestion tree. Figure 3 shows a network topology where seven B nodes create such a tree. First, at time t_1, four nodes create a set of branches for the congestion tree, the red lines, while later, at time t_2, a different set of nodes create a different set of branches, the blue lines. The branches have moved, while the root is the same. Notice that even though we in figure 3 show the branches as having grown all the way back to the source nodes, this will most likely not be the case in most situations. The branches will usually have different lengths, depending on the traffic pattern and the value of p.

Again, if B is divided into subsets $B_1, B_2, ..., B_n$ where each subset sends p_i percent to its own hotspot, the corresponding network will grow a forest of windy congestion trees. Such a traffic scenario could bear a resemblance to a set of compute nodes that communicate and exchange data with their peers, while at the same time store data at a set of storage nodes, i.e. the hotspots. The ratio between the peer communication and the storage usage is then given by the value of p.

The windy forest of congestion trees poses a new challenge for the IB CC mechanism. The branches of the congestion trees need to be pruned as before, but now the number of contributors are varying depending on p and what nodes that happen to send to the hotspot at a given time. In addition, it becomes important for the IB CC mechanism to separate the injection rate of different flows, to make sure that traffic not headed for the hotspot is not held back by the throttling mechanism. It becomes important that the IB CC operates at the QP level (or at least at the source-destination-pair level), and not the SL/VL level. The recovery process after a period of congestion has ended is, however, still not really challenged as the hotspots are always the same.

C. The Forest of Moving Congestion Trees

Both the silent and the windy congestion trees have a root related to a permanent hotspot. Now let us assume that the contributors to congestion sending to the hotspot H_i, no matter if they are contributing to a silent or windy congestion tree, change their target destination to a new hotspot, hotspot H_j. Independent of the congestion control mechanism, the congestion tree caused by the traffic headed for H_i will now have its root moved towards a new root location related to the new hotspot H_j, where new branches will grow. The congestion tree has *moved*[4]. Figure 4 illustrates a moving congestion tree before and after a set of contributors have changed focus from one hotspot at time t_1, to another at t_2. The red lines correspond to the congestion tree at t_1, while the blue lines correspond to the congestion tree at t_2. Again, the branches are for clarity shown as having grown all the way back to the source nodes.

A forest of moving congestion trees has a more dynamic nature than a forest of silent or windy congestion trees, no

[4]Strictly speaking, what is happening is that one tree disappears as a new one is created. The original congestion tree is not actually moving as such, but as both events happen at the same time, with contributors changing focus from one hotspot to another, we will think of it as a congestion tree moving from one part of the network to another.

matter if the contributors of the moving trees are C or B nodes, or both. As we shorten the lifetime of each hotspot, the forest of moving congestion trees becomes increasingly more dynamic. Different trees will grow and shrink around the network in a continuously more nonsystematic way, and the traffic pattern as a whole is moving closer to a chaotic scenario where knowledge about the traffic is limited. Such a traffic scenario could resemble a cluster running a set of virtual machines or virtual jobs, where the communication pattern is unknown, depending on the jobs currently running. Short (and long) lived congestion can appear anywhere in the network at any given time, depending on the lifetime of the hotspots, the number of contributors and their nature.

In a forest of moving congestion trees, the IB CC mechanism faces new challenges as the dynamics increases. Not only is it still important to detect congestion fast, decrease the injection rates accordingly, and separate flows contributing to congestion from other flows at a source node, but it also becomes important to recover fast from congestion. That is, the injection rate of a flow contributing to congestion should at a source node be increased again fast enough to ensure good utilization of network resources as soon as the congestion disappears. It is an indisputable fact that a feedback control loop like the one the IB CC mechanism relies upon, can lead to a CC mechanism constantly operating (too far) behind schedule as it takes time to bring information about the congestion from the root of the congestion tree back to the contributors. The shorter the lifetime of the hotspots, the more challenging it will be for the CC mechanism to react fast enough to keep up with the actual situation in the network, both when decreasing the injection rate as congestion is detected, as well as increasing it again when congestion is resolved. An interesting question is then if this really poses a problem, as the shorter lifetime of the hotspots, and the correspondingly more dynamic traffic pattern, by itself implies that the occurrences of congestion will be less severe, and the negative effects of congestion less dramatic. The dynamic traffic pattern will by its own nature reduce the severity of the HOL blocking in the network.

The Diverse and Stormy Forest of Moving Congestion Trees

It can be argued that the classification of congestion trees given above is somewhat artificial. In a network, even if the main contributors to congestion create a silent congestion tree, other small and short lived congestion trees will be created by other traffic flows present in the network. In addition, HOL blocking will make congestion trees grow towards victims of congestion, and not only towards the initial contributors like shown in figure 2. Furthermore, a silent congestion tree can be seen as a windy congestion tree with $p = 1$, and a moving congestion tree can be seen as two separate events where one congestion tree disappears while another one is created. Typically, in a network, we will find a multitude of different congestion trees that all have their own twists. The congestion trees truly come in all kinds of flavors.

Still, even if a network in general can grow a diverse and stormy forest of moving congestion trees, the classification of congestion trees into silent, windy, and moving trees has proved to be very useful when conducting a systematic study of the IB CC mechanism's performance as the degree of dynamic behaviour in the network increases. By controlling the type of main contributors to congestion, being C or B nodes, and the duration of each hotspot, we control the main types of congestion trees that will form in the network. The main congestion trees are the ones causing the most HOL blocking, and by that the main reasons for the performance degradation observed in the network. Then, by starting with contributors that create silent trees, and later moving on to nodes creating windy and moving congestion trees, we can study how well the IB CC mechanism is able to cope with an increasingly more dynamic traffic pattern in the network. In addition, notice that even though we focus on the three different types of congestion trees, in our simulations (as in a real network) any background traffic and the corresponding HOL blocking introduced, will obviously both create a multitude of small congestion trees, as well as make the main trees grow towards victim nodes. This behavior will be captured and included in the overall network performance measurements presented in section V.

IV. The Simulator

Our network simulator and switch model is built on the OMNet++ platform [19] and has been previously described in [20]. Below we give a brief overview of the model and the simulation parameters used in all the simulations presented in section V. The IB model consists of a set of simple and compound modules to simulate an IB network with support for the IB flow control scheme, arbitration over multiple virtual lanes, congestion control, and routing using linear forwarding tables.

The two building blocks for creating networks using the IB model are the *Host Channel Adapter* (HCA) compound module and the *Switch* compound module. The *Switch* consists of a set of *SwitchPort*s, which are by themselves compound modules. During a simulation, an HCA represents both a traffic injector and a traffic sink in the network, while a *Switch* acts as a forwarding node. The HCAs and *switches* are connected using *gates*, corresponding to links in the network.

The HCAs and the *SwitchPort*s consist of the a set of common simple modules *ibuf*, *obuf*, *vlarb*, and *ccmgr*, while the simple modules *gen* and *sink* are exclusive to the HCAs. The *ibuf* represents an input buffer with support for virtual lanes, virtual output queuing (VoQ) and virtual cut through switching. The *obuf* represents a simple output buffer, while the *vlarb* implements round robin arbitration over the

different VLs and multiple input ports (if the module is part of a *SwitchPort*). The *gen* implements traffic generation in a HCA, while the *sink* is the part of the HCA responsible for removing traffic from the network. The *gen* module supports several traffic generation schemes, e.g. varying the injection rate, the packet size and the destination node distribution.

In general, the *gen* module at an HCA generates traffic that is forwarded through the *vlarb* to the *obuf* of the HCA. From there it is sent out into the network. There is no internal HoL blocking in the *gen* module or in the HCA so the only place where the traffic can experience blocking is in the switches. At a *Switch*(*Port*) the *ibuf* receives the traffic, does the routing decision and moves the traffic into the corresponding VoQ. Here the traffic waits until the *vlarb* of the given output port grants access to the corresponding *obuf*. At an HCA the *ibuf* receives traffic and forwards it to the *sink*. The IB flow control is managed by the *ibuf*s and the *obuf*s, exchanging flow control messages.

The InfiniBand Congestion Control mechanism is implemented by the simple module *ccmgr*. The *ccmgr* is included in the compound modules for the HCA and the *SwitchPort*, and manages everything related to congestion control in these modules, with the help of the other simple modules therein. In particular, all CC parameters specified in the InfiniBand Architecture Specification release 1.2.1 [9] are supported by the *ccmgr* module. The throttling mechanism operates at the packet level. That is, all the flits belonging to the same packet are always injected back-to-back, if allowed by the link-level flow control mechanism, even when the throttling mechanism is active. The IRD calculations follow the IB specification. The simulation model, including the CC behaviour, has been carefully tuned against Mellanox MTS3600 InfiniBand switches as described in [20].

As our simulation scenario we have selected a three stage fat-tree network, which is a topology common in large scale InfiniBand switches [21], [22]. This topology supports non-blocking communication of up to 648 compute nodes and is built from 54 36-port crossbars. In our simulations we used a link speed of 20 Gbit/s (4x DDR) and an MTU size of 2048 bytes. Each message sent by a node consists of two packets, giving a total size of 4096 bytes per message. A node injects packets at full link speed whenever possible. The traffic patterns used correspond to the scenarios described in section III, while the specific destination distributions used are given during the discussion of the results in section V. Frame I gives a detailed example of how traffic is generated at a node during a simulation, here using a B node with $p = 50$.

The congestion control parameter values used during all the simulations discussed in this paper are listed in table I. These values were found during our initial study of CC capable IB hardware, a work presented in [7]. While it proved to be a nontrivial task to identify the parameter values, even in a small cluster with a simple and static traffic pattern, the values found in [7] have proved to be quite robust, as we will see in section V. Note that the CCT values have been increased to reflect the larger number of possible contributors to congestion in our fat-tree topology, compared to our earlier hardware experiments.

Frame I - *packet generation at a* B *node with* p=50:

A B node with $p = 50$ sends 50% of its generated traffic to its designated hotspot, hs, while the rest is sent using a uniform destination distribution including all nodes in the network (expect the node itself). Using a message size of two packets, corresponding to a total of 4096 bytes per message, the sequence of packets generated and sent could then look like this:

- 2 x Msg sent with random destination addresses (most likely two different destinations)
- 1 x Msg sent to hotspot hs (=4KB)
- 1 x Msg sent with random destination address
- 3 x Msg sent to hotspot hs (=12KB)
- 2 x Msg sent with random destination addresses (most likely two different destinations)
- 1 x Msg sent to hotspot hs (=4KB)
- ... and so on.

In a scenario of moving congestion trees, the B node changes the address of the hotspot at each new timeslot (e.g. each 1msec); the hs_i address changes into hs_j, and by that the hotspot is moved, while the value of p remains the same.

The packets are sent back-to-back when not held back by the CC mechanism or the link-level flow control. Note though, that the $p\%$ and $(1-p)\%$ fractions of a B node are related to the (simulation) time, and not each other. That is, after a given time, t, a maximum of $p\%$ of the traffic has been sent to the hotspot, while a maximum of $(1-p)\%$ has been sent to the non-hotspots. The link will remain idle when non-hotspot traffic has fulfilled its $(1-p)\%$ of the total possible traffic sent (t times link capacity) and the hotspot traffic is held back by the CC mechanism. It is important to keep the two types of traffic (hotspot and non-hotspot) independent of each other, i.e. it is important that non-hotspots traffic is not HOL blocked internally in the generator when the hotspot traffic is held back by the CC mechanism, while at the same time it is just as important that the non-hotspot traffic does not exceed the $(1-p)\%$ during a simulation as this is the amount of traffic supposedly requested by non-hotspot traffic during a simulation time.

Note that by changing the value of p, the fraction of traffic going to the hotspot changes, and by that, the likely (and average) lengths of the *trains of packets* going to the hotspot changes as well. When using a B node with a p value of 50%, most trains going to the hotspot will have sizes in the range of [4KB, a few KB> while most trains going to non-hotspots will have the size of a single message, 4KB, due to the uniform destination distribution.

Table I
CC PARAMETER VALUES.

Parameter	Value	Parameter	Value
$CCTI\_Increase$	1	$Threshold$	15
$CCTI\_Limit$	127	$Marking\_Rate$	0
$CCTI\_Min$	0	$Packet\_Size$	0
$CCTI\_Timer$	150		

V. SIMULATION RESULTS

We have divided the presentation and the analysis of our simulation results into three sections, corresponding to the

Table II
PERFORMANCE NUMBERS (GBPS), SILENT CONGESTION TREES.

No hotspots, no CC	
Avg. receive rate	2.699
No hotspots, CC on	
Avg. receive rate	2.701
Hotspots, no CC	
Hotspots avg. rcv.	13.602
Non-hotspots avg. rcv	0.168
Hotspots, CC on	
Hotspots avg. rcv.	13.279
Non-hotspots avg. rcv	2.246
Totale Network Throughput, Hotspots	
Without CC	216.073
With CC	1543.793

three categories of congestion trees introduced in section III. In the first section, section V-A, we start by looking at the performance of the IB CC mechanism in a network growing a quiet forest of eight silent congestion trees. Then, in section V-B, we continue our study by gradually exchanging the nodes in the network with B nodes, making the congestion trees increasingly more windy, and by that, the traffic pattern in the network more dynamic. Finally, in section V-C, we end our study by looking at the performance of IB CC as the hotspots move. By increasing the number of times the hotspots move during a given timeslot, we decrease the hotspot lifetimes accordingly, and by that, the dynamics of the traffic pattern in the network increases even further.

A. The Silent Forest of Congestion Trees

In this scenario, the end nodes in the network are divided into 80% C nodes and 20% V nodes. Recall that this means that 80% of the 648 nodes in the network send traffic solely to the hotspots. The rest of the nodes, the V nodes, send traffic with a uniform destination distribution. The V nodes are randomly distributed in the topology. All nodes constantly try to inject traffic into the network at maximum capacity; 13.5Gbps[5].

Before enabling the C nodes, we simulate a scenario where only the V nodes are active and the CC mechanism disabled. Doing this, we get the throughput of the nodes in the network that potentially will be victims of congestion when the hotspots are introduced. As shown in the first part of table II, the average receive rate of the nodes in the network is approximately 2.7Gbps. This is as expected, as each V node injects traffic at 13.5Gbps into the non-blocking topology. The soon-to-be hotspots and the non-hotspots receive the same amount of traffic. These performance numbers remain the same if we enable CC, still without activating the C nodes (the second part of table II). Enabling CC causes no harm to the network performance in this lightly loaded network.

Now let us disable the CC again, and enable the C nodes. The C nodes are evenly divided into eight subsets, each subset sending to one of eight hotspots. The third part of table II shows the average receive rates of both the newly created hotspots and the non-hotspots. The hotspots, randomly distributed in the network, each receives 13.6Gpbs[6], while the non-hotspots have their average receive rate lowered from 2.7Gpbs down to 0.168Gbps. The growth of the eight silent congestion trees results in a huge amount of HOL blocking in the network, and by that a severe performance degradation for the V nodes. The V nodes have become victims of congestion. To remove the congestion trees and the HOL blocking, we turn the CC back on. The fourth part of table II shows the resulting improvement in performance. At the cost of a small drop in the receive rate for the hotspots, down 2.5%, we improve the receive rate of the non-hotspots by more than 1200%. Enabling the CC mechanism of IB, the non-hotspots now experience a receive rate only 17% below the receive rate they achieved before the hotspots were introduced. The last part of table II shows the total network throughput of the network running with and without CC enabled. Even when accounting for the drop in the receive rate of the hotspots when CC is enabled, enabling CC leads to a performance improvement by more than 610%. The CC mechanism, using the CC parameters from Table I, is clearly able to improve the performance in our network when silent congestion trees are present.

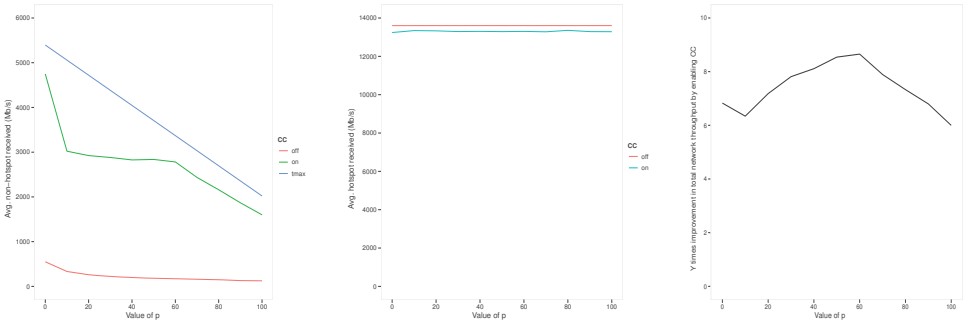
B. The Windy Forest of Congestion Trees

Let us now increase the dynamics of the traffic pattern used in the previous section by gradually exchanging $x\%$ of the nodes in the network with B nodes, increasing x in steps of 25%. The eight permanent hotspots are still present, but as x increases so does the amount of wind in our congestion trees. The B nodes are evenly divided into eight subsets, just like the C nodes, each subset sending to one of the eight hotspots. The nodes not being B nodes, i.e. $(100 - x)\%$ of the nodes, are divided into 80% C nodes and 20% V nodes – as before. Note that until x reaches 100%, this implies that there are always some permanent contributors to congestion present in the network (and some permanent potential victims). The figures 5, 6, 7, and 8 show performance plots from simulations ran with $x = 25\%$, $x = 50\%$, $x = 75\%$, and $x = 100\%$, respectively. These four scenarios and their corresponding figures will be discussed in the following subsections.

1) 25% B Nodes: The figures 5(a) and 5(b) show the average receive rates of non-hotspots and hotspots, respectively, in a network running with and without CC, as p

[5]This corresponds to the injection rate of the end nodes the simulator is tuned against, an injection rate limited by the PCIe v1.1 protocol overhead and other system components in the hardware.

[6]This matches the receive rate of the end nodes the simulator is tuned against. The hardware has a receive rate approximately 0.1Gbps higher than the injection rate.



(a) Average receive rate, non-hotspots. (b) Average receive rate, hotspots. (c) Total network throughput improvement.

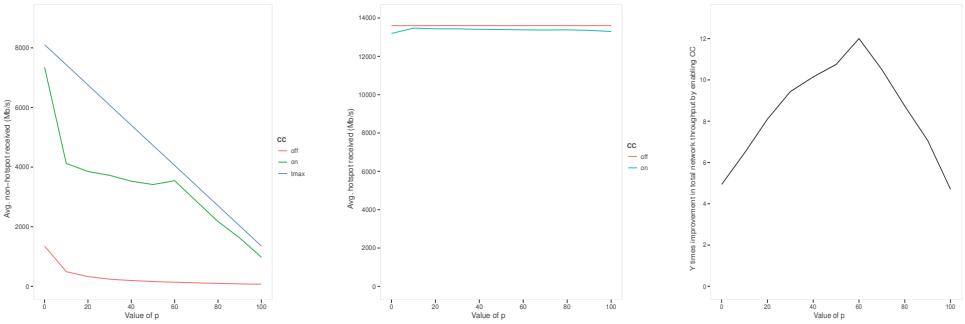
Figure 5. Measurements for a windy forest with 25% B nodes as p increases.

increases from 0 to 100. Recall that a B node sends p% of its traffic to a given hotspot. The remaining $(1-p)$% of the traffic is sent using a uniform destination distribution. Note that this implies that as p increases, a decreasing amount of traffic in the network is destined for the non-hotspots. That is, as p increases, the theoretical maximum amount of traffic that the non-hotspots can receive decreases. This theoretical maximum is plotted as *tmax* in figure 5(a), and represents the maximum average receive rate the non-hotspots could possibly achieve if the hotspots were not present.

Looking at figure 5(a), it is evident that enabling CC in the network results in an immense improvement in the average receive rates for the non-hotspots, independent of the value of p. At $p = 0$ the average receive rate of a non-hotspot is 4.75Gbps when CC is enabled, compared to 0,55Gbps when CC is disabled and the tmax value of 5.4Gbps. In this case, enabling CC leads to an 8.6 times improvement in performance; a 760% increase. As p increases from 0 to 10, the performance when CC is enabled drops from 88% to 60% of tmax, while the relative performance actually increases to a factor of 9.1 compared to the scenario with CC disabled. As the B nodes start to send traffic to the hotspots, but still generate 90% uniformly distributed traffic, the CC mechanism is not fully able to cope with all the HOL blocking in the network. As p increases from 10 to 60, however, the relative performance of CC compared to tmax increases to 80% again, where it stays as p reaches the value of 100. The performance increase by enabling CC in the network, as the p values increases from 30 to 100, equals to a factor ranging from 12.9 to 16.3, with the peak at $p = 60$; a 1530% performance boost in the receive rate of the non-hotspots at this point.

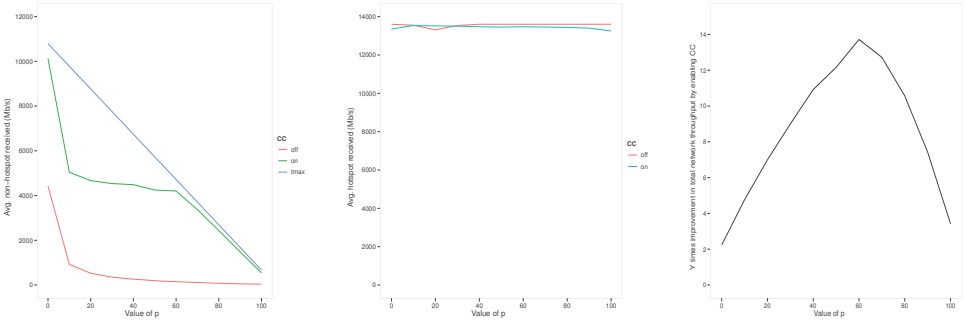
Turning our attention towards the hotspots, figure 5(b) shows that the average receive rates of these nodes are independent of the values of p. When CC is disabled, each hotspot receives a steady 13.6Gbps, which equals to the maximum receive rate of an end node. The average receive rate then drops by 2.2%, down to 13.3Gbps, when the CC is enabled. While this indicates a tiny underutilization of the scarce resources at the roots of the congestion trees, bearing the results from figure 5(a) in mind, the price we have to pay when enabling CC is negligible. Figure 5(c) plots the improvement in the total network throughput when CC is enabled as a function of p. By enabling CC, we improve the total network throughput by a factor ranging from 6.0 ($p = 100$) to 8.7 ($p = 60$). That is, a minimum improvement in performance by at least 500%.

2) 50% and 75% B Nodes: Figures 6 and 7 show performance plots from simulations where the fraction of B nodes in our network is 50% and 75%, respectively. Comparing these results with the ones presented in figure 5, we observe that the performance trends are the same. Enabling CC leads to a vast improvement in the average receive rates of the non-hotspots, without penalizing the average receive rates of the hotspots. Notice that as the fraction of B nodes increases and the fraction of C and V nodes decreases, the impact the value of p has on the overall traffic pattern in the network increases accordingly. This influences the tmax values of the average receive rates of the non-hotspots. At $p = 0$, the tmax value increases as the fraction of B increases, because the traffic pattern in the network as a whole then moves towards a uniform destination distribution. On the other hand, at $p = 100$, the tmax value decreases as the fraction of B nodes increases. At this p value, a B node sends all its traffic to a hotspot, and then as the fraction of B nodes increases, the fraction of the traffic in the network headed for the hotspots increases accordingly. To sum up, the graph of decreasing tmax values as a function of p, becomes steeper as the fraction of B nodes in the network increases. This influences the possible



(a) Average receive rate, non-hotspots. (b) Average receive rate, hotspots. (c) Total network throughput improvement.

Figure 6. Measurements for a windy forest with 50% B nodes as p increases.

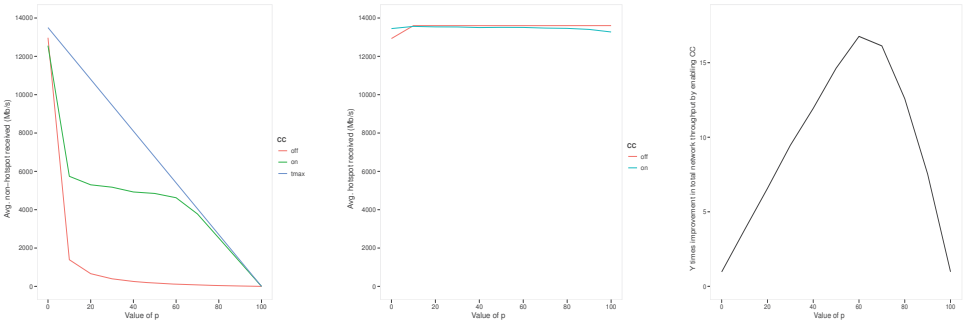


(a) Average receive rate, non-hotspots. (b) Average receive rate, hotspots. (c) Total network throughput improvement.

Figure 7. Measurements for a windy forest with 75% B nodes as p increases.

benefit we can achieve by enabling CC in our network, an effect clearly present in the graphs showing the total network throughput improvement (figure 6(c) and 7(c)). The graphs become increasingly more $\cap$ shaped as the fraction of B nodes increases. At low and high p values the total network throughput improvement decreases, while the peak at $p = 60$ increases. This effect becomes even clearer as we increase the fraction of B nodes to 100%.

3) 100% B Nodes: Figure 8 shows the performance plots as the fraction of B nodes has increased to 100%. We are creating purely windy congestion trees with no traffic generated by V and C nodes. Now, for the first time, we observe a small penalty of 3% in the receive rate of the non-hotspots at $p = 0$ as CC is enabled (figure 8(a)). Note though that at this p value, the B nodes have no preference for the hotspots. The traffic is uniformly distributed in the network, and there is no real congestion for the CC to resolve. As soon as the B nodes start to send traffic to the hotspots (at $p > 0$), the improvement by enabling CC is again evident. At $p = 10$, enabling CC leads to a 4.1 times improvement in the receive rate of the non-hotspots, an improvement increasing to 64.1 times as p approves 90. In this scenario, the non-hotspots experience a total collapse in performance in a network without CC, while when enabling CC, the same nodes experience a receive rate very close to the theoretical maximum when $p > 60$. Figure 8(b) shows that enabling CC has no negative effect on the receive rate of the hotspots. Finally, looking at the total network throughput improvement in figure 8(c), the IB CC mechanism's ability to improve network performance when congestion is present in the network is clear. At $p = 0$ and $p = 100$, enabling CC has virtually no effect, as the CC mechanism is left with no room for improvement - as explained in the previous section. Note though, that the CC mechanism does not



(a) Average receive rate, non-hotspots. (b) Average receive rate, hotspots. (c) Total network throughput improvement.

Figure 8. Measurements for a windy forest with 100% B nodes as p increases.

cause any harm at these extreme p values. With p values in the interval $< 0, 100 >$, the CC mechanism shows a performance improvement, with a peak at p=60, leading to a seventeen-fold increase in total network performance.

Summing up, a network with CC enabled outperforms a network running without CC as long as congestion is present in the network, no matter how windy the congestion trees are, showing a peak in performance when approximately 60% of the traffic is headed towards the hotspots. Furthermore, the CC mechanism causes no harm to the network performance when all traffic is headed for the hotspots, and the reduced performance caused by CC in a network with purely uniformly distributed traffic is negligible.

C. The Stormy Forest of Moving Congestion Trees

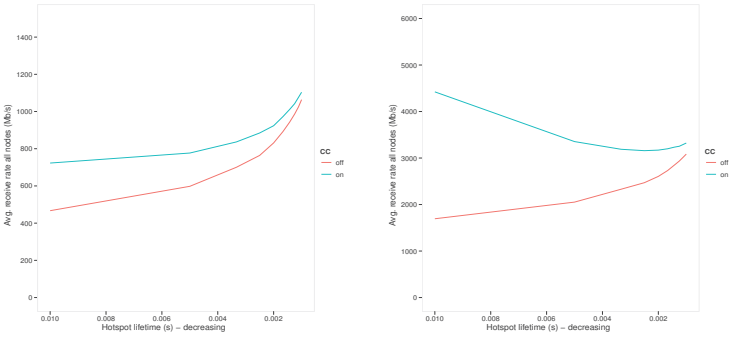
This far, all our hotspots were locked to a permanent position in the network during a simulation. To continue our study of the performance of the IB CC mechanism as the dynamics in the network increases, we now start to move the hotspots. During a simulated timeslot of 0.1s, the hotspots are moved n times, n ranging from 10 to 100. This corresponds to a shortening of the hotspot lifetimes from 10ms to 1ms. By measuring the performance of the IB CC for different values of n, we can to study the performance of the CC mechanism in a systematic way as the dynamics in the network increases. The CC now needs to deal with contributors that themselves, dynamically, tear down and recreate congestion trees in the network. In addition, the moving process itself may create temporarily congestion trees at unforeseeable places in the network. By moving the hotspots, we turn our network into a stormy forest of moving congestion trees.

We start our study by moving silent congestion trees, before we move on to a scenario where we move windy congestion trees. The contributors are as before divided into eight subsets, each subset sending to one of the eight hotspots.

Figure 9(a) shows the average receive rate of all nodes in a network consisting of 20% V nodes and 80% C as a function of the decreasing hotspot lifetime. When the hotspots are moved each 10th ms, the nodes receive 723Mbps when CC is enabled, compared to 467Mbps in a network without CC. The performance increases by 55% when enabling CC. Then, as the lifetime of the hotspots are shortened, the improvement in performance by enabling CC is reduced. When the hotspots move every second millisecond, the performance increase is down to 10%, and as the hotspot lifetime reaches 1ms, the performance increase is only 4%. The performance increase is less impressive than the ones achieved when the hotspots are not moving, but the benefit from enabling the IB CC mechanism is still clear even as the hotspot lifetime approaches 1ms.

Note that as the lifetime of the hotspots decreases, the receive rate increases in general, while the advantage from enabling CC decreases. When the hotspot lifetime decreases, the contributors change focus more often and by that they actually distribute the traffic more evenly in the network. The result is that the network throughput as a whole increases, as it is less dependent on the limited number of hotspots' ability to receive traffic. At the same time, the change in focus of the contributors will help to resolve congestion in the network. This also implies that the CC mechanism is left with less room for improvement. In addition, when the lifetime of the hotspots decreases, it becomes increasingly hard for IB CC mechanism to keep up with the situation in the network (due to the feedback loop).

A network with 20% V nodes and 80% C nodes has relatively few possible victims of congestion; only 20% of the nodes. Figure 9(b) shows the performance numbers we get if we increased the number of V nodes to 60%. Now,



(a) 20% V nodes, 80% C nodes.

(b) 60% V nodes, 40% C nodes.

Figure 9. Average receive rates in a network with silent congestion trees and moving hotspots, shown as a function of decreasing hotspot lifetimes.

the advantage of enabling CC has increased again when the hotspot lifetime is 10ms. At this point, enabling CC results in an increase in the average receive rate by a factor 2.6. The advantage by enabling CC decreases faster, however, than when having 20% V nodes. At a hotspot lifetime of 1ms, the improvement is down to 10% again.

Finally, figure 10 shows the average receive rate of the nodes in a network consisting of only B nodes, using p values of 30, 60, and 90. As we move the hotspots and their corresponding windy congestion trees, we observe the same performance trends as we did when moving the silent congestion trees. The enabling of CC leads to an improvement in performance in all cases, even though the improvement decreases as the hotspot life time decreases and the traffic pattern itself alleviate the side effects of congestion in the network.

VI. Conclusions

A central question in the understanding of congestion control in interconnection networks is whether throttling of contributors to congestion may have adverse effects. Setting parameters related to the feedback-loop of such mechanisms have previously been shown to require deep understanding of the problem, even for simple traffic. Therefore uncertainty has lingered as to whether congestion control may actually be harmful in some scenarios.

In this paper we have shown that for fat-trees, there exist parameter settings that makes the throttle-based Congestion Control of InfiniBand robust. Through carefully designed experiments we have stressed the mechanism with congestion scenarios varying from the completely static ones, to cases where congestion is highly dynamic both with respect to contributors, intensity, duration and placement of congestion points. Our identified parameter settings showed increased throughput varying from a few percent, to a seventeen-fold increase. The only adverse effect we registered was a negligible decrease in throughput for the contributors to congestion.

Our most important result is that InfiniBand congestion control can be tuned to be stable for a given installation based on fat-trees. The tuning itself remains a highly specialized task [7], but the gains in performance is huge when it is done correctly. Regarding other topologies, the question is still open. There is reason to believe that other multistage-topologies that have a similar pattern of inter-relations between streams, will expose the same behavior. Regarding Tori or Meshes, the picture is more unclear, thus this question should form the basis for further research.

References

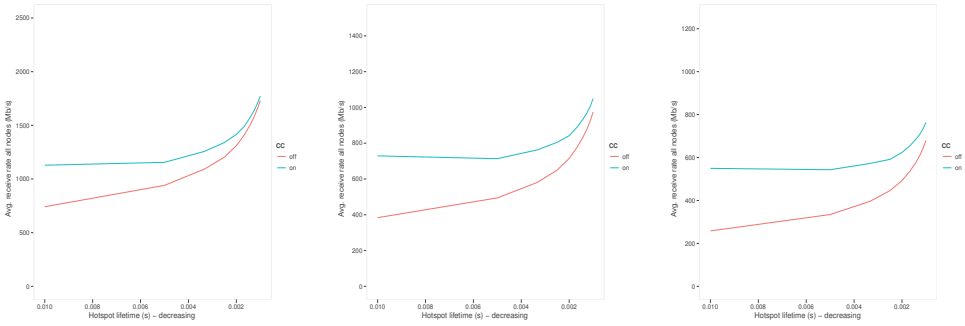
[1] G. F. Pfister and V. A. Norton, “”Hot Spot” contention and combining in multistage interconnection networks,” *IEEE Trans. Computers*, vol. 34, no. 10, pp. 943–948, 1985.

[2] W. Dally, “Virtual-channel flow control,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[3] P. García, J. Flich, J. Duato, I. Johnson, F. Quiles, and F.Naven, “Dynamic evolution of congestion trees: Analysis and impact on switch architecture,” in *High Performance Embedded Architectures and Compilers*, 2005, pp. 266–285.

[4] V. Jacobson, “Congestion avoidance and control,” in *SIGCOMM*. ACM, 1988, pp. 314–329.

[5] L. S. Brakmo and L. L. Peterson, “TCP vegas: End to end congestion avoidance on a global internet,” *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465–1480, 1995.



(a) B nodes with $p = 30$. (b) B nodes with $p = 60$. (c) B nodes with $p = 90$.

Figure 10. Average receive rates in a network with 100% B nodes and moving hotspots, shown as a function of decreasing hotspot lifetimes.

[6] C. Parsa and J. Garcia-Luna-Aceves, "Improving TCP congestion control over internets with heterogeneous transmission media," in *7th International Conferance on Network Protocols (ICNP99)*. IEEE Computer Society, 1999, pp. 213–221.

[7] E. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, Apr. 2010, pp. 1–12.

[8] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-end congestion control for InfiniBand," in *INFOCOM*, 2003.

[9] *Infiniband architecture specification*, 1st ed., InfiniBand Trade Association, November 2007.

[10] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification.*, IEEE 802.1Qau-2010 ed., IEEE 802 LAN/MAN Standards Committee, 2010. [Online]. Available: http://www.ieee802.org/1

[11] J. R. Santos, Y. Turner, and G. J. Janakiraman, "Evaluation of congestion detection mechanisms for InfiniBand switches," in *IEEE GLOBECOM – High-Speed Networks Symposium*, 2002.

[12] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Congestion management in MINs through marked and validated packets," in *PDP*, 2007, pp. 254–261.

[13] ——, "On the influence of the packet marking and injection control schemes in congestion management for MINs," in *Euro-Par*, 2008, pp. 930–939.

[14] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 108–119.

[15] J. Escudero-Sahuquillo, P. GarcÃa, F. Quiles, J. Flich, and J. Duato, "FBICM: Efficient congestion management for high-performance networks using distributed deterministic routing," in *High Performance Computing - HiPC 2008*, ser. Lecture Notes in Computer Science, P. Sadayappan, M. Parashar, R. Badrinath, and V. Prasanna, Eds. Springer Berlin / Heidelberg, 2008, vol. 5374, pp. 503–517.

[16] "Top 500 supercomputer sites," http://top500.org/, November 2011.

[17] E. G. Gran, E. Zahavi, S.-A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, "On the relation between congestion control, switch arbitration and fairness," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGRID '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 342–351. [Online]. Available: http://dx.doi.org/10.1109/CCGrid.2011.67

[18] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using InfiniBand architecture congestion control," Invited paper in High Performance Interconnects for Distributed Computing, july 2005.

[19] "Omnet++ network simulation framework," http://www.omnetpp.org/.

[20] E. G. Gran and S.-A. Reinemo, "InfiniBand congestion control, modelling and validation." OMNeT++ 2011, Barcelona, Spain.

[21] Sun Microsystems, "SUN DATACENTER INFINIBAND SWITCH 648 ARCHITECTURE AND DEPLOYMENT," Sun Microsystems, Tech. Rep. June, 2009.

[22] "IS5600 - 648-port InfiniBand Chassis Switch," Mellanox Technologies, http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5600.pdf.

Paper V

Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks

Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García,
José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato

2011 International Conference on Parallel Processing

Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks

Jesus Escudero-Sahuquillo*, Ernst Gunnar Gran†, Pedro Javier Garcia*, Jose Flich‡,
Tor Skeie†, Olav Lysne†, Francisco Jose Quiles* and Jose Duato‡

*Dept. of Computing Systems, University of Castilla-La Mancha, Spain.
Email:{jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es
†Simula Research Laboratory, Norway. Email:{ernstgr, tskeie, olav.lysne}@simula.no
‡Dept. of Computer Engineering, Technical University of Valencia, Spain. Email:{jflich,jduato}@gap.upv.es

***Abstract*—Existing congestion control mechanisms in interconnects can be divided into two general approaches. One is to throttle traffic injection at the sources that contribute to congestion, and the other is to isolate the congested traffic in specially designated resources. These two approaches have different, but non-overlapping weaknesses. In this paper we present in detail a method that combines injection throttling and congested-flow isolation. Through simulation studies we first demonstrate the respective flaws of the injection throttling and of flow isolation. Thereafter we show that our combined method extracts the best of both approaches in the sense that it gives fast reaction to congestion, it is scalable and it has good fairness properties with respect to the congested flows.**

***Keywords*-Interconnection Networks; Congestion Management; HoL-blocking;**

I. Introduction

In interconnection networks [1], traffic congestion may degrade the network and overall system performance if no countermeasures are taken [2]–[4]. Congestion is simply a result of high load of traffic fed into a network link, exceeding the link capacity at that point. Hot spot traffic patterns, network burstiness, re-routing around faulty regions, and conducting link frequency/voltage scaling (lowering the link speed in order to save power), can all lead to congestion. If all these factors are known in advance, the network administrator may alleviate the consequences by effective load balancing of the traffic, but typically this is not the case. Furthermore, in cases where multiple nodes send more data to a single destination than the node can handle, no dynamic re-routing can be done to avoid network congestion. It becomes even more severe when a parallel computer is running multiple different jobs as an on-demand service (e.g. cloud computing), where the resulting traffic pattern becomes unpredictable.

Congestion control (CC) as a countermeasure for relieving the effects of congestion has been widely studied. In particular, this problem is well understood and solved by dropping packets in traditional lossy networks such as local area networks (LANs) and wide area networks (WANs). In these environments packet loss and high latency are indications of network congestion. Herein it is mainly TCP that implements end-to-end CC, either by a traditional window control mechanism [5] for detecting dropped packets or through changes in latency [6], [7]. Very often those networks are also over-provisioned in order to avoid congestion.

In high performance computing (HPC) data centers low latency is crucial, and packet dropping and retransmission are not allowed under regular circumstances due to loss of performance. Lossless behavior is achieved with credit based link-level flow control, which prevents a node or a switch from transmitting packets if the downstream node or switch lacks buffer space to receive them.

Typically, when congestion occurs in a lossless network, a congestion tree starts to build up due to the backpressure effect of the link-level flow control. The switch where the congestion starts will be the root of a congestion tree that grows towards the source nodes contributing to the congestion. This effect is known as congestion spreading. The tree grows because buffers fill up through the switches in the network as the switches run out of credits (not necessarily in the root). As the congestion tree grows, it introduces head-of-line (HoL) blocking [8] that slows down packet forwarding. HoL-blocking appears when the packet of the head of a queue is blocked and prevents packets behind it from advancing. This effect also affects flows which are not contributing to the congestion, severely degrading the entire network performance. The HoL- blocked flows become victims of congestion [8].

CC for link-level flow controlled networks cannot be based on a traditional window control mechanism as deployed in TCP, though it effectively limits the amount of buffer space that a flow can occupy in the network [9]. The reason for this is the relatively small bandwidth-delay product in this environment, where even a small window size may saturate the network [8]. A rate control (throttling) based CC mechanism is more appropriate for link-level flow controlled networks, since it increases the range of control compared to a window-based system. The mechanism relies on the switches to detect congestion, and inform the sources that contribute to the congestion they must reduce their corresponding injection rates. By reducing the injection rate, the sources remove the congestion tree and by that the HoL-blocking as well.

A problem associated with throttling-based CC is that it takes time from a switch detects congestion until the sources contributing to congestion are notified about congestion. During that time HoL-blocking degrades network performance. This slow reaction has inspired Duato et. al. to take a completely different approach for CC [10], [11]. Instead of removing the congestion tree itself, this approach strives to relieve the unfortunate side effects the congestion tree has on flows not contributing to the congestion. That is, the HoL-blocking is removed by using special set-aside-queues for contributors to congestion, effectively making it possible for

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.80

victim flows to bypass the congested flows without actually removing the congestion tree. Such an approach has the advantage of being able to react immediately and locally at each switch, at the cost of the extra buffers needed for the set aside queues and the added complexity in the switch to manage them. Isolating the congested flows in this way does, however, not address the real cause of the problem, sources injecting too much traffic into the network. The congestion trees themselves are left untouched. This poses a scalability challenge as the number of congestion trees could exceed the resources available for the set aside queues in the switches. If this happens, congestion trees will grow inside the queues supposed to be used only for victim flows, and by this reintroduce HoL-blocking and lead to performance degradation in the network.

Summing up, an injection throttling mechanism is able to remove the congestion tree, and by that the introduced HoL-blocking, but the mechanism has a challenge when it comes to reaction time. It operates behind schedule. A mechanism based on congested-flow isolation, on the other hand, reacts immediately but faces scalability issues as the number of congestion trees increases. In addition, there is another less obvious difference between the two CC mechanisms. A throttling mechanism has been shown to have the potential of improving fairness in the network by solving the well known parking lot problem [12]. This problem arises when several flows are stored in a queue at a switch while another flow addressed to the same output port is the sole user of another different queue. The flows sharing the same queue are granted access to the requested output port with less frequency, than the flow being the sole user of the queue. The throttling mechanism solves the parking lot problem by decreasing the injection rate on a per flow basis at the sources. On the other hand, a congested-flow isolation mechanism can actually have a negative effect on fairness depending on the arbitration in the switches.

In this paper we present *Combined Congested-Flow Isolation and Throttling* (CCFIT), a novel mechanism which combines the ideas of congested-flow isolation with a throttling mechanism. Using simulations, we show that CCFIT is able to remove HoL-blocking immediately, assure scalability by removing the congestion trees, improve fairness in the network, and last but not least, CCFIT achieves an allover higher throughput than injection throttling and congested-flow isolation do as standalone concepts.

The remainder of the paper is organized as follows: In Section II we discuss previous work related to CC in interconnection networks, to place our proposed CCFIT mechanism into the proper context. Section III contains a thorough description of the CCFIT mechanism, while in Section IV we evaluate the mechanism using results from simulation studies. In Section V we conclude the paper.

II. Related Work

Congestion control (CC) based on injection throttling is a popular approach to congestion handling. As aforementioned, the basic idea is to detect congestion in the network at the switches, then to notify the contributing sources about the congestion, and finally for the contributors to stop or cease traffic injection. This closed-loop feedback control philosophy is the basic approach of several proposals which, on the other hand, differ in several aspects. For instance, notifications could be sent to all the sources [13] or just to the sources contributing to congestion [14]. Other proposed mechanisms [15] notify congestion just to the local endpoints attached to the switch where congestion is detected. Furthermore, the switches can mark the packets contributing to congestion in order to notify the destinations about the situation, which subsequently notify the sources (the forward explicit notification approach), or the switches can themselves generate notification packets that are sent directly to the source nodes (the backward explicit notification approach). The InfiniBand (IB) network [16] applies the former approach, while the emerging Data Center Bridging standard [17] is implementing the latter.

There is also a body of work that propose different strategies for congestion notification and marking, e.g. a congested packet can be marked both in the input and output switch buffer, as well as being tagged with information about the severity of the congestion. Moreover, there are some different approaches for designing sources response function, i.e. the actions taken to reduce the injection rate, later followed by an increase in the rate when congestion is resolved [9], [18]–[20].

The injection throttling part of the CCFIT mechanism is inspired by the injection throttling mechanism specified for InfiniBand (IB), one of the most successful interconnect technologies. The IB Architecture Specification [16] defines two bits in the packet header for congestion notification. Specifically, if a packet is considered to contribute to congestion at a switch port, the *Forward Explicit Congestion Notification* (FECN) bit in the packet header is set. The FECN bit is then carried through the network to the destination node by the packet contributing to congestion. Upon reception of a "FECN-marked" packet, a destination will return back to the source a packet whose header will have the *Backward Explicit Congestion Notification* (BECN) bit set. Any source receiving a BECN packet will reduce its injection rate of the corresponding congested traffic flow, thus alleviating congestion.

The performance of IB CC depends on several configurable parameters. For instance, a *threshold* parameter mapped to a buffer fill ratio at a switch port[1] determines when the port is considered to be congested. If the buffer fill ratio is above the *threshold* the corresponding switch port is moved into the *Congestion State*, given that the port is also considered to be at the root of the congestion tree, that is, the port has available credits to forward packets. However, in order not to generate too many BECNs, not all the packets crossing a port in the Congestion State are "FECN-marked": Only those whose size is greater than the value of the *Packet_Size* parameter, and among them again, only a fraction corresponding to the *Marking_Rate*

[1]For simplicity, the concept of *Virtual Lanes* (VL) has been left out of this explanation of the IB CC.

parameter, are finally marked.

Similarly, the exact reaction of a source node upon the reception of a BECN also depends on a set of CC parameters. Specifically, the injection rate of a congested flow is reduced by introducing a *injection rate delay* (IRD) between consecutive packets of that flow. Source nodes store a list of possible IRD values in a *Congestion Control Table* (CCT), each congested flow holding an index (CCTI) into this table. CCT values are typically arranged in such a way that the higher the index, the greater the IRD. Upon reception of a BECN the index of a flow is increased by a value stated by the *CCTI_Increase* parameter. The CCTI is decremented again by one when a timer (whose value is stated by the *CCTI_Timer* parameter) expires. In this way, flows contributing to congestion will be throttled while congestion is present, and being released when congestion vanishes. More details about the IB CC mechanism can be found in [8].

While the IB CC mechanism, like injection throttling techniques in general, has the potential to remove the congestion tree and even improve fairness [8], the mechanism has a major drawback. The delay between congestion detection and reaction at the sources results in a CC mechanism operating behind schedule, where oscillating sources are adjusting their injection rates based on "old" information.

An alternative approach to the injection throttling mechanism is to remove the HoL-blocking without removing the congestion tree, e.g. using a congested-flow isolation technique. If HoL-blocking produced by congested flows to non-congested ones is eliminated, congestion turns harmless [4].

Many techniques have been proposed to reduce or eliminate HoL-blocking, most of them relying on having different queues at each switch port, in order to separately store packets belonging to different flows. For instance, a well-known HoL-blocking elimination technique is *Virtual Output Queues* (VOQs), either at switch level (VOQsw) [21] or at network level (VOQnet) [22]. The latter requires at each port as many queues as destinations in the network. Then, at each port, all the packets addressed to a specific destination are exclusively stored in the queue assigned to that destination, and they never share that queue with packets addressed to other destinations, thus completely removing HoL-blocking. Note, however, that VOQnet does not scale with network size. VOQsw uses as many queues at each port as output ports in the switch, so that each incoming packet is stored in the queue assigned to its output port. VOQsw scales with network size, and eliminates HoL-blocking in a switch if it is directly caused by packets contending for output ports in the same switch. Unfortunately, in switches affected by congestion spreading from other switches, VOQsw can not guarantee that congested packets do not share queues with non-congested ones, thus VOQsw just partially eliminates HoL-blocking. Other similar techniques that also reduce HoL-blocking, but do not completely eliminate it, are *Dynamically Allocated Multi-queues* (DAMQs) [23], *Destination-Based Buffer Management* (DBBM) [24], *Dynamic Switch Buffer Management* (DSBM) [25] and *Output-Based Queue-Assignment* (OBQA) [26].

All the mentioned HoL-blocking elimination techniques do not explicitly identify congested flows, but they rely on separating packets from different flows as much as possible with the available queues at each port. Their effectiveness then greatly depend on the number of queues per port. By contrast, other techniques explicitly detect and keep track of congested flows in order to isolate them in special, dynamically-assigned queues, while the non-congested flows may share queues without suffering significant HoL-blocking. In this way, the number of queues required to efficiently eliminate HoL-blocking is reduced. This is the main strategy followed by *Regional Explicit Congestion Notification* (RECN) [10], [27], *Regional Explicit Congestion Notification-Input Queued* (RECN-IQ) [28] and *Flow-Based Implicit Congestion Notification* (FBICM) [11].

In order to identify congested flows, these techniques implement some mechanism to locate congested points (i.e. to detect congestion), then identifying congested packets as those whose route crosses a congested point. In general, like the IB mechanism, these techniques detect congestion by locally monitoring queue occupancy at each switch port, and comparing it with a *Detection Threshold*. Once congestion is detected in a port, a special queue is immediately allocated to store congested packets crossing that port. Additionally, these techniques also require a set of queues, at each port, devoted to store congested packets[2], and some control memory to manage them, mainly to store the location of the congested point each special queue is assigned to. This control memory is implemented by means of a Content-Addressable Memory (CAM) present at each port. In the case of RECN and RECN-IQ, designed for source-based routing networks, each CAM line stores (among other information) the explicit path towards the root of the congestion tree assigned to a specific SAQ, while in the case of FBICM, designed for deterministic distributed-based routing, stores a set of destinations. If the occupancy of any SAQ (or CFQ) in a port reaches a specific threshold, the congestion information stored in its corresponding CAM line is propagated (by control packets) to the switch connected to that port, which in turn will allocate a new SAQ (or CFQ) for this specific congestion tree. In this way, special queues are allocated all along the way of congested flows to separate them from non-congested ones, thereby eliminating HoL-blocking. SAQs (or CFQs) are dynamically deallocated when the corresponding congestion tree vanishes, and later reallocated if a new congestion tree appears.

As mentioned in the introduction, although these solutions are quite effective, they also present some flaws, probably the most important one being the limited number of special queues per port, which may not be enough to handle all the possible congestion trees simultaneously present at a port. Note, however, that it is unlikely that many congestion trees are present in many ports at the same time, thus in most cases only a small fraction of ports would run out of SAQs. Nevertheless, any congestion tree may partially spoil

[2]These queues are called Set-Aside-Queues, SAQs, in RECN and RECN-IQ, and Congested-Flow-Queues, CFQs, in FBICM.

network performance if not suitably managed in a port.

To conclude, note that two of the most popular strategies for CC in high-performance interconnection networks, injection throttling and HoL-blocking elimination based on congested-flow isolation in dynamically-allocated queues, present different drawbacks. The initial idea behind CCFIT was that a combination of both approaches would alleviate the respective flaws. On one hand, congested-flow isolation eliminates HoL-blocking even if sources are not yet aware of the appearance of congestion. On the other hand, the throttling of congested flows would reduce the probability of having many of them simultaneously present in any port, i.e. also reducing the probability of running out of special queues in any port. In the following sections we describe and evaluate CCFIT, our new proposal to combine these approaches to CC. In addition, we also show that CCFIT in certain situations greatly improves fairness, both compared to a CC mechanism based on congested-flow isolation alone, as well as compared to a network configuration running without CC.

III. CCFIT Description

In this section, we describe the CCFIT mechanism, detailing the switch and end-node architecture, as well as their specific operation. After that, we analyze the CCFIT parameters.

A. Switch Architecture

The injection throttling part of the CCFIT mechanism is heavily influenced by the CC throttling mechanism specified for IB, evaluated in [8], [12]. This throttling mechanism is then combined with FBICM to achieve congested-flow isolation in switches using distributed routing. That is, the CCFIT switches are responsible for both detecting congestion and notifying the contributing sources, as well as isolating congested-flow packets, eliminating the HoL-blocking: When detecting congestion, a CCFIT switch moves the corresponding output port into the congestion state, mark packets using this output port by setting the FECN bits, and allocates a CFQ for packets belonging to the corresponding congestion point.

Regarding switch architecture, CCFIT does not limit the number of switch ports. Specifically, it has been developed for Input Queued (IQ) switches, where memories are only present at input ports. The current and popular IQ-switches are simpler and cheaper than the CIOQ ones, offering high bandwidth and low latency if Virtual Output Queues (VOQ) are used [29].

Regarding switch routing logic, CCFIT has been designed for networks using distributed deterministic routing (InfiniBand being a prominent example), thus routing logic can be implemented using any distributed deterministic routing hardware solution, like tabled-based routing, look-ahead routing, etc. The unique routing information packets need to include in their header is the destination they are addressed to, instead of a explicit, complete route as source routing does. Actually, this routing information and the stored congestion information allow CCFIT to detect if a packet is congested or not.

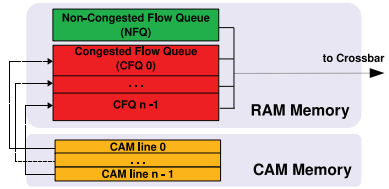


Figure 1. CCFIT Input Port Organization Diagram.

In order to deal with the HoL-blocking effect, CCFIT uses the input port organization shown in Fig. 1. Specifically, RAM is organized in queues, dynamically managed: CCFIT assumes two types of queues per input port: a normal flow queue (NFQ), where non-congested packets are stored, and a small number of congested flow queues (CFQs), where congested packets are isolated, thereby, not delaying the advance of non-congested ones. Moreover, CCFIT uses a post-processing mechanism (see Section III-C), similar to the FBICM one, in order to move congested packets from the NFQ to the CFQs. As we describe latter, the post-processing mechanism is in charge of moving an output port into the congestion state.

Like FBICM, CCFIT uses content addressable memories (CAMs) [30], in order to both keep track of the congestion information and store the CFQ status. Notice that each CAM line is associated with one CFQ. Although switch output ports have neither NFQs nor CFQs, CCFIT requires a CAM per output port, in order to propagate congestion information from a given input port CAMs to upstream input port CAMs. In that sense, CCFIT follows the same approach used in FBICM for congestion information propagation and resource deallocation.

Regarding switch scheduling, CCFIT uses iSlip [31], a Round-Robin (RR) algorithm achieving a fair arbitration inside the switch (as demonstrated in [12]). Specifically, all the switch input ports are served in a round robin fashion. An input port currently accessing an output port, will not get access to the same output port again until all other input ports requesting the same output port have been granted access. In particular, when a congestion tree arises inside the switch this scheduling policy allows a complete fairness between all the input ports which have allocated CFQs, even if several flows are sharing the same CFQ. Introducing injection throttling, however, may break the fairness property of RR if care is not taken. As shown in [12] it is important to use two thresholds (“high” and “low”) for congestion detection to maintain fairness in RR-based switches. CCFIT follows this approach, though comparing the thresholds against the fill ratio of the CFQs rather than the VOQs. This is further described in Section III-C.

Although RR scheduling combined with the use of two thresholds allows fairness between input ports, it does not achieve fairness between traffic flows if some flows are exclusive users of their CFQs while other flows are sharing a CFQ (the parking lot problem). This problem is, however, solved by introducing the injection throttling [12]. All in

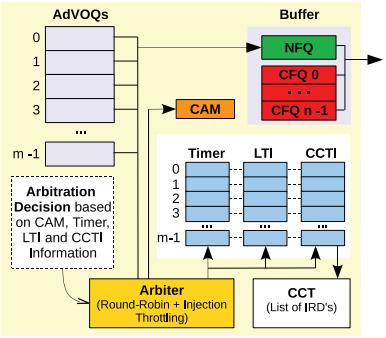


Figure 2. CCFIT Input Adapter Architecture.

all, CCFIT is able to achieve fairness at two levels: among different flows arriving at a hot spot switch from different input port CFQs, and by solving the parking lot problem. The fairness of CCFIT is further addressed in the evaluation Section IV-C.

Summing up, by using the architecture outlined above, the CCFIT switches are, as we describe in Section III-C, able to detect congestion, move output ports into the congestion state, marking packets contributing to congestion, isolating congested flows in the CFQs, and finally, release the required congestion information resources when congestion vanishes. The next section describes the end-nodes architecture.

B. End-nodes Architecture

An end-node receiving a packet with the FECN bit active should, as soon as possible, notify to the packet source about the congestion in order to throttle the injection. Similar to the IB CC mechanism, CCFIT returns a congestion notification packet (CNP) with the BECN bit active. The BECN packet has priority in the switches for being transmitted, and it only uses NFQs. Fig. 2 shows a diagram of the end-node architecture in charge of generating traffic, receiving BECNs and throttling the injection. For the sake of simplicity, we have omitted end-node structures in charge of receiving data packets and generating BECNs. In the following, we will refer this part of the end-node as the Input Adapter (IA).

Basically, CCFIT IAs have a fixed number of admittance queues (AdVOQs) equal to the network end-nodes, each AdVOQ$_i$ storing packets addressed to destination i, thus avoiding the HoL-blocking that may arise while generating traffic. Like the switch input ports, IAs have an output buffer organized in queues: one NFQ storing non-congested packets, and a small number of CFQs storing congested packets. Moreover, IA has a CAM with the same behavior as the ones located at switches. The CCFIT post-processing mechanism moves congested packets to the corresponding CFQ, so the HoL-blocking elimination is assured. Furthermore, CCFIT IAs include specific structures for the injection throttling, following an IB approach (see Section II). Specifically, the *Congestion Control Table* (CCT) stores a list of Injection Rate Delays (IRDs) which can be applied to any AdVOQ$_i$ in order to reduce its injection rate. The *CCT indexes array* (CCTI) stores a CCT index for each AdVOQ$_i$ and, by that, the IRD for a given AdVOQ$_i$ can be found at $CCT[CCTI[i]]$. Each CCTI index is increased when a BECN is received at the IA, thus increasing the value of the IRD applied to the AdVOQ$_i$. Notice that, during heavy congestion situations the IA will receive a lot of BECNs which will increase the CCTI index, and therefore the IRD value. In this way, the IA reduces the injection for the congested destinations.

For a given AdVOQ$_i$, the *Timer* array is used to decrease in one unit the CCTI[i] when the timer expires. In this way the IRD is reduced for that AdVOQ$_i$. A *Last Time of Injection* (LTI) array is in charge of storing the last time an AdVOQ$_i$ injected a packet in the network. This value is used by the arbitration together with the IRD in order to calculate if the next packet of the corresponding AdVOQ$_i$ could be sent or not. That is, the IA arbiter selects a packet from a specific AdVOQ$_i$ by applying a RR policy, making the "arbitration decision" based on the CAM, Timer, LTI and CCTI structures.

Finally, the most important effect of using CCFIT is achieved by the injection throttling since the CAM lines and CFQs, which were allocated when congestion appeared, are released quickly. As we show in the CCFIT evaluation (section IV) the network throughput is increased in comparison with the RECN-like and injection throttling techniques. In order to clarify the overall behavior of CCFIT at the IAs, an operation example is described in Section III-D.

C. Switch Behavior

Fig. 3 shows an example of CCFIT switches behavior. The switch stores an incoming packet (Event #1) in the corresponding NFQ. After that, CCFIT may detect congestion (Event #2) based on the NFQ occupancy level. When the NFQ fill ratio exceeds the congestion detection threshold, a congestion situation is detected and a CFQ is allocated in the input port, together with a CAM line containing the information related to the new congestion point.[3] Therefore, the CAM line information is used to detect if an incoming packet is addressed to the same destination. [4]

Like FBICM, CCFIT moves congested packets from the NFQ to the corresponding CFQ by means of the packet post-processing mechanism (Event #3). Basically, when a packet reaches the head of the NFQ, CCFIT looks up in the input port active CAM lines if the packet destination is stored in one of them. In the case of a match, the packet is moved to the CFQ the CAM line is referred to, otherwise, the packet crosses to the requested output port. Note, the post-processing mechanism leaves in the head of the NFQ only non-congested packets, thus avoiding the HoL-blocking problem between congested and non-congested packets. Moreover, this mechanism decides which input port queue (NFQ or CFQs) can request the output

[3]Notice that CCFIT, like FBICM, only requires to store in the CAM the destination the congested packet is addressed to.

[4]More information about FBICM CAMs can be found in [11], [32].

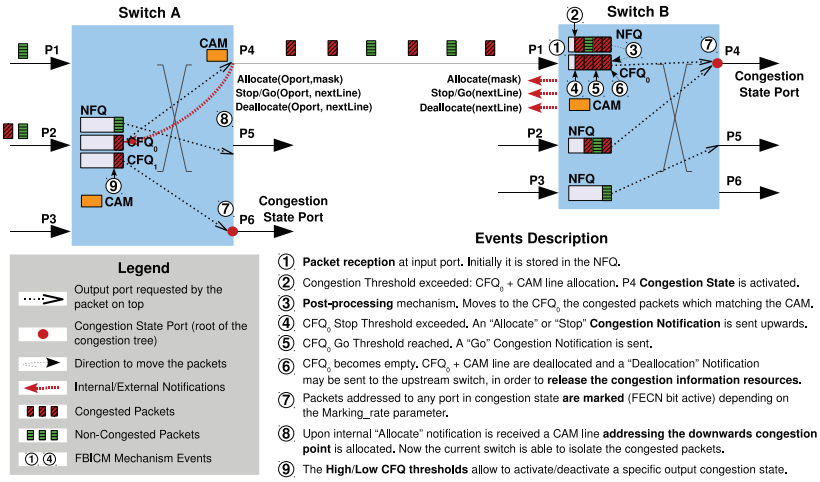


Figure 3. Example of the CCFIT Operation at Switches.

port. In that sense, it will create the crossing-requests that the arbiter will use for crossing packets to their requested output port. As we have described, the switch uses the iSlip scheduling algorithm.

CCFIT follows the FBICM scheme for propagating the congestion information. A CFQ Stop/Go flow control (Events #4 and #5) is used between every two switches containing allocated CFQs belonging to the same congestion tree. In this way, CCFIT separates congested and non-congested flows in different CFQs along any path followed by congested packets, thus isolating them and minimizing the HoL-blocking effect.

The dynamic and distributed resource deallocation process begins when a CFQ satisfies some conditions: it is empty and its associated CAM line is in Go status. When a CFQ is deallocated (Event #6), and it is part of one congestion tree branch (a previous "allocation" notification was sent to the upstream switch), a "deallocation" message is sent to the upstream switch to notify about the new situation. A similar process takes place between deallocated output port CAM lines and its linked input port CFQs+CAM lines.

As previously described, an important feature of the post-processing mechanism is that it is in charge of deciding if some output port should be moved into the congestion state. For each CFQ allocated in the root of the congestion tree (it is 1-hop away from the congested point), CCFIT looks at the CFQ occupancy level and, if that CFQ occupancy level exceeds the "High" threshold, the post-processing mechanism moves the output port, pointed to by the CFQ, into the congestion state. However, it may occur that the output port was already in the congestion state. When this happens CCFIT keeps track of the number of CFQs which have an occupancy level above the "High" threshold. Packets crossing an output port in the congestion state will be marked (Event #7) as congested (FECN bit set). If the CFQ occupancy level decreases below the "Low" threshold the output port counter is decreased, until it reaches the value 0, then moving the output port out of the congestion state. From this moment, no more packets are marked at the output port. Notice that, a CFQ which is placed 2-hops away from the congestion state (e.g. in Fig.3 CFQ_0 of P2 at Switch A) does not move its referred output into the congestion state, thus packets are not marked.

As it has been described in Section II, a FECN bit will be set or not depending on *Packet_Size* and *Marking_Rate* parameters. The influence of the *Packet_Size* and *Marking_Rate* parameters in the accuracy of the injection throttling mechanism is described in Section III-E.

D. Input Adapter Behavior

Fig. 4 shows an example of CCFIT IAs behavior. Basically, the IA is connected to switch A, which has just detected a congestion situation, which in turn, has been triggered by an incoming packet (Events #1 and #2). The post-processing mechanism moves all the congested packets to the CFQ (Event #3), and the CFQ flow control (Event #4) propagates the congestion information to the IA.

As the packets crossing through port P3 at Switch A are marked, the IA will receive BECN notifications indicating the injection throttling must start. When a BECN is received at the IA (Event #6), CCFIT obtains the $AdVOQ_i$ (i being the destination generating the BECN) which is sending packets to the destination this BECN belongs to. At this moment, CCFIT increases the CCTI[i] by one, thus increasing the IRD_i for the $AdVOQ_i$. Moreover, the Timer[i] (see Fig. 2) is initialized with the *CCTI_Timer* value. When the timer expires (Event #7) the CCTI[i] is decreased by one and the IRD_i is reduced, thus the IA increases the injection rate.

The IA arbiter makes the decision of which packet must be moved from the $AdVOQ_i$ to the NFQ (Event #8) based on a RR policy among all the AdVOQs. For each $AdVOQ_i$

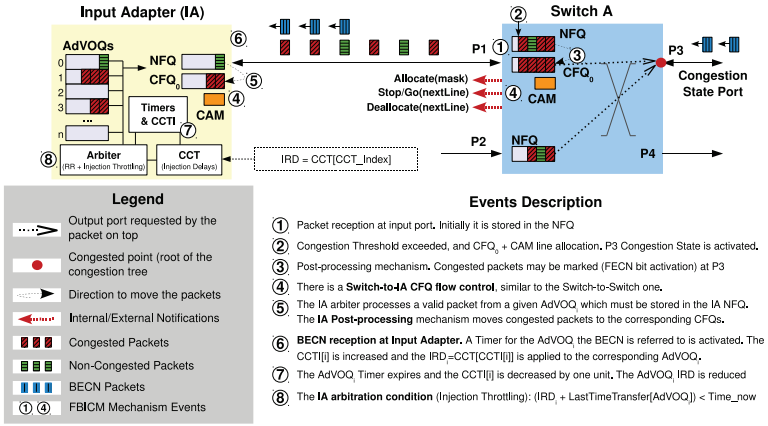


Figure 4. Example of the CCFIT operation at IAs.

the arbiter checks if the IRD_i value applied to that AdVOQ is greater than the current time, thus allowing the injection.

In this way CCFIT reduces the injection for congested destinations during congested situations, thus reducing the congestion trees and releasing the required resources for storing congestion information (mainly CFQs and CAMs) in a fast way. The injection rate is increased when congestion vanishes. As the evaluation shows (see Section IV), CCFIT significantly improves the overall network throughput achieved by RECN-like approaches, such as FBICM, and injection throttling ones.

E. Parameter Tuning Discussion

As it has been mentioned throughout the paper, CCFIT requires several parameters to be configured in the switches and IAs. These parameters are *Congestion detection threshold*, *CFQ stop/go thresholds*, *CFQ High/Low Thresholds*, *CCTI_Timer*, *Marking_Rate* and *Packet_Size*. We have made the same experiences as in [8] for the *CCTI_Timer*, *Marking_Rate* and *Packet_Size* parameters, thus a further description has been omitted. The remainder of the parameters need to be established taking into account the following ideas: The CFQ "High/Low" thresholds, as it is described in [12], should have a distance of at least one packet MTU. As one CFQ moves the corresponding output port into the congestion state when its occupancy level exceeds the "High" threshold, the "Stop" flow control threshold should be greater than the "High" one, in order to not block congested packets being transmitted from upward switch CFQs. On its side, the difference between "Stop" and "Go" thresholds needs to be sufficient for neither blocking too much upward congested flows, or allowing too much forwarding of congested traffic. Finally, the *detection threshold* value should allow to detect congestion not too early and not too late.

IV. Evaluation

In this Section CCFIT is evaluated in terms of network performance and fairness. It is important to note that the CCFIT main contribution is the significant good performance in comparison to FBICM, which is achieved especially in congestion situations where the latter has not a sufficient, available number of CFQs for storing congested packets. First of all, we describe the simulation tool, modeled traffic patterns and network configurations used in the experiments. Next, we analyze CCFIT performance results and fairness.

A. Simulation Model

The simulation tool used in our experiments is an event-driven simulator written in the programing language $C++$, which models interconnection networks at the cycle level, end-nodes and links. In our experiments, we model different network configurations which are shown in table I.

Table I
Evaluated Interconnection Network Configurations

	Config. #1	Config. #2	Config. #3
# Nodes	7	8	64
Topology	Ad-hoc (Fig. 5)	2-ary 3-tree (Fig. 6)	4-ary 3-tree
# Switches	2	12	48
Crossbar BW	5 GBytes/s	2.5 GBytes/s	
Switching	Virtual Cut-Through		
Scheduling	iSlip algorithm [31]		
Packet MTU	2048 Bytes		
Memory Size	64 KBytes		
Link Bandwidth	2.5, 5 GBytes/s	2.5 GBytes/s	
Flow Control	Credit-based		
Routing Algorithm	Deterministic	Deterministic (referred as DET [33])	
Routing Logic	Table-Based		

Config. #1 (Fig.5) and Config. #2 (Fig.6) are used to study throughput and fairness when several traffic flows create congestion. Config. #2 is furthermore used for studying congestion when uniform (random) traffic is generated. As congestion appears/disappears in a fast fashion, congested-flows need to be isolated immediately, while the fairness is maintained. A third configuration, Config. #3, is used for

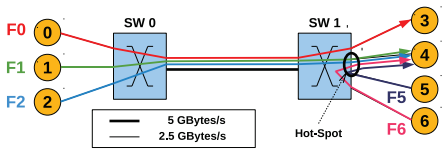


Figure 5. Configuration #1 Diagram.

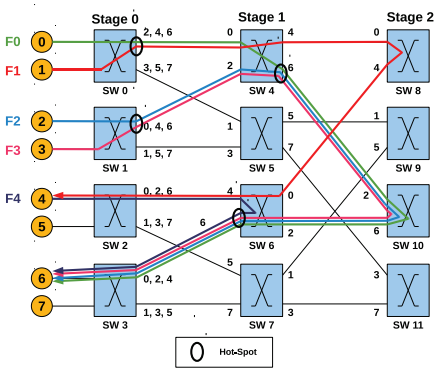


Figure 6. Configuration #2 Diagram.

testing CCFIT's reaction against heavy congestion situations, where several congestion trees overflows the number of available CFQs. In addition, the modeled traffic patterns for the above configurations are:

- **Case #1** (Config. #1). Five flows (F0, F1, F2, F5 and F6, see Fig. 5) are injected in the network. The injection rate is 100% of the link bandwidth (2.5 GBytes/s). Specifically, F0 (the victim flow) is active during the whole simulation period, while the other flows are activated in a sequential way. F1 is active during the time interval [$2ms$, $10ms$], F2 between [$4ms$, $10ms$], F5 between [$6ms$, $10ms$], and finally F5 is activated in the interval [$6ms$, $10ms$]. This traffic pattern generates a congestion point in the link connecting switch 1 with end-node 4.
- **Case #2** (Config. #2). Like the Case #1, five flows (F0-F4) are sequentially injected in the network at 100% of the link speed. In this case the flow F1 remains active during the whole simulation period. F0 is activated in the interval [$2ms$, $10ms$], F4 between [$4ms$, $10ms$], F2 between [$6ms$, $10ms$], and F3 in the interval [$6ms$, $10ms$]. This traffic pattern creates several congestion points in the network which divide the link bandwidth among all the flows contributing to congestion.
- **Case #3** (Config. #2). Here the situation is the same as for the Case #2, but three uniform traffic flows (sending packets to random destinations) are now active during the simulation (from nodes 5, 6 and 7). All the flows are injected at 100% of the link bandwidth. In particular this traffic pattern may add short lived congestion situations which quickly appear and disappear. Such congestion situations require a fast CC mechanism.
- **Case #4**. (Configuration #3) 75% of the sources inject uniform traffic at 100% rate of the link bandwidth. Suddenly, the remaining 25% of the sources generate congested traffic during the time interval [$1ms$,$2ms$]. These sources stop injecting traffic after this time period. This configuration tests if CCFIT copes with heavy congestion situations where more congestion trees than the number of CFQs are present. We have introduced 1, 4 and 6 congestion trees in the network during the congested traffic period of time.

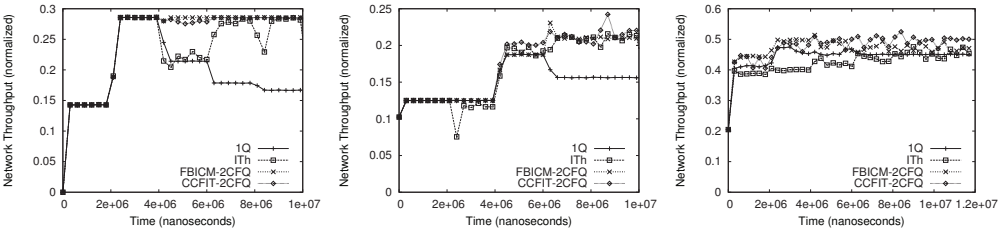
We have modeled an IQ-switch architecture, thus RAM memories are only added at each switch input port, with different sizes depending on the CC technique. Specifically, the simulator models the following CC techniques:

- Single Queue (1Q). This is the simplest case, with only one queue at each input port storing all the incoming packets. Hence, there is no HoL-blocking reduction policy at all. Note that this scheme is used for evaluating the performance of the DET routing algorithm "alone".
- FBICM. We use 2 CFQs per input port. Moreover, there are CAMs both at input and output ports.
- Injection Throttling (ITh). We have fixed the *CCTI_Timer* to 8000 ns, and the *Marking_Rate* to 85% of packets. We assume 8 Virtual Output Queues (VOQs) per input memory. High/Low thresholds are respectively set to 4 and 2 packets (2 MTUs as it is discussed in [12]).
- CCFIT. We assume 2 CFQs per input port. Like ITh technique, the same values for the *CCTI_Timer* and *Marking_Rate* are assumed. Only uses 2 CFQs per queue are defined for congested packets and entering output ports in the congestion state as a difference with ITh which has been configured with 8 VOQs. Moreover, the "Stop" threshold is established to 10 packets MTUs while the "Go" one is set to 4 MTUs.
- VOQnet. This scheme (theoretically the most effective one) requires greater memory sizes per input port, because each memory must be divided into as many queues as network end-nodes, and each queue requires a minimum size. Considering flow control restrictions, packet size, and link bandwidth and delay, we fix minimum queue size to 4 KB, which implies port memories of 256 KB for configuration #3 networks. This scheme is actually almost unfeasible, but it is used to show the theoretical maximum efficiency in HoL-blocking elimination.

Finally, although the simulator offers many metrics, we base our evaluation on two metrics: Flow Bandwidth, which shows the throughput achieved by each traffic flow, and network throughput, which shows network efficiency when normalized. In the following subsections we analyze, by means of these metrics, the obtained network performance.

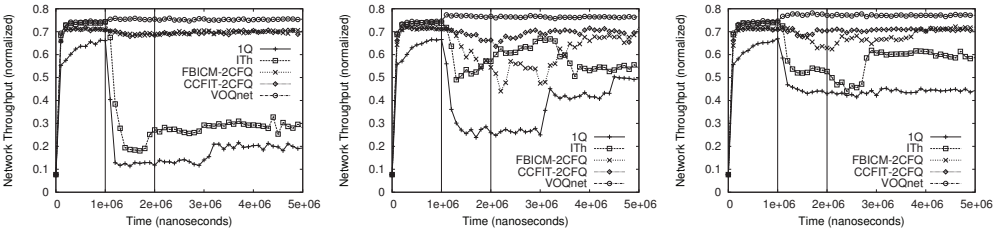
B. Throughput Results

Fig. 7 and 8 show the overall network throughput as a function of time for the network configuration #1, #2



(a) Configuration #1. Traffic Case #1 (b) Configuration #2. Traffic Case #2 (c) Configuration #2. Traffic Case #3

Figure 7. Throughput versus Time. Flow-Based Traffic Configuration



(a) 1 Congestion Tree. (b) 4 Congestion Trees. (c) 6 Congestion Trees.

Figure 8. Throughput versus Time (Configuration #3, Traffic Case #4).

and #3, using different traffic patterns. Considering all the plots, we clearly see how CCFIT outperforms the other CC techniques, even FBICM in some of the configurations. Specifically, in the plots shown in Fig. 7, the three CC techniques all show similar results, while 1Q struggles as soon as congestion is introduced in the network. In these scenarios the injected traffic is small, leaving the congestion less severe, making it rather easy for the different techniques to cope with the situation. In Fig. 7a ITh experience a drop in performance in the $[4ms, 6ms]$ time frame due to congestion detection at the left switch (Fig. 5). In configuration #2 case #3, Fig. 7c, we observe the tendency of ITh operating too slow as it takes time for this throughput to reach the level of the others.

In Fig. 8, we clearly see how CCFIT benefit from both doing congested-flow isolation and injection throttling as we increase the number of congestion trees from 1 to 4 and 6 in the network. In Fig. 8a, having one congestion tree in the network, CCFIT achieves excellent performance, at the level of FBICM. In this case FBICM, using 2 CFQs, have sufficient resources to isolate packets belonging to the single congestion tree. Notice that VOQnet, which achieves the maximum performance, has 64 queues per input port and uses 256 KB of memory, while the other CC techniques use less queues and uses less memory. The ITh scheme is not able to cope well with the situation. This could partly be caused by unfortunate CC parameter values for ITh, but then again finding optimal CC parameters for throttling is a challenging task [8], and is probably the main reason for CC not being in widespread use in IB networks today. Notice, however that the CCFIT is not as sensitive to the parameters (see in Fig. 8a), ITh reacts slow, even though the switches are VOQsw based, while CCFIT only has 1 NFQ and 2 CFQs per input port.

Fig. 8b shows the network throughput when 4 congestion trees are present in the network. Now FBICM struggles as it has not enough resources to isolate all the congested flows in a switch (FBICM uses only 2 CFQs per port): HoL-blocking is happening in the NFQs. CCFIT, on the other hand, shows a significant throughput improvement with respect to FBICM. The CCFIT injection throttling is able to release the resources used for isolating congestion flows (CFQs and CAMs) before the congestion-flow isolation part of the mechanism runs out of resources. Resources are released and made available to handle new congestion situations before the new situations arise. If we look at the ITh technique alone, it is in this scenario able to better cope with the congestion even though it still shows sign of instability and oscillation; the "saw-shape" effect. The 1Q scheme, which does not implement any HoL-blocking elimination mechanism at all, again achieves the worst results.

Finally, when 6 congestion trees appear in the network (Fig. 8c) we obtain similar results. This traffic pattern represents a situation where congested traffic is better balanced in the network. Again, CCFIT outperforms FBICM, while ITh needs more time to adjust the injection rate.

In conclusion, CCFIT significantly outperforms the other CC techniques, even FBICM, especially as the number of congestion trees grows, as the throttling part of CCFIT is able to release the resources used for congested flow isolation, preventing the congested-flow isolation part of the mechanism to run out of resources.

C. Fairness Study

Fig. 9 shows the throughput as a function of time for each individual traffic flow of config. #1 using traffic pattern case #1 (Fig. 5). In Fig. 9a, when no CC mechanism is present, not only does the throughput of the victim flow, F0, suffer from HoL-blocking, but the contributors to congestion also suffers from the parking lot problem as some contributors (the ones being the sole users of their input buffers) got more than their fair share of the link between switch 1 and end node 4. Enabling ITh, we not only improve the performance of the victim flow, but at the same time the parking lot problem is solved as the throttling happens at a per flow basis. A flow exploiting the parking lot problem will in return get more packets marked with a FECN (and then receive more BECNs), and then slow down. The improved fairness is clearly visible in Fig. 9b. Enabling FBICM, on the other hand, improves the throughput of the victim, even compared to ITh, but the parking lot problem prevails. The flows being the sole users of their respective CFQs still get more than their fair share of the bottleneck link towards end node 4. Actually, the unfairness has increased when using FBICM, as priority has been given to the victim flow addressed to the end-node 3.

Similar fairness study results, for network configuration #2, traffic case #2, are shown in Fig. 10. As can be seen in Fig. 6, there are now 4 congestion points, the parking lot problem possibly being present at two of them (switch 4 and switch 6). Again, both HoL-blocking and the parking plot problem leads to poor throughput and unfairness in the 1Q scenario (Fig. 10a), while the introduction of ITh improves performance in both aspects (Fig. 10b). FBICM improves the throughput even further, but again the unfairness in the network is dominant (Fig. 10c). Finally, Fig. 10d shows that both the best throughput and the highest degree of fairness is achieved by the CCFIT mechanism. It reacts fast due to the congested-flow isolation and improves fairness by reducing the injection rate on a per flow basis.

Summing up, besides efficiently eliminating HoL-blocking, CCFIT improves fairness in the network by solving the parking lot problem.

V. Conclusions

The link-level flow control of interconnection networks makes congestion spread from the oversubscribed link to the rest of the network. This has the adverse effect that flows that do not contribute to congestion will suffer from it. For this reason mechanisms that handle congestion are important.

The two classes of congestion control mechanisms previously described, attack the problem in different ways. Injection throttling approaches try to remove congestion by reducing the amount of traffic that is injected. On the other hand, congested-flow isolation methods lets congestion prevail, but confines the traffic that goes through the oversubscribed link to specially designated resources. This ensures that flows that do not contribute to congestion do not fall victim to it. These two approaches have different strengths and weaknesses.

In this paper we have presented three insights. Firstly, we have demonstrated that the weakness of injection throttling mechanisms is the reaction time of congestion events Secondly, we have shown that the weakness of congested-flow isolation is that it has limited scalability with respect to the number of congested points, and that it displays poor fairness between congested flows. Finally, and most importantly, we describe in detail CCFIT a mechanism that combines congested-flow isolation with injection throttling.

Our simulation results demonstrate that CCFIT extracts the best features of its two predecessors. In particular, injection throttling works in a way that limits the number of congested points that are alive in the network, and thus removes the scalability problem of congested-flow isolation. Furthermore, congested-flow isolation provides a quick and local response to congestion that removes the problems created by the slow reaction time of injection throttling. Finally, the good fairness properties of injection throttling are preserved in the combined method.

Acknowledgments

This work is jointly supported by the MEC, MICINN and European Commission under projects Consolider Ingenio-2010-CSD2006-00046 and TIN2009-14475-C04, and by the JCCM under projects PCC08-0078 (PhD. grant A08/048) and POII10-0289-3724.

References

[1] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks An Engineering Approach*, revised edition ed. Morgan Kaufmann, 2003.

[2] G. F. Pfister and V. A. Norton, ""Hot Spot" contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. 34, no. 10, pp. 943–948, 1985.

[3] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[4] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic evolution of congestion trees: Analysis and impact on switch architecture," *Proc. 1st HiPEAC Conf.*, pp. 266–285, November 2005.

[5] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*. ACM, 1988, pp. 314–329.

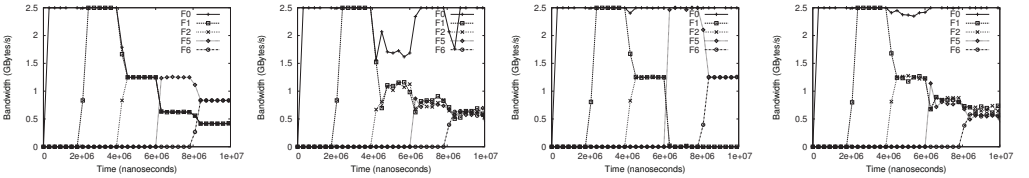
[6] L. S. Brakmo and L. L. Peterson, "TCP vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465–1480, 1995.

[7] C. Parsa and J. Garcia-Luna-Aceves, "Improving TCP congestion control over internets with heterogeneous transmission media," in *7th International Conferance on Network Protocols (ICNP99)*. IEEE Computer Society, 1999, pp. 213–221.

[8] E. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1–12.

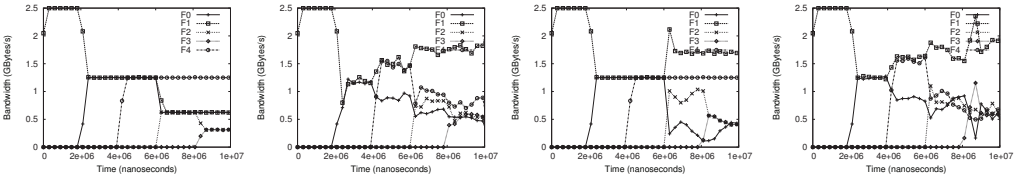
[9] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-end congestion control for InfiniBand," in *INFOCOM*, 2003.

[10] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *Proceedings of the 11th Symposium on High Performance Computer Architecture (HPCA)*, 2005.



(a) 1Q. (b) ITh. (c) FBICM 2-CFQ. (d) CCFIT 2-CFQ.

Figure 9. Flow Bandwidth versus Time (Configuration #1, Traffic Case #1).



(a) 1Q. (b) ITh. (c) FBICM 2-CFQ. (d) CCFIT 2-CFQ.

Figure 10. Flow Bandwidth versus Time (Configuration #2, Traffic Case #2).

[11] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "FBICM: Efficient congestion management for high-performance networks using distributed deterministic routing," in *LNCS Series - 15th Conference on High Performance Computing - (HiPC 2008), Bangalore, India*, December.

[12] E. G. Gran, E. Zahavi, S.-A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, "On the relation between congestion control, switch arbitration and fairness," in *International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*.

[13] M. Thottetodi, A. Lebeck, and S. Mukherjee, "Self-tuned congestion control for multiprocessor networks," in *Proc. of 7th. HPCA*, February 2001.

[14] J. Kim, Z. Liu, and A. Chien, "Compressionless routing: a framework for adaptive and fault-tolerant routing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 229 –244, Mar. 1997.

[15] E. Baydal and P. López, "A robust mecahnism for congestion control: Inc," in *Euro-Par*, 2003, pp. 958–968.

[16] *InfiniBand architecture specification. Release 1.2.1*, InfiniBand Trade Association, Nov. 2007.

[17] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification.*, IEEE 802.1Qau-2010 ed., IEEE 802 LAN/MAN Standards Committee, 2010. [Online]. Available: http://www.ieee802.org/1

[18] J. R. Santos, Y. Turner, and G. J. Janakiraman, "Evaluation of congestion detection mechanisms for InfiniBand switches," in *IEEE GLOBECOM – High-Speed Networks Symposium*, 2002.

[19] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Congestion management in MINs through marked and validated packets," in *PDP*, 2007, pp. 254–261.

[20] ——, "On the influence of the packet marking and injection control schemes in congestion management for MINs," in *Euro-Par*, 2008, pp. 930–939.

[21] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.

[22] W. Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *Proc. of 6th Hot Interconnects*, 1998, pp. 41–50.

[23] Y. Tamir and G. Frazier, "Dynamically-allocated multi-queue buffers for vlsi communication switches," *IEEE Transactions on Computers*, vol. 41, no. 6, June 1992.

[24] T. Nachiondo, J. Flich, and J. Duato, "Buffer management strategies to reduce hol blocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 739–753, 2010.

[25] W. Olesinski, H. Eberle, and N. Gura, "Scalable alternatives to virtual output queueing," in *Proc. IEEE International Conference on Communications*, 2009.

[26] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, and J. Duato, "An efficient strategy for reducing head-of-line blocking in fat-trees," in *LNCS Series. Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy*, september 2010, pp. 413–427.

[27] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Efficient, scalable congestion management for interconnection networks," *IEEE Micro*, vol. 26, no. 5, pp. 52–66, September 2006.

[28] G. Mora, P. J. García, J. Flich, and J. Duato, "RECN-IQ: A cost-effective input-queued switch architecture with congestion management," in *Proc. ICPP*, 2007.

[29] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *IEEE TRANSACTIONS ON COMMUNICATIONS*, 1996, pp. 296–302.

[30] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, March 2006.

[31] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[32] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "Cost-effective congestion management for interconnection networks using distributed deterministic routing," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010), Shanghai, China*, december 2010.

[33] C. Gomez, F. Gilabert, M. Gomez, P. Lopez, and J. Duato, "Deterministic versus adaptive routing in fat-trees," in *Workshop on Communication Architecture on Clusters, as a part of IPDPS'07*, March 2007, p. 235.

Paper VI

Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks

Jesús Escudero-Sahuquillo, Ernst Gunnar Gran, Pedro J. García,
José Flich, Tor Skeie, Olav Lysne, Francisco J. Quiles, and José Duato

Part I: Main Paper
Part II: Supplemental Document

Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks

Jesus Escudero-Sahuquillo, *Member, IEEE,* Ernst Gunnar Gran, *Member, IEEE,*
Pedro J. Garcia,José Flich, *Member, IEEE,* Tor Skeie, *Member, IEEE,*
Olav Lysne, *Member, IEEE,* Francisco J. Quiles, *Member, IEEE,* and José Duato

Abstract—Interconnection networks are key components in High-Performance Computing (HPC) systems, their performance having a strong influence on the overall system one. However, at high load, congestion and its negative effects (e.g. Head-of-line blocking) threaten the performance of the network, and so the one of the entire system. Congestion control (CC) is crucial to ensure an efficient utilization of the interconnection network during congestion situations, as one major trend is to reduce the effective wiring in interconnection networks to reduce cost and power consumption. This means that the network will operate very close to its capacity, thus congestion control becomes essential. Existing CC techniques can be divided into two general approaches. One is to throttle traffic injection at the sources that contribute to congestion, and the other is to isolate the congested traffic in specially designated resources. However, both approaches have different, but non-overlapping weaknesses: injection throttling techniques have a slow reaction against congestion, while isolating traffic in special resources may lead the system to run out of those resources. In this paper we propose EcoCC, a new Efficient and Cost-Effective CC technique, that combines injection throttling and congested-flow isolation to minimize their respective drawbacks and maximize overall system performance. This new strategy is suitable for current commercial switch architectures, where it could be implemented without requiring significant complexity. Experimental results, carried out by means of simulations under synthetic and trace-based traffic patterns, show that this technique improves by up to 55% over some of the most successful congestion control techniques.

Index Terms—Interconnection Networks; Congestion Control; Head-of-Line blocking; Injection Throttling

1 Introduction

High-speed, switch-based interconnection networks are nowadays essential components for different types of parallel-computing systems, from Networks-on-Chip to Massively Parallel Processors. In particular, in High-Performance Computing (HPC) systems, the performance offered by the interconnection network must mandatorily meet the high requirements of the applications supported by these systems; otherwise the network would become the system bottleneck and the processing nodes would be idle while waiting for new data to arrive, thereby wasting computational power. Thus, every aspect of the networks of HPC systems (topology, routing, etc.) should be designed bearing in mind the very high requirements of these networks.

Even thoroughly designed networks, however, may experience performance degradation during situations of congestion. Basically, congestion consists in intense traffic clogging a number of internal network paths, thereby slowing down traffic flows. Typical causes of congestion are hot-spots, network burstiness, re-routing around faulty regions, and conducting link frequency/voltage scaling (i.e. lowering the link speed in order to save power). Whatever the cause is, congestion originates when several packet flows simultaneously request access to the same output port in a switch, or if a destination node is not able to process packets at the rate they arrive. In networks where discarding blocked-packets is not allowed (like those currently used in HPC systems), any packet without granted access to a requested output port will remain blocked in a queue until its request is accepted. If this situation persists, the involved queues rapidly fill up as new packets arrive, then blocking packets in other switches due to the backpressure of the link-level flow control. Eventually, congestion spreads throughout the network from the disputed output port or destination node where it originates, usually referred to as the *congestion root*.

Note that congestion spreads not only across the paths followed by flows going through the congestion root (usually known as *hot flows*), but possibly also along the paths of *cold flows* (i.e. flows that are not headed for the congestion root). When a cold flow shares resources with a hot flow, the cold flow is slowed down or even blocked by the hot flow, so that eventually that cold flow contributes to congestion spreading towards its own origin. In this way congestion very well spreads all the way back to source nodes of not only hot flows, but back to source nodes of cold flows as well. The overall structure of blocked traffic created by all the hot flows (and possibly some cold ones), is in general known as a *congestion tree* [1].

Figure 1 shows how a congestion tree is formed by three hot flows, injected from sources A, B and D, all addressed to destination X. Notice that other flows, injected respectively from sources C and E, are addressed to other destinations (Z and Y, respectively). Thus, at the beginning of traffic injection, they do not contribute to congestion (i.e. they are cold flows). The port connecting Switch 8 to X has become the root of a congestion tree. The injection rates of the three hot flows are 33% of the link speed, due to the sharing of the bottleneck link between Switch 8 and X. This, however, also reduces the injection rates of the cold flows accordingly, as cold packets addressed to Y and Z share queues with hot ones at some ports (see the enlarged views of these queues in

- *J. Escudero-Sahuquillo, Pedro J. Garcia and Francisco J. Quiles are with the Computer Systems Department, University of Castilla-La Mancha, Campus Universitario, s/n 02071, Albacete, Spain. E-mail: {jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es*
- *Ernst G. Gran, Tor Skeie and Olav Lysne are with the Simula Research Laboratory, Martin Linges vei 17, Fornebu, Norway. E-mail: {ernstgr, tskeie, olav.lysne}@simula.no. Tor Skeie is also affiliated with the University of Oslo. E-mail:{tskeie}@ifi.uio.no*
- *J. Flich and J. Duato are with the Department of Computer Engineering (DISCA), Universitat Politècnica de València, Camí de Vera, s/n 46071, Valencia, Spain. E-mail: {jflich, jduato}@disca.upv.es*

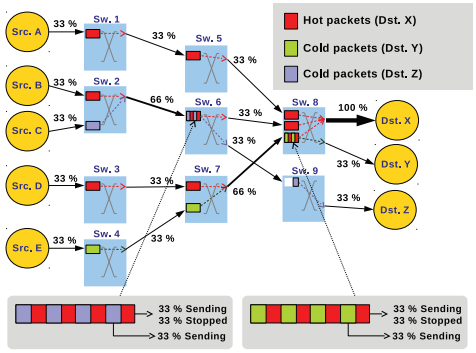


Fig. 1. Head-of-Line (HoL) blocking appearance

Figure 1). Eventually, cold flows end up advancing at the same reduced speed as hot flows.

This effect is a particular case of *Head-of-Line (HoL) blocking* that, in general, occurs when a blocked packet at the head of a queue prevents other packets in the same queue from advancing, even if these other packets request free resources (i.e. idle output ports). Indeed, hot flows may cause HoL-blocking to cold flows either at the switch where congestion originates (low-order HoL-blocking [2], e.g. that suffered by the flow addressed to Y at Switch 8) or at switches along branches of a congestion tree (high-order HoL-blocking [3], e.g. that suffered by the flow addressed to Z at Switch 6). In general, cold flows suffering HoL-blocking from hot flows are called *victim* flows. Congestion does not only leads to HoL-blocking. It may also lead to *buffer hogging* [4]. Indeed, when dynamically assigning buffer space to traffic flows, hot flows are likely to take most of the buffer space at each port they visit. This can leave the cold flows visiting the same ports with too little buffer space to maintain a smooth flowing, resulting in a performance degradation in the network, even if HoL blocking is not present.

All together, the problems derived from congestion are serious enough to significantly degrade network performance if no countermeasures are taken [1], [5], [6]. In this context, *Congestion Control* (CC) can be defined as any strategy focused on avoiding, reducing, or eliminating congestion and/or its negative impact on network performance. There exists a huge research body in this field (see Section 6 in supplementary file). However, not all the CC proposals are suitable for the interconnection networks of current HPC systems (e.g. packet discarding and network overdimensioning have become obsolete).

Nowadays, the two most popular approaches to CC in HPC systems are *injection throttling* and *queue-based flow-separation*. The first approach [5] relies on the switches to detect congestion, and then inform the sources that contribute to congestion that they must reduce their corresponding injection rates. By reducing the injection rates, the sources remove the congestion tree and by that the derived problems (HoL-blocking, buffer hogging, etc.). The second approach, i.e. separating flows into different queues at switch ports, tries to prevent the hot flows from interfering with the cold ones, thereby preventing the negative effects of congestion without removing the congestion trees. Many techniques follow this approach (see Section 6 in the supplementary file), but the most efficient ones are those that completely isolate hot flows in queues dynamically allocated upon congestion detection [7], [8], [9], [10], [11]. We will hereafter refer to these strategies as *Hot-Flow Dynamic Isolation* (HFDI) techniques.

Unfortunately, both injection-throttling-based and HFDI techniques have drawbacks. The main problem associated with throttling-based CC is the delay since congestion detection to final notification to the sources contributing to congestion. During this period, HoL-blocking and buffer hogging degrade network performance. Moreover, this delay in the notification process may also lead to oscillation, as the contributors constantly try to adjust their injection rates based on "old" information about the situation in the network.

On the other hand, the main problem of the HFDI techniques arises when the number of congestion trees present in the network exceeds the maximum number of queues assigned to hot-flow isolation. When this happens, congestion trees will grow inside the queues supposed to be used only by cold flows, and by this, reintroduce HoL-blocking and lead to performance degradation in the network. More details about the implementations of these two approaches can be found in Section 6 of the supplementary file, while their drawbacks are in-depth analyzed in Section 2.

In [12] we showed that, by combining the two aforementioned approaches, their respective weaknesses were overcome. The HFDI part of the technique quickly reacts to congestion, thereby overcoming the slow reaction of the throttling-based mechanism. On the other hand, in the case of several congestion trees, the injection throttling part removes the trees before the HFDI mechanism runs out of resources to isolate the hot flows. The resulting "hybrid" (i.e. combined) CC technique was called Combined Congested-Flow Isolation and Throttling (CCFIT), and we showed that it is able to remove HoL-blocking immediately, ensure scalability by removing the congestion trees, improve fairness in the network, and achieve higher throughput than injection throttling and HFDI as standalone CC mechanisms.

In this paper we significantly refine and optimize the hybrid CC approach to take advantage of features in current state-of-the-art HPC switch architectures, so that we can propose a CC technique that is even more efficient and resource friendly. Specifically, CCFIT was designed to prevent both the low- and high-order HoL-blocking that appears in congestion situations, but modern commercial HPC switches "naturally" prevent low-order HoL-blocking, as the *de facto* standard is to have several read ports to connect each switch input buffer separately to all the output buffers of the switch (i.e. the effect is similar to have *Virtual Output Queues* (*VOQs*) at switch level [13]). As a consequence, any CC technique suitable for modern commercial switches can focus on preventing only high-order HoL-blocking, thereby leveraging the resources and mechanisms devoted to deal with congestion. The new Efficient and cost-effective Congestion-Control technique (*EcoCC*) that we propose in this paper has been designed according to this key idea, so that (among other advantages) the available resources are used more efficiently to isolate (and throttle) hot flows, congestion is detected more accurately, and implementation complexity is reduced.

The rest of the paper is organized as follows: Section 2 focuses on the drawbacks of injection throttling and HFDI techniques. Section 3 contains a thorough

description of the new EcoCC technique. In Section 4 we evaluate this technique using simulation results obtained under different traffic conditions. In Section 5 some conclusions are drawn.

2 Problem Statement

From our point of view, a perfect solution to congestion problems in interconnection networks of HPC systems should fulfill a list of criteria:

1) *Effectiveness:* Flows that do not contribute to congestion (i.e. cold flows) on any link should not be affected by congestion in other places in the network (i.e. there should not exist victim flows).
2) *No adverse effects:* Overall behavior of the network should not suffer from having Congestion Control turned on, relative to having it turned off (e.g. the CC mechanism should not introduce out-of-order packet delivery, and it should in particular not cause any harm to the network performance when no congestion is present).
3) *Fairness:* All flows that do contribute to congestion on a link should get their fair share of the link bandwidth. Otherwise, the CC mechanism may introduce unfairness among hot flows [14]. Note that this criterion also implies that the *parking lot problem* [15] will be solved at the root of the congestion tree.
4) *Scalability:* The CC mechanism should be able to handle any number of congestion trees concurrently present in a port or in the network as a whole. Also, the mechanism should scale in terms of network size.
5) *Reaction time:* The CC mechanism should be fast-reacting, so that it can handle highly dynamic congestion patterns before network performance degradation becomes irreversible.

As mentioned above, the two main strands in research on congestion control in interconnection networks of HPC systems are the injection throttling approach and the HFDI one. Both of these have been shown to fulfill the effectiveness property, and the property that there should be no adverse effects. The doubts that have been associated with both of them are related to the observation that each of them struggle to support some of the requirements 3-5.

Specifically, the main weakness of the throttling approach, regardless of implementation details, is that the feedback loop from the congestion point back to the throttling sender is long and unpredictable. From the moment that congestion appears until it is alleviated, the congestion notification will have to travel a distance that is dependent on network size and on where the congestion is located. Furthermore, it will have a latency dependent on how many congested points it crosses. Thus, injection throttling is only efficient for congestion situations that have some longevity, and it has clear weaknesses related to the 5th requirement above.

Observing that the feedback loop of injection throttling may be too long, the only reasonable remedy is to look for methods whose effective reaction is taken close to the congestion root. As mentioned above, HFDI techniques are a family of methods that do this, as the different variants of HFDI provide local, fast reaction to congestion (i.e. they meet the 5th requirement in the above list). On the other hand, it is clear that any implementation of HFDI will have to decide a priori on the number of concurrent congestion trees it can handle. The reason for this is that this number is bound to the maximum number of special queues per port that are configured to store hot flows. Even though it can be argued that it is unlikely that many congestion trees are present in several ports at the same time, and thus unlikely that more than a small fraction of ports would run out of special queues at any given time, just one or a few congestion trees growing outside of the special queues may be enough to spoil network performance dramatically. HFDI is thus not able to satisfy the 4th requirement. Furthermore, HFDI based solutions are not able to assure fairness between the contributors to congestion and by that solve "the parking lot problem" (see 3rd requirement in the above list). The hot flows are simply moved from the ordinary queues over to the special queues.

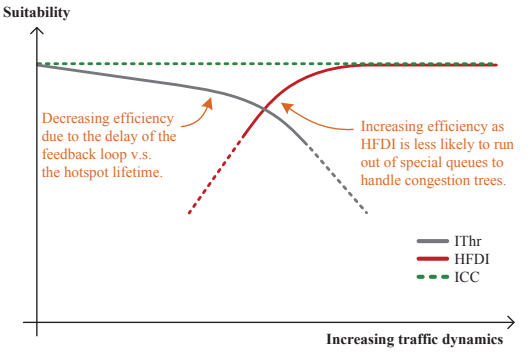


Fig. 2. An abstract view of the suitability of different congestion control approaches.

The suitability of the two families of CC mechanisms can also be seen in relation to the traffic dynamics in the network. A throttling-based CC mechanism is able to handle all kinds of congestion trees, including *moving congestion trees* [16] with some longevity quite well, while the mechanism struggles as the lifetime of the hot-spots decreases or the congestion trees move faster. That is, the efficiency of a throttling-based CC mechanism decreases as the traffic dynamics in the network increase. An HFDI mechanism, on the contrary, could actually be challenged by stable hot-spots as such hot-spots grow long-lived silent or windy congestion trees. Keep in mind that an HFDI mechanism does not remove the congestion trees. Even a small set of stable hot-spots could then by their corresponding congestion trees permanently occupy most, or in the worst case all, of the special queues in a large part of the network, leaving less or no room for additional congestion trees. When that happens, HoL-blocking is reintroduced.

Figure 2 gives an abstract view of the suitability of the two main CC approaches, injection throttling (IThr) and HFDI, as a function of the traffic dynamics in the network. In addition, the behavior of an ideal CC (ICC) mechanism is plotted. Even though the exact look of such a figure obviously varies depending on the number of hot-spots present, the traffic pattern, the topology and so on, it still gives an intuitive idea about the shortcomings of the two main CC approaches, compared to how we would like an ideal CC mechanism to behave.

Summing up, it seems that the two main trends in CC for HPC systems do not fulfill the list of requirements previously stated. Note, however, that their respective drawbacks are unrelated with each other, while their

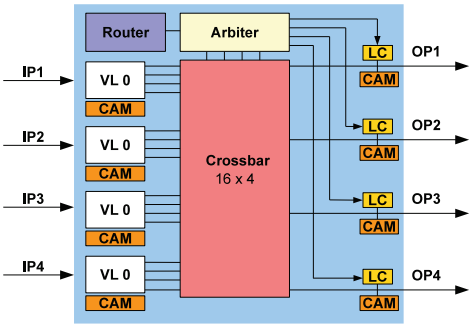


Fig. 3. Assumed Switch Architecture.

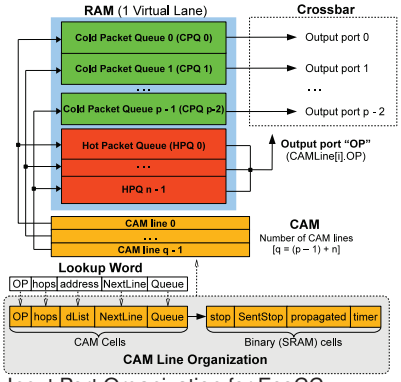


Fig. 4. Input Port Organization for EcoCC.

advantages are complementary as they jointly meet all the stated requirements. Hence, the idea of an approach combining both philosophies is obviously appealing. Next, we show in detail how these two approaches can be combined in a switch architecture that leverages all the features of modern interconnect technologies, then obtaining an approach to congestion control that satisfies all of the criteria listed at the beginning of this section.

3 EcoCC Description

In the following subsections we describe the EcoCC (Efficient and cost-effective Congestion Control) technique as we detail the architecture of a switch and an end-node supporting it.

3.1 The Switch Architecture

Figure 3 shows a schematic view of a switch architecture implementing EcoCC, while Figure 4 shows in detail the organization of each input port of the switch. For the sake of clarity, the switch in Figure 3 is shown as an "unfolded", unidirectional switch having four input ports and four output ports, where each port supports a single virtual lane (VL). Note, however, that the EcoCC technique is valid for bidirectional switches and by itself it does not impose any restrictions regarding the number of ports or VLs supported by a switch. We will refer to these figures in the following subsections.

3.1.1 Low-Order HoL-blocking Prevention

State-of-the-art switches typically utilize a VOQ-based switch architecture, where each input port administers a buffer divided into several sections, one section per possible output port. The input ports in Figures 3 and 4 implement such a VOQ scheme by providing a private read port into the switch crossbar for each buffer section corresponding to a given output port (i.e. the buffer at an input port has one buffer section and a corresponding read port per output port). Hereafter, a single instance of such a buffer section with its corresponding read port will be referred to as a VOQ. As soon as the part of a packet header that contains the destination address (or LID) is received at an input port, the routing decision is made by the Router component (assuming Virtual Cut-Through switching), and the packet is mapped to the VOQ corresponding to the next output port for which the packet is headed. At this queue the packet will be held waiting to be forwarded through the switch, i.e. through the crossbar. When arbitrating[1], the multiple read ports allow the arbiter to arrange concurrent transfers from multiple VOQs of the same input port in parallel through the crossbar, as the transfers are not headed for the same output port. On the other hand, it is worth pointing out that this architecture differs from the one assumed by CCFIT, which lacks VOQ support.

Note that this architecture by its own features completely removes the low-order HoL-blocking. Packets waiting for arbitration due to a busy output port will never share a VOQ with packets requesting a free output port, i.e. HoL-blocking due to local congestion (low-order HoL-blocking) is not possible. High-order HoL-blocking, on the contrary, is left untouched though, as two packets headed for the same local output port at a switch may share a VOQ even if they later, at a downstream switch, will be considered to belong to two different flows, one being hot, the other one being cold.

3.1.2 High-Order HoL-blocking Prevention

To avoid high-order HoL-blocking, EcoCC utilizes a combination of HFDI and injection throttling similar to CCFIT, but with several improvements and optimizations, as CCFIT considers neither several read ports nor $VOQs$. In EcoCC, each VOQ, shown as a green queue in Figure 4, initially functions as a Cold-Packet Queue (CPQ). That is, in a non-congested scenario, all incoming packets are stored in the CPQ corresponding to the next output port the packet is headed for. Apart from the CPQs, EcoCC also uses another type of queues at each input buffer, called Hot-Packet Queues (HPQs), which store only hot-packets, as explained later.

Each CPQ has two congestion-detection thresholds, one lower and one higher bound, LTh and HTh, respectively. When a CPQ is filled above the HTh threshold, congestion is detected and the corresponding output port is considered to be congested as long as the fill ratio of the CPQ remains above the LTh. As explained in [14], the use of two thresholds is also important for the assurance of fairness among hot flows. Note that in contrast with CCFIT that mixes in the same CPQ packets headed for different output ports, the detection of congestion in EcoCC is highly accurate, as all the packets stored in a specific CPQ which has exceeded the HTh threshold are headed for the same output port. Thus, this port is for sure a congestion root (i.e. all the packets in a CPQ detecting congestion are for sure hot packets). Thus, the

1. We assume the use of an efficient and fair scheduling algorithm like iSLIP [17] to organize and grant the crossing requests.

VOQ scheme removes the uncertainty of congestion root detection in CCFIT.

When congestion is detected in a CPQ, the corresponding output port enters into the *congestion state* and starts to mark packets contributing to congestion, by following the InfiniBand CC scheme[2]. If the output port was already in the congestion state, the output port only increases a counter to keep track of how many CPQs are contributing to congestion. Anyway, the injection-throttling part of EcoCC has now been activated. In addition, the HFDI part of EcoCC is also activated as a CAM line[3] (Figure 4) is created to store information that uniquely identifies the newly discovered congestion tree. Specifically, the new CAM line contains this information: the output port that is the root of the congestion tree (op), number of hops to reach op ($hops$, initially 1 as this port is inside this switch), and the number of the CPQ/HPQ ($Queue$) that CAM line is referred to. A few more bits for control information are used in every CAM line: the $stop$ bit (only active in upstream CAM lines) indicates that the associated HPQ can forward packets, the $sentStop$ bit says that the CAM line has sent a $stop$ to the upstream switch, the $propagated$ bit shows that the CAM line information has been propagated upstream, and the $timer$ sets the life time of a CAM line. Further details of the use of those control bits are given in [19].

At its allocation, the CAM line information is propagated to the upstream switch by means of an *allocate* congestion notification containing the location of the congestion tree root. That notification includes the $dList$ field: a list of destinations obtained from the packets crossing through the congestion root, that has been just detected; the remainder information being a copy from the source CAM line. Note that, at the switch where the congestion root is located, no HPQ needs to be allocated to isolate hot packets headed for that congestion root (i.e. to prevent low-order HoL-blocking), as they are already isolated in the CPQ. This is a further optimization over CCFIT, where a special queue needs to be allocated for each input port contributing to a congestion tree, even when the root is local. Thus, EcoCC uses HPQs exclusively to prevent the high-order HoL-blocking.

Initially, when an upstream switch receives an allocate notification from a CAM line, this information is stored in a CAM line at its output port, op, connected to the input port sending that notification. Then, as CPQs of the upstream switch request access to op, action has to be taken to avoid high-order HoL-blocking: The first time that a packet in a given input CPQ$_x$ request to use op and has a destination address that matches the $dList$ of a CAM line at op, the CAM line information is propagated internally to the input port where CPQ$_x$ is, and a HPQ$_x$ + CAM line (with local-switch op, $hops$, $dList$, $NextLine$ storing the CAM line number in the next switch and $Queue$, pointing to HPQ$_x$) are allocated there. When an HPQ$_x$ is assigned, a post-processing procedure will move all packets, whose destination address is included in the $dList$ of the newly allocated CAM line, from the CPQ$_x$ to HPQ$_x$[4]. For that purpose, any packet arriving at the head of the CPQ$_x$ will be compared against the CAM information: if there is a match, the packet is stored in the HPQ$_x$; otherwise it is forwarded to its corresponding output port. In this way, all hot packets associated with the congestion tree (i.e. "matching" a given CAM line) are isolated in the HPQ$_x$, and high-order HoL-blocking is prevented.

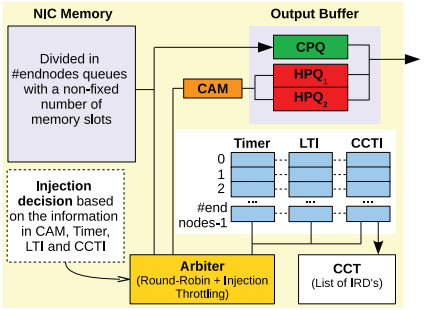


Fig. 5. EcoCC Host Channel Adapter (HCA) organization.

In HPQs, we use the same LTh and HTh thresholds that are used in CPQs, so that the first time an HPQ is filled above the HTh threshold, an *allocate* notification is sent to the upstream switch to allocate a CAM line, and so on. Thus, "consecutive" HPQs are allocated in consecutive switches to store packets from the same congestion tree. To avoid oversubscription of HPQs, a *Stop & Go* flow-control is implemented between consecutive HPQs, based also on the *LTh* and *HTh* thresholds (i.e. *stop* notifications are sent the second time the *HTh* threshold is reached and further ones), and *go* notification is sent the *LTh* threshold is reached. The *LTh* threshold is also used to deactivate the marking of packets (i.e. the injection-throttling part of EcoCC).

Finally, if an HPQ is completely emptied, this HPQ and the corresponding CAM line are deallocated and the corresponding upstream switch is notified about the new situation. In particular this indicates that the same CAM line can be removed from the output port of the upstream switch. Thus, EcoCC, like CCFIT implements a distributed, dynamic policy for releasing resources devoted to CC. Further details about memory (RAM and CAM) requirements of EcoCC can be found in Section 8 of the supplementary file.

Note that the use of only two thresholds, LTh and HTh, is a further optimization of EcoCC over CCFIT. In CCFIT five different thresholds are needed to keep track of congestion detection for the injection-throttling and HFDI parts of the technique, as well as for managing the flow-control between consecutive HPQs.

3.2 The End Node Architecture

As previously described, the injection throttling part of EcoCC is based on the InfiniBand CC: an output port in the congestion state marks packets, by activating the FECN (Forward Explicit Congestion Notification) bit at their headers. When a FECN-marked packet is received at an endnode, it generates a Backward Explicit Congestion Notification (BECN) addressed to the source endnode of the received FECN-marked packet. Source

2. A packet is marked as contributing to congestion according to the parameters *Packet_Size* and *Marking_Rate*, and if the output port is considered to be at the root of the congestion tree, i.e. the output port has free credits [18].

3. The CAM line is implemented partially by means of a Content Addressable Memory (CAM cells), hence the name.

4. This action is implemented by rearranging memory pointers for the involved packets, without actually "moving" data inside the RAM.

endnodes are in charge of applying the injection throttling policy, as defined by the InfiniBand specification. Figure 5 shows the organization of a Host Channel Adapter (HCA) implementing EcoCC.

Specifically, the organization and behavior of the endnode HCA is similar to the behavior of the Input Adapters (IAs) used for CCFIT [12]. At the heart of the EcoCC HCA is the Congestion Control Table (CCT) holding preconfigured increasing values of Injection Rate Delay (IRD). An IRD value indicates the delay in the injection of two consecutive packets from the same traffic flow. Each possible destination of a flow has three associated registers: Timer, Last Time of Injection (LTI) and CCTI_Index (CCTI). Each time a congestion notification message (BECN) associated with a flow with destination D is received, the CCTI associated with D is increased, so that it points to an entry in the CCT with a higher value of the IRD, thus the injection rate of that flow is decreased, accordingly. The CCTI, and by that the corresponding IRD are decreased at regular intervals given by the *Timer*. In this way, traffic flows have their injection rates decreased/increased in response to congestion in the network in a similar way to InfiniBand CC.

Each HCA defines a Network Interface Controller (NIC) memory divided into as many generation queues as endnodes in the network, so that each generated message is stored in a queue$\#i$, i being the LID of its destination. Thus, the HoL-blocking is prevented at traffic generation. Besides, each HCA has its own CPQ and a set of HPQs managed by CAM lines in the same way as for switches. Packets are transferred from the generation queues to the CPQ. The purpose of these CAM lines and HPQs is to avoid any HoL-blocking at the HCA if the injection-throttling mechanism is not able to remove a congestion tree before it reaches the HCA. In turn, the HCA Arbiter conducts arbitration taking into account the information in both the CAM and the CCT. Specifically, a packet from a flow with destination D is only eligible for arbitration if the time since the last packet injection of that flow (LTI[D]), is greater than the IRD (i.e. CCT[CCTI[D]]), and there exists no blocked CAM-line (i.e. with the $stop$ bit active) with a $dList$ including D.

3.3 Operation Example

Figure 6 shows how EcoCC deals with a congestion situation in a portion of an interconnection network composed of two switches (A and B) and two endnodes-HCAs (#0 and #3). In the top part of the Figure, EcoCC detects that congestion situation at the input port 1 (IP1) of Switch B (Event #1), when the HTh is exceeded at CPQ_1. Thus, the output port 1 (OP1) of Switch B enters into congestion state and an Allocate notification is sent from IP1 to Switch A (Event #2), in order to propagate the congestion-root information. From this moment, packets will be FECN-marked at OP1 of Switch B according to the EcoCC *Marking_Rate*. Besides, at OP1 of Switch A, when a hot-packet addressed to the congestion root is received (Event #3) (i.e. a packet which matches the CAM line information of that congestion root), an Allocate notification is sent (Event #4) to the input port sending that packet (IP1 at Switch A). The post-processing mechanism (Event #5) moves the packets addressed to the congestion root from the CPQ_1 to HPQ_1 at Switch A. Note that low-order HoL-blocking is prevented at Switch B, as there is one CPQ per output port, and high-order HoL-blocking is prevented at Switch A, as hot-packets are isolated in HPQs. Note also that, FECN-marked packets are received at the destination HCA #3 (Event #6), so that BECN notifications are sent from that endnode to the source HCA #0 (Event #7).

In the bottom part of Figure 6 when HPQ_1 at IP1 of Switch A fills up, it sends an Allocate notification to HCA #0 (Event #8) that allocates an HPQ to isolate hot packets. Besides, when a BECN notification is received at source endnode-HCA #0 (Event #9), EcoCC increases the CCTI, and so the IRD, for the flow with destination #3 in order to adjust its injection rate (see Section 3.2). Thus, the arbitration decision (Event #10) is made for those flows with small IRD. Note that the post-processing mechanism also happens in the HCA (Event #11).

Summing up, EcoCC ensures immediate reaction to congestion by locally assigning HPQs to hot flows (HFDI part), while the injection throttling (ITh) part removes the congestion trees to prevent EcoCC from running out of HPQs. Note that the ITh part of EcoCC and the assignment of HPQs at each switch are totally independent of each other, even if they share the congestion-detection mechanism. Thus, in the rare case that EcoCC runs out of HPQs, the ITh part still removes the congestion trees.

3.4 Advantages of EcoCC over CCFIT

Although throughout the previous sections we have indicated the differences between EcoCC and CCFIT, in the following points these novelties are summarized, pointing out their advantages:

- In order to be suitable for current commercial switch designs, EcoCC is based on a switch architecture with VOQ support, thus low-order HoL-blocking is completely eliminated without need for allocation of HPQs at the congestion root switch, in contrast with CCFIT that requires this allocation. This leaves more HPQs available to EcoCC for dealing with high-order HoL-blocking. Note also that, if CCFIT is used, some temporary low-order HoL-blocking may occur since congestion appearance until "local" HPQ allocation upon congestion detection, while this is not possible for EcoCC.
- The VOQ support also guarantees that the congestion-detection criterion (i.e. the identification of the location of a congestion root) is absolutely exact for EcoCC, as the root is for sure the output port associated with a filled-up VOQ (i.e. CPQ). By contrast, CCFIT introduces some degree of inaccuracy since the congestion root is detected as the output port requested by the packet at the head of a filled-up queue where packets requesting different output ports may be stored.
- The number of thresholds required by the whole technique is lower in EcoCC (2 against 5 in CCFIT), thereby reducing implementation complexity.

4 Performance Evaluation

In this section we evaluate EcoCC in comparison to other congestion control (CC) techniques. First, we describe the simulation model. Next, we show and discuss the results obtained from the simulation experiments.

4.1 Simulation model

We base the EcoCC evaluation in a set of experiments carried out by means of a custom-made, event-driven simulation tool written in the C++ language. Specifically,

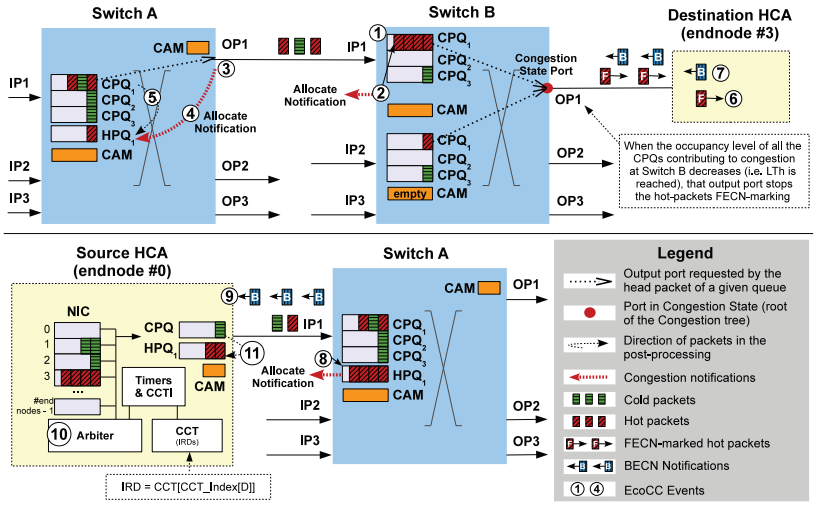


Fig. 6. EcoCC Operation Example.

the simulation tool models, with cycle-level accuracy (i.e. each event takes X cycles to finish, X being an integer), several types of interconnection networks by defining their topology, the routing algorithm, the switching and flow-control policies, the switch architecture, and the characteristics of the endnodes and links.

On the other hand, the simulation tool also models some details of the InfiniBand™ specification and other well-known features included in commercial solutions such as the usage of several read ports at input buffers and the demultiplexed crossbars. Indeed, the simulation tool has been validated against real, small-size InfiniBand-based systems like those evaluated in [20].

Table 1 shows the network configurations that we assume for our experiments. Specifically, we have modeled several fat-tree patterns based on the k-ary n-tree scheme. Fat-trees are a very popular network topology for HPC systems. Note that, although EcoCC is valid for other topologies, fat tree are among the most common ones in HPC systems. For all the network configurations, we assume serial full-duplex pipelined links with 5 GByte/s (40 Gbps) of bandwidth and 6 nanoseconds of link propagation delay. We have assumed links with a length of 1.2 meters and a delay of 5 ns/meter (based on the InfiniBand specification [18]), both for switch-to-switch and endnode-to-switch links.

TABLE 1
Evaluated BMIN configurations.

#	Network Size	Topology	#Switches	#Stages
1	64 × 64	4-ary 3-tree	48	3
2	256 × 256	4-ary 4-tree	256	4
3	1024 × 1024	4-ary 5-tree	1280	5

Regarding the switch architecture, we have modeled the one described in Section 3.1, including switch input buffers with several read-ports, in particular, as many as there are output ports in the switch. We assume that switches have 8-bidirectional ports (e.g. the Mellanox™'s non-blocking switch IS5022). The routing algorithm is DESTRO [21]. The basic flow control policy is credit-based, and it is performed at virtual lane (VL) level. The packet size is 2048 bytes. The simulator models several Congestion Control techniques:

- **Single VL** (1VL). This is the simplest case, with only 1 Virtual Lane (VL) at each input port, and no specific CC support. Note that the switch architecture natively prevents the low-order HoL-blocking, but this scheme cannot prevent the high-order HoL-blocking, thus it is considered just to show the worst reaction of the network when congestion arises. The size of the buffer at input ports is 128 KB of RAM.
- **IB-ITh**. This is the congestion control mechanism included in the InfiniBand specification [14], [20]. It has been modeled in order to show the performance of a CC technique based only on injection throttling which is implemented in real hardware. It is also assumed a 128 KB RAM per input port. Furthermore, we have set the *CCTI_Timer* and *Marking_Rate* according to the analysis of [20].
- **VOQnet**. This scheme has been configured with as many VLs as endnodes are in the network, each one of them storing packets addressed to the same, single destination. Theoretically, this is the most effective scheme, since it completely prevents the high-order HoL-blocking, but it requires a prohibitive memory space per input port, since each VL requires a minimum space. Specifically, if we consider flow-control restrictions and packet size, VOQnet requires 8 MB of RAM per input buffer for network configuration #1, 32 MB for network configuration #2 and 128 MB for network configuration #3[5].
- **EcoCC**. It is modeled as explained in section 3. We assume 1 VL and 128 KB of RAM per input buffer. In addition, 2 HPQs are configured per input buffer, along with a small CAM to control them. As we have 7 CPQs and 2 HPQs per input buffer, we will require 9 CAM lines per port, which are configured with a limited number of destinations in the $dList$ field (8 destinations for network configuration #1, 16 destinations for network configuration #2 and 32 destinations for network configuration #3). The High threshold (HTh) is established in 6 packets, the Low threshold (LTh) is set to 4 packets (see Sec-

5. These values have been calculated assuming 64 packets per VL and a packet size of 2 KB.

TABLE 2
Synthetic traffic patterns.

	Uniform Traffic		Hot-Spot Traffic			
#	%Srcs	Gen. Rate	%Srcs.	Gen. Rate	Gen. Interval	#Roots
1	87.5%	100%	12.5%	100%	1000−2000 μs	1
2	75%	100%	25%	100%	1000−2000 μs	4
3	62.5%	100%	37.5%	100%	1000−2000 μs	6

tion 7 in the supplementary file for further details about the tuning of these parameters). Finally, the *CCTI_Timer* and *Marking_Rate* have been set to the same values as the IB-ITh technique.

- **HDFI** (Hot-Flow Dynamic Isolation). It is the HFDI part of EcoCC, i.e. EcoCC without injection throttling. Actually, it is an adaptation of the FBICM technique (see Section 6 of the supplementary file) to the switch architecture described in Section 3.1.

Endnodes are connected to switches by means of Host Channel Adapters (HCAs), each one modeled with a NIC memory organized as explained in Section 3.2, i.e. divided into generation queues. Messages are packetized before being transferred to HCA injection queues, which are organized with the same queue scheme as that established by the different techniques for the switch ports. Note that, for EcoCC and HFDI, each HCA has only 1 injection CPQ (see Figure 5) regardless of the number of CPQs at switch input ports, while the number of HCA HPQs is the same as in switch ports (2 HPQs).

Regarding traffic patterns, we use both synthetic traffic modeling ideal traffic scenarios and real-traffic traces from benchmarks used to measure the performance of HPC systems. Synthetic traffic patterns (see Table 2) allow us to evaluate the impact of sudden congestion trees on the network performance. Specifically, for each traffic case, a percentage of sources constantly generate traffic with a uniform (random) distribution of destinations, while the remaining sources generate traffic addressed to 1, 4 or 6 hot-spot destinations only during a time interval [$1ms$,$2ms$]. Note that, in traffic case #1 a single congestion tree is generated, while in #2 and #3, 4 and 6 congestion trees are generated, respectively. These traffic patterns are used for testing how EcoCC copes with heavy congestion situations where there are more congestion trees than the number of HPQs per port.

In addition, we have used real-traffic traces obtained from several tests of the HPC Challenge benchmark [22], by means of the Extrae tool v2.2.0 [23]. The collective operations have been modeled as regular MPI point-to-point communication messages. The specific HPCC tests that we have run independently for extracting real-traffic traces are indicated in Section 4.3.

Finally, although the simulation tool offers many, different metrics for analyzing the behavior of modeled networks architectures, we base our evaluation on the network throughput (normalized), the normalized execution time of traffic traces and, for EcoCC and HFDI the number of switch ports that run out of HPQs (see Section 9.1 in supplementary file).

4.2 Hot-Spot Traffic Results

Figure 7 shows normalized network throughput as a function of time for all the network configurations of Table 1 when traffic patterns #1, #2 and #3 of Table 2 are used. Note that we vary the network size and the number of congestion trees generated in the network.

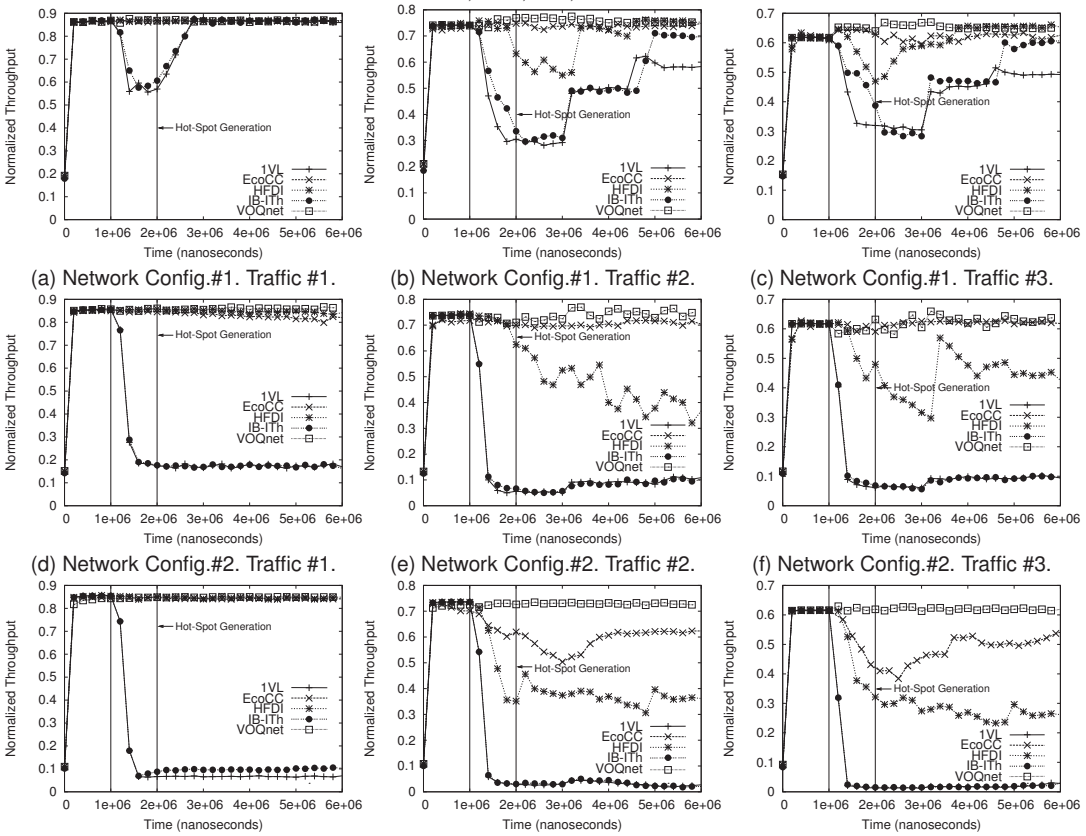
In the Figures 7a, 7d and 7g, (i.e. when only one hot-spot suddenly appears in the time interval [$1ms$,$2ms$]), the EcoCC and HFDI performance are close to that of VOQnet, which achieves the maximum performance, but requires as many VLs per input port as destinations in the network. EcoCC and HFDI achieve similar, good performance with only 2 HPQs because they are enough to isolate hot packets belonging to the single congestion tree, thereby preventing the high-order HoL-blocking. Note that this behavior is independent of the network size. The IB-ITh scheme is not able to cope with the congestion situation, because the $VOQs$ associated with the congestion root (at different input ports of the root switch) quickly grow in size as they receive many hot packets. Then, those $VOQs$ initially hog the buffer space of their respective input ports and, due to the back-pressure of the flow-control mechanism, congestion is propagated to the switches forwarding packets which contribute to congestion. Note that although the low-order HoL-blocking is prevented in all the modeled techniques due to the use of $VOQs$, it may happen that a full VOQ hogs the buffer space, thereby struggling the space of other $VOQs$. Besides, the congestion propagation is faster than the injection throttling reaction, thereby the IB-ITh technique being affected by the high-order HoL-blocking, especially when the network size increases. On the other hand, the Single Queue (1VL) scheme, as expected, performs the worst as it is affected strongly by the high-order HoL-blocking.

Figures 7b, 7e and 7h show network throughput as a function of time when the traffic pattern #2 is used (i.e. 4 congestion trees are generated during the time interval [$1ms$,$2ms$]) for network configurations of Table 1. In general, HFDI suffers a significant performance degradation as it has not enough HPQs to isolate all the different congestion trees that may appear in a switch port (HFDI has only 2 HPQs per port). On the other hand, EcoCC significantly outperforms HFDI, as the injection-throttling part of EcoCC eliminates congestion trees before the hot-flow-isolation part of EcoCC runs out of HPQs. Thus, the resources used for isolating hot flows (HPQs and CAMs) are released and made available to handle new congestion situations before these situations arise. For its part, IB-ITh shows no sign of removing the congestion tree in these figures. This is because sources are waiting for a BECN notification to start the throttling, HoL-blocking appearing throughout the network in the meanwhile as four congestion trees are present and no immediate countermesures are taken. This does not happen for EcoCC thanks to its HFDI part, which is able to assist and improve the throttling, dealing with HoL-blocking as soon as it appears. The 1VL scheme again achieves the worst results in all the cases.

Figures 7c, 7f and 7i show the network throughput versus time when the traffic pattern #3 is used (i.e. 6 congestion trees are generated in the network) for the network configurations in Table 1. Again, EcoCC outperforms HFDI, while IB-ITh is not able to cope with the high-order HoL-blocking. Note that, in these networks, EcoCC achieves performance results moderately close to those of VOQnet, although the former is configured with far fewer queues than the latter.

4.3 Real Traffic Results

In this section, we show experiment results when real traffic traces are injected in network configuration #2 of Table 1. We use the traces obtained from the following tests included in the HPC Challenge (HPCC) Benchmark:



(a) Network Config.#1. Traffic #1. (b) Network Config.#1. Traffic #2. (c) Network Config.#1. Traffic #3.

(d) Network Config.#2. Traffic #1. (e) Network Config.#2. Traffic #2. (f) Network Config.#2. Traffic #3.

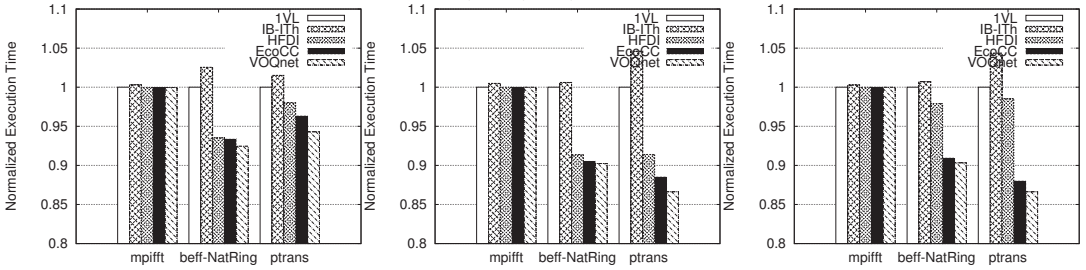
(g) Network Config.#3. Traffic #1. (h) Network Config.#3. Traffic #2. (i) Network Config.#3. Traffic #3.

Fig. 7. Normalized Network Throughput versus Time.

- **mpifft**. Calculates a Discrete Fourier Transform (DFT) of a very large one-dimensional complex data vector. This test requires the tasks exchanging MPI messages, but it does not create significant congestion. However, it has been used to test the consistency of the modeled techniques when MPI collective-operations are used by the applications.
- **beff-NatRing**. Effective bandwidth benchmark (beff) is a set of MPI tests that measure the latency and bandwidth of a number of simultaneous communication patterns. In particular, we have used the *Natural Ring* pattern, based on messages exchange by following a ring communication pattern among the MPI tasks. This traffic pattern introduces a moderate load into the network.
- **ptrans**. Implements parallel matrix transpose that exercises a large-volume communication pattern whereby pairs of processes communicate with each other simultaneously. This traffic pattern creates a high traffic load in the network.

Besides, as other applications may contend with the HPCC tests for the network resources of the HPC system, we also combine the execution of HPCC traces with synthetic traffic modeling "heavy" congestion situations. Next, we summarize the modeled real-traffic scenarios:

- **Real-traffic scenario #1** (SC1). Only HPCC tests are present in the network, without contending with other applications for the network resources.
- **Real-traffic scenario #2** (SC2). Each HPCC test is combined with synthetic traffic generating a single hot-spot. Specifically, 32 endnodes generate traffic addressed to destination #123 (i.e. a 12.5% of endnodes generate hot-spot traffic, like traffic pattern #1 in Table 2). The generation of the hot-spot traffic occurs during all the execution time of the HPCC application. In order to prevent the endnodes from generating traffic both from real-traces and synthetic traffic (i.e. to show the impact of both types of traffic interacting only in the network), we map synthetic traffic to odd nodes and the HPCC test to even nodes. This traffic scenario models a strong, permanent congestion situation appearing in the network while the HPCC tests are run (e.g. due to I/O traffic sharing the same network space).
- **Real-traffic scenario #3** (SC3). HPCC traces are combined with synthetic traffic generating a hot-spot, similar to the traffic pattern #2 in Table 2. (i.e. 25% of the endnodes generate traffic addressed to 4 different hot-spots). This scenario is used to show situations where HFDI may run out of resources,



(a) SC1: Traffic traces alone. (b) SC2: Traffic traces + 1 hot-spot. (c) SC3: Traffic traces + 4 hot-spots.

Fig. 8. Simulation results with real-traffic traces from the HPCC Benchmark in network configuration #2 (See Table 1)

since it is configured with only 2 HPQs.

Figure 8 shows the execution time for the traffic scenarios described above, normalized with respect to the 1VL results. As the *mpifft* test does not create a high traffic load, then it does not show significant differences among the modeled techniques, so that its execution time is barely affected by other applications sharing the network. On the other hand, in SC1 (see Figure 8a), the *beff-NatRing* and *ptrans* tests show that VOQnet achieves the best results (8% and 6% of improvement with respect to 1VL for the *beff-NatRing* and *ptrans* tests, respectively), near to the ones achieved by EcoCC and HFDI (around 7% and 4% with respect to 1VL for *beff-NatRing* and *ptrans* tests, respectively). Note that EcoCC and HFDI achieve similar results, their available HPQs being enough to deal with the temporary congestion trees appearing in the network. Similar conclusions as for synthetic traffic patterns can be extracted for 1VL and IB-ITh (see Section 4.2). In SC2 (see Figure 8b), the differences increase between 1VL and VOQnet, with respect to SC1 (around 10% and 13%, for the *beff-NatRing* and *ptrans* tests, respectively), as the hot-spot traffic delays the execution of the HPCC tests. Besides, the results of EcoCC and HFDI, using only 2 HPQs, are close to those of VOQnet, and they improve those of 1VL around 10% and 11% for the *beff-NatRing* and *ptrans* tests, respectively. Note that in SC1 and SC2, HFDI results are slightly worse than those of EcoCC: the major difference is about 4% in the *ptrans* test of SC2 (see Figure 8b). However, in SC3 (see Figure 8c) HFDI runs out of HPQs in many ports to deal with the 4 congestion trees generated in this scenario, while the injection-throttling part of EcoCC prevents the hot-flow-isolation part running out of resources. Indeed, the improvement of EcoCC with respect to HFDI is almost a 12% in this scenario.

4.4 EcoCC-CCFIT performance comparison

CCFIT [12] has not been considered in the evaluation experiments whose results are shown in the previous subsections. The reason is that CCFIT has been designed for a switch architecture without VOQ support, in contrast with the architecture assumed by EcoCC, thus a "direct" performance comparison between both techniques would be neither accurate nor fair. However, as other CC techniques like 1VL, IB-ITh, HFDI and VOQnet are valid for both architectures, it is possible to get an "indirect" performance comparison between EcoCC and CCFIT based on their respective gains when separately compared with the aforementioned techniques in the appropriate switch architectures.

Table 3 shows that performance gain of both CCFIT and EcoCC over the CC techniques described in Section 4.1 when traffic pattern #2 from Table 2 is used as traffic load for BMIN configurations #1 and #2 from Table 1. In general, the performance gain of EcoCC over other CC techniques is better if compared to the CCFIT performance gain. Notice in particular that EcoCC performance is much closer to VOQnet than the CCFIT one. Notice also that the CCFIT gain is better than the EcoCC gain only over 1VL in BMIN configuration #1 (specifically, 67% against 58%). The reason is that the switches with multiple read-ports have a native low-order HoL-blocking prevention which improves even the 1VL performance.

TABLE 3
Average Gain of EcoCC and CCFIT over other techniques (Synthetic Traffic Pattern #2).

	EcoCC		CCFIT	
Technique	Net. conf.#1	Net. conf.#2	Net. conf.#1	Net. conf.#2
1VL	58%	83%	67%	60%
IB-ITh	55%	75%	46%	51%
HFDI	29%	43%	10%	20%
VOQnet	$\simeq 0\%$	$\simeq 0\%$	-8%	-33%

5 Conclusions

Interconnection networks are essential elements for HPC systems. Network designers are focused on fulfilling requirements such as low power consumption, cost-effectiveness and scalability in order to cope with the objective of the decade: exascale computing, while parallel applications programmers face the challenge of managing and processing big amounts of data (big-data) in a parallel way. In this context, new network designs are prone to suffer from congestion and their negative effects (e.g. the Head-of-Line blocking), since they reach the saturation point with lower traffic loads. Thus, new and powerful congestion control (CC) mechanisms, able to deal with congestion problems, are required. In this paper, we have analyzed the weaknesses of two of the most successful CC approaches, and we have proposed a new technique for CC that combines both approaches to minimize their respective shortcomings, so that the problems derived from congestion are more efficiently eliminated. Our new hybrid technique, called Efficient and cost-efficient Congestion Control (EcoCC), has been designed for switch architectures that, like those of current commercial switches, have Virtual-Output-Queuing (VOQ) support, in contrast with our precedent hybrid CC technique CCFIT. This improvement allows EcoCC to more accurately locate congestion roots and to more efficiently use resources to isolate and throttle hot flows. Experiments results show that, under certain traffic conditions, EcoCC outperforms by up 55% some of the existing most popular CC techniques.

REFERENCES

[1] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic Evolution of Congestion Trees: Analysis and Impact on Switch Architecture," *Proc. 1st HiPEAC Conf.*, pp. 266–285, November 2005.

[2] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queuing on a space-division packet switch," *IEEE Transactions on Communications.*, vol. COM-35, pp. 1347–1356, 1987.

[3] M. Jurczyk and T. Schwederski, "Phenomenon of Higher Order Head-of-Line Blocking in Multistage Interconnection Networks under Nonuniform Traffic Patterns," *IEICE Transactions on Information and Systems*, vol. E79-D, no. 8, pp. 1124–1129, Aug. 1996.

[4] K. Yoshigoe, "Threshold-based Exhaustive Round-Robin for the CICQ Switch with Virtual Crosspoint Queues," in *Proceedings of IEEE International Conference on Communications, ICC 2007, Glasgow, Scotland, 24-28 June 2007*, 2007, pp. 6325–6329.

[5] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control," in *In Proceedings of Int. Workshop High-Performance Interconnects for Distributed Computing*, 2005.

[6] W. Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 194–205, 1992.

[7] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. Nachiondo, "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks," in *Proceedings of the 11th Symposium on High Performance Computer Architecture (HPCA)*, 2005.

[8] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Efficient, Scalable Congestion Management for Interconnection Networks," *IEEE Micro*, vol. 26, no. 5, pp. 52–66, September 2006.

[9] G. Mora, P. J. García, J. Flich, and J. Duato, "RECN-IQ: A Cost-Effective Input-Queued Switch Architecture with Congestion Management," in *Proc. ICPP*, 2007.

[10] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "FBICM: efficient congestion management for high-performance networks using distributed deterministic routing," in *LNCS Series. Proceedings of the 15th international conference on High performance computing*, ser. HiPC'08, 2008, pp. 503–517.

[11] J. Escudero-Sahuquillo, P. J. Garcia, F. J. Quiles, J. Flich, and J. Duato, "An Effective and Feasible Congestion Management Technique for High-Performance MINs with Tag-Based Distributed Routing," *IEEE Transactions on Parallel and Distributed Systems*, no. DOI:10.1109/TPDS.2012.303, 2012.

[12] J. Escudero-Sahuquillo, E. Gran, P. J. García, J. Flich, T. Skeie, O. Lysne, F. J. Quiles, and J. Duato, "Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks," in *Proc. Int. Conf. Parallel Processing*, 2011.

[13] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.

[14] E. Gran, E. Zahavi, S.-A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, "On the Relation between Congestion Control, Switch Arbitration and Fairness," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 342–351, 2011.

[15] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[16] E. G. Gran, S.-A. Reinemo, O. Lysne, T. Skeie, E. Zahavi, and G. Shainer, "Exploring the Scope of the InfiniBand Congestion Control Mechanism," in *2012 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, B. Werner, Ed. IEEE Computer Society, 2012.

[17] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, 1999.

[18] *InfiniBand architecture specification volume 1. Release 1.2.1*, InfiniBand Trade Association, Nov. 2007.

[19] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "Cost-Effective Congestion Management for Interconnection Networks Using Distributed Deterministic Routing," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010), Shanghai, China*, december 2010.

[20] E. Gran, M. Eimot, S. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *In Proceedings of Int. Parallel and Distributed Processing Symposium*, 2010, pp. 1–12.

[21] C. Gomez, F. Gilabert, M. Gomez, P. Lopez, and J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," in *Workshop on Communication Architecture on Clusters, as a part of IPDPS'07*, March 2007, p. 235.

[22] The HPCC Benchmark, Web Page at: http://icl.cs.utk.edu/hpcc.

[23] H. S. Gelabert and G. L. Sánchez, *Extrae User Guide Manual for version 2.2.0*, Barcelona Supercomputing Center (BSC), November 2011.

SUPPLEMENTAL DOCUMENT

Jesus Escudero-Sahuquillo, *Member, IEEE*, Ernst Gunnar Gran, *Student Member, IEEE*, Pedro J. Garcia, José Flich, *Member, IEEE*, Tor Skeie, *Member, IEEE*, Olav Lysne, *Member, IEEE*, Francisco J. Quiles, *Member, IEEE*, and José Duato

In this document, we provide the supplemental material for the submission: "Efficient and Cost-Effective Hybrid Congestion Control for HPC Interconnection Networks", consisting of an overview of existing congestion-management approaches in high-performance interconnection networks, some details about the EcoCC parameters tunning and memory requirements, as well as additional evaluation results not included in the main document due to space constraints.

6 Congestion Control in High-Performance Interconnects

As we mention in the main paper, some "classical" congestion control (CC) approaches like blocked-packet discarding or network overdimensioning are not suitable for current HPC systems. Packet discarding in congestion situations is allowed in computer communication networks ("lossy" networks, e.g. Internet). By contrast, in HPC systems, low latency is crucial, thus packet dropping and retransmission are not allowed under regular circumstances, so "lossless", high-speed networks are used (e.g. Infiniband [1]). Similarly, in the old days, CC in HPC systems was performed using a combination of i) tuning and tailoring of network characteristics like topology and routing, depending on the traffic pattern of a given application, and ii) overprovisioning of network resources. By introducing virtualization to improve utilization, knowledge about the traffic pattern imposed is limited, thus option i) has become obsolete. Furthermore, overprovisioning the network contradicts the current trend of making the HPC installation cost-effective and green. This renders option ii) invalid.

Similarly, proactive CC techniques [2], [3], [4], [5] are not nowadays considered to be appropriate CC solutions for HPC systems. These strategies assign network resources to each data transmission before it starts, according to some planning, so that congestion situations never happen (or, at least, they are unlikely to happen). In general, this approach requires a traffic scheduler (either centralized or distributed) usually based on a-priori knowledge of (total or partial) network status and/or resource requirements for each transmission. In HPC scenarios, this information may be difficult to provide and, in addition, the resource assignment procedure adds significant overhead. Actually, this approach is more suited to Quality of Service (QoS) provision.

Other strategies that may help to alleviate congestion or delay its appearance are the use of fully adaptive routing [6], [7], [8] or load balancing techniques [9], [10]. In order to be effective against congestion, these strategies should make routing decisions based on network status, as the technique proposed for Networks-on-Chip in [11] does. However, these techniques cannot avoid network performance degradation once congestion appears. In particular, note that adaptive routing or load balancing techniques will not help when there is no alternative route avoiding the root of the congestion tree. This is the case when the root of the tree is located at the last switch on the path towards a hot-spot destination node, or in general when a destination node is overwhelmed with traffic. In such a scenario, adaptive routing might actually broaden the congestion tree, and make the HoL blocking more severe. As the routing mechanism is trying to distribute the traffic headed for the hot-spot destination in the network, the effect of the routing on the congestion is unfortunately that the branches of the congestion tree are growing wider and/or that the root of the congestion tree is moved closer to the destination node. In addition, adaptive routing can cause out-of-order packet delivery, which is unacceptable to typical HPC applications. For their part, efficient switch scheduling algorithms (e.g. iSLIP [12]) can only delay the appearance of low-order HoL-blocking,

- *J. Escudero-Sahuquillo, Pedro J. Garcia and Francisco J. Quiles are with the Computer Systems Department, Escuela Superior de Ingeniería Informática, Universidad de Castilla-La Mancha, Campus Universitario, s/n 02071, Albacete, Spain. E-mail: {jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es*
- *Ernst G. Gran, Tor Skeie and Olav Lysne are with the Simula Research Laboratory, Martin Linges vei 17, Fornebu, Norway. E-mail: {ernstgr, tskeie, olav.lysne}@simula.no. Tor Skeie is also affiliated with the University of Oslo. E-mail:{tskeie}@ifi.uio.no*
- *J. Flich and J. Duato are with the Department of Computer Engineering (DISCA), Technical University of Valencia, Camino de Vera, s/n 46071, Valencia, Spain. E-mail: {jflich, jduato}@disca.upv.es*

and cannot prevent high-order HoL-blocking (examples of low- and high-order HoL-blocking can be found in the main paper); indeed they are not actually CC mechanisms.

In most modern HPC systems, CC is based either on injection throttling or on separating hot flows from cold ones in order to prevent the problems derived from congestion, mainly HoL-blocking. In the following sections we analyze in-depth these two basic approaches and the main proposals based on them.

6.1 Injection-Throttling Strategies

As explained in the main document, the carrying idea of injection throttling is that the source nodes that contribute traffic to a congested link should inject less traffic into the network. This requires a flow of information from the switch adjacent to the congested link, i.e. the switch at the root of the congestion tree, and back to the source nodes. There are countless variants of this procedure. For instance, congestion notifications could be sent to all the sources [13] or just to the sources contributing to congestion [14]. Other proposed mechanisms notify congestion just to the local endpoints attached to the switch where congestion is detected [15]. Furthermore, the switches can mark the packets contributing to congestion in order to notify the destinations about the situation, which subsequently notify the sources (the forward explicit notification approach), or the switches can themselves generate notification packets that are sent directly to the source nodes (the backward explicit notification approach). InfiniBand applies the former approach [16], [17], while the Data Center Bridging standard [18] is implementing the latter. There is also a body of work that propose different strategies for congestion notification and marking, e.g. a hot packet can be marked both in the input and output switch buffer, as well as being tagged with information about the severity of congestion [19]. Moreover, there are some different approaches for designing sources response function, i.e. the actions taken to reduce the injection rate, later followed by an increase in the rate when congestion is resolved [20], [21], [22].

The injection throttling part of our hybrid proposals are inspired by the CC mechanism specified for InfiniBand. The InfiniBand CC defines two bits in the packet header for congestion notification. Specifically, if a packet is considered to contribute to congestion at a switch port, the *Forward Explicit Congestion Notification* (FECN) bit in the packet header is set. The FECN bit is then carried through the network to the destination node by the packet. Upon reception of a "FECN-marked" packet, a destination will return back to the source a packet whose header will have the *Backward Explicit Congestion Notification* (BECN) bit set. Sources receiving BECN packets will reduce the injection rate of the corresponding hot flow, thus alleviating congestion.

The performance of InfiniBand CC depends on several configurable parameters [23]. A congestion detection *threshold* parameter mapped to a buffer fill ratio at a switch port[1] determines when the port is considered to be congested. If the buffer fill ratio is above the *threshold* the corresponding switch port is moved into *Congestion State*, so that packets crossing that port may thereafter be FECN-marked. In order not to generate too many BECNs, not all the packets crossing a port in Congestion State are FECN-marked: Only those whose size is greater than the value of the *Packet_Size* parameter, and among them again, only a fraction corresponding to the *Marking_Rate* parameter, are finally marked.

Similarly, the exact reaction of a source node upon the reception of a BECN also depends on a set of CC parameters. Specifically, the injection rate of a hot flow is reduced by introducing an *injection rate delay* (IRD) between consecutive packets of that particular flow. Source nodes store a list of possible IRD values in a *Congestion Control Table* (CCT), each hot flow holding an index (CCTI) into this table. CCT values are typically arranged so that the higher the index, the greater the IRD. Upon reception of a BECN the index of a flow is increased by a value stated by the *CCTI_Increase* parameter. The CCTI is decremented again by one when a timer (whose value is stated by the *CCTI_Timer* parameter) expires. Hence, hot flows are throttled while congestion exists, and released when congestion vanishes. More details about the InfiniBand CC can be found in [23], [24], [25].

As discussed in Section 2 of the main document, the InfiniBand CC mechanism, like injection throttling techniques in general, bear a weakness: the delay between the congestion detection at a switch and the reaction at the sources results in a CC mechanism operating behind schedule [26], where oscillating sources are adjusting their injection rates based on "old" information [23], [25].

6.2 Hot-Flow Isolation Strategies

Most proposals to prevent HoL-blocking by flow isolation are based on allocating several queues at each switch port to separately store packets, thus lowering HoL-blocking probability. For instance, a well known HoL-blocking prevention technique is Virtual Output Queues (VOQs), either at the switch level (VOQsw) [27] or at the network level (VOQnet) [28]. VOQsw uses at each port as many queues as there are output ports in the switch. The incoming packets are then managed in such a way that each packet immediately is mapped to a queue corresponding to

1. For the sake of simplicity, the concept of *Virtual Lanes* (VL) has been left out of this explanation of the InfiniBand CC.

its next output port. Thus VOQsw totally prevents low-order HoL-blocking, but not high-order one. On the other hand, VOQnet prevents packets addressed to different destinations from sharing queues at any port, thereby totally preventing low- and high-order HoL-blocking. This scheme, however, requires at each port as many queues as destinations in the network, thus it does not scale with network size as each queue requires to reserve its own memory space. The Dynamically Allocated Multi-Queues (DAMQs) [29] use the same queue scheme as VOQsw (thus not preventing high-order HoL-blocking), but in this case queue size may dynamically vary as required. Similar strategies are Destination-Based Buffer Management (DBBM) [30], Dynamic Switch Buffer Management (DSBM) [31] and Output-Based Queue Assignment (OBQA) [32], [33], [34]; although these solutions have been specially proposed to deal with the HoL-blocking caused by congestion, their mapping policies actually may lead to hot packets sharing queues with cold ones, and thereby only partially preventing HoL-blocking.

Virtual Channels [35], VCs (or Virtual Lanes, VLs) may also reduce HoL-blocking, but their behavior depends mainly on the specific policy used to map packet flows to VCs. Actually, all the aforementioned HoL-blocking prevention techniques (or equivalent versions) could be implemented through VCs, provided that enough VCs are available. An alternative proposal [36] shuffles different flows sharing a link among the available VLs, but (again) this proposal can guarantee a complete HoL-blocking prevention only if the number of VLs equals the maximum number of different flows in a link.

Note that all the aforementioned HoL-blocking prevention techniques do not explicitly identify hot flows, but they rely on "obliviously" separating packets from different flows as much as possible with the available queues at each port. Thus, their effectiveness greatly depend on the number of queues per port. By contrast, the Hot-Flow Dynamic Isolation (HFDI) techniques explicitly detect congestion roots to keep track of hot flows, in order to isolate them in special, dynamically-assigned queues. Then, the cold flows may share queues without suffering significant HoL-blocking. Hence, the number of queues required to efficiently eliminate HoL-blocking is reduced. The main HFDI techniques are *Regional Explicit Congestion Notification* (RECN) [37], [38], *Regional Explicit Congestion Notification-Input Queued* (RECN-IQ) [39] and *Flow-Based Implicit Congestion Notification* (FBICM) [40]. The "HFDI-part" of our hybrid proposals are inspired by the latter technique, as it has been proposed for interconnects using distributed routing, like InfiniBand.

HFDI techniques detect congestion roots by locally monitoring queue occupancy at each switch port, and comparing it with a *Detection Threshold*. Once a congestion root has been located in a port, a special queue is allocated to store hot packets contributing to that root. Such a technique then requires a set of special queues at each port to store hot packets, one queue per identified root of congestion. These special queues are called Set-Aside-Queues, SAQs, in RECN and RECN-IQ, and Congested-Flow-Queues, CFQs, in FBICM. In addition, a control memory is required to manage these queues, mainly to store the location of the congestion root each special queue is assigned to. This control memory is implemented by means of a Content-Addressable Memory (CAM) present at each port. In the case of RECN and RECN-IQ, designed for source-based routing networks, each CAM line stores (among other information) the explicit path towards the root of the congestion tree assigned to a specific SAQ, while in the case of FBICM, designed for deterministic distributed routing, each CAM line stores a set of destinations. Hot packets are identified by comparing the routing information of each packet arriving to a port with the information stored in the active CAM lines at that port. If the occupancy of any SAQ/CFQ in a port reaches a specific threshold, the congestion information stored in its corresponding CAM line is propagated (by control packets) to the neighboring upstream switches, which in turn will allocate a new SAQ/CFQ for this specific congestion tree. In this way, special queues are allocated all along the way of hot flows to separate them from cold ones, thereby completely preventing HoL-blocking. SAQs/CFQs are dynamically deallocated when the corresponding congestion tree vanishes, and later reallocated if a new congestion tree appears.

As discussed in Section 2 of the main document, although HFDI solutions are very effective, they bear the important flaw of the limited number of special queues per port, which may not be enough to handle all the possible congestion trees simultaneously being present at a port.

7 EcoCC Parameter Tunning Principles

As it has been mentioned throughout the main document of this paper, EcoCC requires several parameters to be configured in the switches and HCAs. These parameters are High (HTh) and Low (LTh) Thresholds, *CCTI_Timer*, *Marking_Rate* and *Packet_Size*. Further details about the tunning of *CCTI_Timer*, *Marking_Rate* and *Packet_Size* parameters are given in [23]. For their part, HTh and LTh thresholds must be configured bearing in mind the following principles:

- At a given input port, when the occupancy level of a specific VOQ exceeds the HTh threshold, EcoCC moves that output port (i.e. the root of the congestion) into the congestion state. In this moment, a percentage of the packets crossing through that congestion root will be marked, according to the *Marking_Rate* parameter. The

value of the HTh threshold has to be high enough to not propagate too early a congestion situation which could disappear quite soon. Furthermore, it has to bee low enough to not detect the congestion too late.
- The HTh and LTh thresholds, as it is described in [24], should have a distance of at least two packet MTUs, in order to implement a fair arbitration of the traffic flows belonging to the different HPQs.
- As mentioned above, when the occupancy level of one HPQ exceeds the HTh threshold, EcoCC sends congestion notifications to upstream switches in order to implement the HPQ flow-control policy. The HTh value for the HPQs should be low enough for both avoiding the buffer hogging, and assuring a minimum storage space for the hot packets that may be flying over the link.

These principles have been taken into account to configure the values of the parameters used in our experiments, that are indicated in Section 4.1 of the main document. Besides, the correctness of these principles have been confirmed by simulation experiments, like those shown in the main paper and in this supplementary document.

8 Memory Requirements

In this Section, we analyze the memory requirements of the congestion control techniques evaluated in the main document in terms of area, power consumption and access time. In particular, we have first estimated the common attributes for the data memories that we have assumed at input ports of the switches that we have modeled for the evaluation shown in Section 4 of the main document. The values of these attributes have been calculated by means of the CACTI tool v5.3 [41] using its *pure SRAM modeling* feature.

TABLE 3
CACTI parameters for input-port data memories

Description	Value
Type of memory	SRAM
Number of banks	1
Total Size (KB)	128
Read Ports per bank	8
Write Ports per bank	1
Technology Size (nm)	32
Readout value (bytes)	5
Vdd	0.9

TABLE 4
Estimated results for SRAM attributes

Attributes for EcoCC / HFDI / IB-ITh / 1VL	Value
Access time (ns)	1.19375615509
Random cycle time (ns)	0.201132407786
Total read dynamic energy per read port (nJ)	0.041173774067
Total read dynamic power per read port at max freq (W)	0.204709795504
Total standby leakage power per bank (W)	0.0761409420512
Total area (mm^2)	7.07580828776

Table 3 shows the values of the SRAM parameters we have used in CACTI for the estimation of SRAM attributes. Note that, as indicated in Section 4.1 of the main document, we assume input-port memories of 128 KB with 8 read ports and one write port for EcoCC, HFDI, IB-ITh and 1VL techniques. Table 4 shows the obtained results. As the same organization for the input-port data memories is assumed, this values are valid for all the aforementioned techniques. However, as we have shown in the evaluation section of the main document, there are significant differences among them in terms of network performance. Note also that we have not included results for VOQnet, since this technique is unfeasible in large interconnection networks, especially in terms of area.

On the other hand, we have estimated the requirements, in number of cells, for the control memories (CAMs), used by EcoCC and HFDI (each cell stores a bit of information). We assume that each CAM line (see Figure 4 of the main document) requires 6 cells for the OP field to refer up to 64 switch ports, 6 cells for the $Hops$ field to refer a congestion root located to a distances up to 64 hops in the network, and 4 cells for the $NextLine$ field to refer up to 16 CAM lines in downstream CAM structures. Besides, some SRAM cells per CAM line are required for the control bits: 1 cell for the $stop$ bit, 1 cell for the $SentStop$ bit, 1 cell for the $propagated$ bit and 32 cells for the $timer$ bits. Summing up, 16 CAM cells and 35 SRAM cells are required per CAM line to store all the fields apart from the $dList$ one. In that sense, if N is the number of network endnodes, we need $log_2(N)$ CAM cells to encode a single destination and store it in the $dList$ field of a CAM line, this field consisting of a limited number of destinations in order to be feasible. Specifically, as it is mentioned in Section 4.1 of the main paper, we assume that 8 destinations are stored per CAM line in networks with 64 nodes (i.e. $8 \times log_2(64) = 48$ CAM cells for the $dList$ field), 16 destinations in networks with 256 nodes (i.e. $16 \times log_2(256) = 128$ cells for the $dList$ field) and 32 destinations in networks with 1024 nodes (i.e. $32 \times log_2(1024) = 320$ cells for the $dList$ field). Table 5 summarizes the number of CAM and SRAM cells required per CAM line for these network sizes.

It is worth pointing out that the number of cells per input port depends on the number of CPQs and HPQs per input port, as each of these queues requires a CAM line. For instance, in Section 4.1 of the main document, we define that 9 CAM lines are required per input-port buffer: one for each of the seven CPQs (remember that we assume 8-port switches, thus 7 VOQs are defined per input-port) and one for the two HPQs configured. Thus, the

TABLE 5
CAM and SRAM cells required per CAM line

Network size	#CAM cells	#SRAM cells
64 nodes	64	35
256 nodes	144	35
1024 nodes	336	35

number of CAM cells per input port for a network of 1024 nodes (the most-requiring case among the evaluated networks) is given by 9 CAM lines per input port $\times$ 336 cells per line = 3024 CAM cells per input port. Similarly, $9 \times 35 = 315$ SRAM cells per input port are required. It is worth mentioning that current technology allows the CAMs structures to be implemented within the same area footprint as a standard RAM cell [42], [43]. Thus, the area requirements of CAM structures per input port, for a network of 1024 endnodes, are equivalent to 3024+315=3339 cells (approximately 418 bytes) of RAM, which is quite affordable.

In conclusion, the requirements of EcoCC regarding both data (SRAM) buffers and CAM structures are reduced in terms of area, access time and power consumption, thereby being cost-efficient.

9 Additional Evaluation Results

In this Section, we show experiment results additional to those shown in Section 4 of the main document. Specifically, we analyze how likely is that EcoCC and HFDI run out of HPQs during congestion situations. Besides, we also add the complete experiment results for the evaluation of the CCFIT technique [26] that are summarized in Section 9.2.

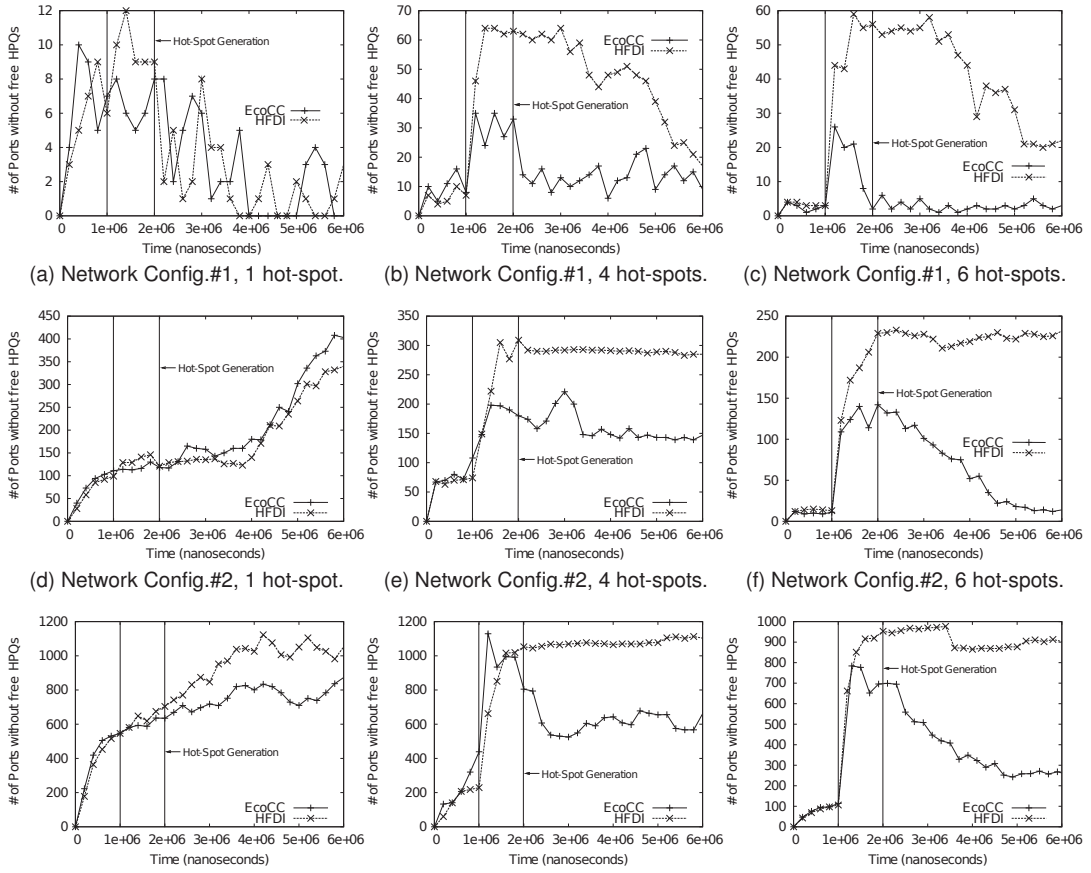
9.1 HPQs demand

Figure 10 depicts, for EcoCC and HFDI, the total number of switch input ports in the network that have no available HPQs to be allocated for isolating hot flows belonging to new congestion trees, i.e. those switch ports where there are no free resources for preventing new hot flows from causing HoL-blocking and buffer-hogging. Note that the simulation model is the one described in Section 4.1 of the main document. Thus, the assumed network configurations are #1, #2 and #3 of Table 1 of the main document, and the traffic cases are those defined in Table 2 of the main document.

Specifically, figures 10a, 10d and 10g show the results when only 1 hot-spot (traffic pattern #1 of Table 2 of the main paper) is generated for the network configurations #1, #2 and #3 of Table 1 (see the main document), respectively. In this case, the uniform traffic generated by the 87.5% of the source nodes (see Table 2 of the main paper), only generates temporary congestion situations that disappear very quickly, thereby that percentage of "uniform" flows only use HPQs during short periods of time. By contrast, hot flows contributing to the single generated hot-spot (i.e. congestion root) will last for long time, but all of them will require only one HPQ to be isolated at any switch port. Thus, regardless the life-time of this HPQ, both HFDI and EcoCC have enough resources to cope with this congestion situation, as they have been configured with 2 HPQs per input port. For that reason, both techniques achieve similar results.

On the other hand, figures 10b, 10e and 10h depict the results when 4 hot-spots are generated (traffic pattern #2 of Table 2 of the main paper), for the network configurations #1, #2 and #3 of Table 1 (see the main document), respectively. In that case, the number of congestion trees is higher than the available number of HPQs per switch input port. Thus, different hot flows contributing to the 4 different congestion trees could converge at the same switch port; if so, some of these hot flows couldn't be isolated in HPQs and would be stored in "normal" VOQs, which are likely to fill up quickly due to the high reception rate of hot packets. If EcoCC is used, this situation can be solved when the injection rate of the hot flows that have been previously isolated in HPQs is reduced, so that some HPQ is eventually deallocated, then reallocated to isolate the hot flows mapped previously to VOQs. By contrast, HFDI is not able to solve these situations as it does not throttle injection, thereby being likely to run out of HPQs at many ports, as can be seen in the figures. Besides, as explained in the main document, EcoCC does not require to allocate HPQs to isolate hot flows contributing to a "local" congestion root, while HFDI requires this allocation, thus the former leverages the use of these resources more efficiently than the latter.

Similarly, figures 10c, 10f and 10i show the results when 6 hot-spots are generated (traffic pattern #3 of Table 2) for the network configurations #1, #2 and #3 of Table 1 (see the main document), respectively. As the number of hot-spots increases, so does the number of ports without free HPQs for the HFDI technique, while EcoCC allocate, release and reallocate HPQs more efficiently, being less likely to run out of resources to deal with congestion.



(a) Network Config.#1, 1 hot-spot. (b) Network Config.#1, 4 hot-spots. (c) Network Config.#1, 6 hot-spots.

(d) Network Config.#2, 1 hot-spot. (e) Network Config.#2, 4 hot-spots. (f) Network Config.#2, 6 hot-spots.

(g) Network Config.#3, 1 hot-spot. (h) Network Config.#3, 4 hot-spots. (i) Network Config.#3, 6 hot-spots.

Fig. 10. Total number of switch ports without free HPQs vs. Time.

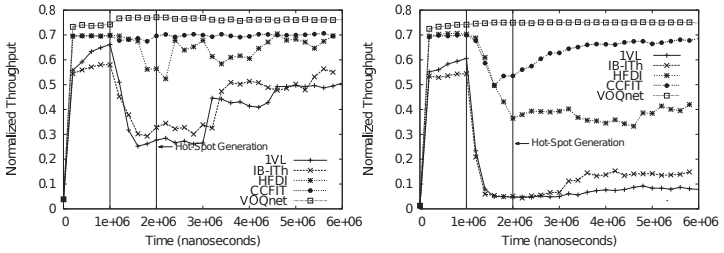
9.2 CCFIT Experiments Results

In Section 4.4 of the main document, EcoCC and CCFIT are compared based on their respective average performance gains with respect to other techniques. The complete series of results used to obtain the average gains of EcoCC are shown in the main document, while the complete series used to obtain the average gains of CCFIT are shown in this Section, in Figure 11.

Specifically, these results have been obtained for network configurations #1 and #2 of the Table 1 (See the main paper), when the traffic pattern #2 (see the Table 2 of the main document). We have made experiments for CCFIT and for the CC techniques which are compared with EcoCC in the main document, but this time assuming switches with single read-ports for all these experiments. CCFIT and HFDI have been modeled assuming buffers at switch input ports organized in 1 CPQ and 2 HPQs, plus CAM structures using 2 CAM lines. For their part, 1VL, IB-ITh and VOQnet have been modeled similarly as for the experiments analyzed in the main document.

Figure 11 shows network throughput as a function of time. In general, HFDI suffers a performance degradation as it has not enough resources per port to isolate all the flows belonging to different congestion trees that may be present in a switch port. Like EcoCC, CCFIT achieves a significant throughput improvement with respect to HFDI. CCFIT also outperforms IB-ITh which suffers from the problems described in the main paper. The 1VL scheme, which does not implement any HoL-blocking prevention mechanism at all, achieves the worst results in all the cases, as expected. Overall, CCFIT achieves quite good results, outperforming other CC techniques except VOQnet, the ideal but unfeasible technique. However, despite these good results of CCFIT, EcoCC achieves a higher

performance gain with respect to the other CC techniques anlyzed, as shown in Table 3, in Section 4.4 of the main document.



(a) Network Configuration #1. (b) Network Configuration #2.

Fig. 11. Network Efficiency vs. Time. k-ary n-tree BMINs. Traffic pattern #2 (4 Congestion Trees).

REFERENCES

[1] *InfiniBand architecture specification volume 1. Release 1.2.1*, InfiniBand Trade Association, Nov. 2007.

[2] P. C. Yew, N. F. Tzeng, and D. H. Lawrie, "Distributing Hot-Spot Addressing in Large-Scale Multiprocessors," *IEEE Trans. Comput.*, vol. 36, no. 4, pp. 388–395, 1987.

[3] M. Wang, H. J. Siegel, M. A. Nichols, and S. Abraham, "Using a Multipath Network for Reducing the Effects of Hot Spots," *IEEE Trans. on Parallel and Distributed Systems*, vol. 6, no. 3, pp. 252–268, March 1995.

[4] L. Peh and W. Dally, "Flit-Reservation Flow Control," in *In Proceedings of the International Symposium on High-Performance Computer Architecture*, 2000, pp. 73–84.

[5] N. Chrysos, "Congestion management for non-blocking Clos networks," in *In Proceedings of ACM/IEEE Symp. Architecture for Networking and Communications Systems*, 2007, pp. 117–126.

[6] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 12, pp. 1320–1331, 1993.

[7] W. Dally and H. Aoki, "Deadlock-free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Transactions Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466–475, April 1993.

[8] M. Thottetodi, A. Lebeck, and S. Mukherjee, "BLAM: A High-Performance Routing Algorithm for Virtual Cut-Through Networks," in *Proc. Int. Parallel and Distributed Processing Symp.*, 2003.

[9] D. Franco, I. Garcés, and E. Luque, "A new method to make communication latency uniform: distributed routing balancing," in *ICS '99: Proceedings of the 13th international conference on Supercomputing*. New York, NY, USA: ACM, 1999, pp. 210–219.

[10] A. Singh, W. Dally, B. Towles, and A. K. Gupta, "Globally Adaptive Load-Balanced Routing on Tori," *Computer Architecture Letters*, vol. 3, no. 1, pp. 6–9, July 2004.

[11] P. Gratz, B. Grot, and S. W. Keckler, "Regional Congestion Awareness for Load Balance in Networks-on-Chip," in *The 14th International Symposium on High-Performance Computer Architecture (HPCA)*, 2008, pp. 203–215.

[12] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, 1999.

[13] M. Thottethodi, A. Lebeck, and S. Mukherjee, "Self-Tuned Congestion Control for Multiprocessor Networks," in *Proc. Int. Symp. High-Performance Computer Arch.*, 2001, pp. 107–118.

[14] J. Kim, Z. Liu, and A. Chien, "Compressionless Routing: a framework for adaptive and fault-tolerant routing," in *Computer Architecture, 1994., Proceedings the 21st Annual International Symposium on*, apr 1994, pp. 289–300.

[15] E. Baydal and P. López, "A Robust Mechanism for Congestion Control: INC," in *Proc. Int. Euro-Par Conf.*, 2003, pp. 958–968.

[16] M. Gusat, D. Craddock, W. Denzel, A. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato, "Congestion Control in InfiniBand Networks," in *In Proceedings of Hot Interconnects*, 2005, pp. 158–159.

[17] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control," in *In Proceedings of Int. Workshop High-Performance Interconnects for Distributed Computing*, 2005.

[18] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification.*, IEEE 802.1Qau-2010 ed., IEEE 802 LAN/MAN Standards Committee, 2010. [Online]. Available: http://www.ieee802.org/1

[19] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "On the Influence of the Packet Marking and Injection Control Schemes in Congestion Management for MINs," in *Euro-Par*, 2008, pp. 930–939.

[20] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-End Congestion Control for InfiniBand," in *INFOCOM*, 2003.

[21] ——, "Evaluation of Congestion Detection Mechanisms for InfiniBand Switches," in *IEEE GLOBECOM – High-Speed Networks Symposium*, 2002.

[22] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Congestion Management in MINs through Marked and Validated Packets," in *PDP*, 2007, pp. 254–261.

[23] E. Gran, M. Eimot, S. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *In Proceedings of Int. Parallel and Distributed Processing Symposium*, 2010, pp. 1–12.

[24] E. Gran, E. Zahavi, S.-A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, "On the Relation between Congestion Control, Switch Arbitration and Fairness," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 342–351, 2011.

[25] E. G. Gran, S.-A. Reinemo, O. Lysne, T. Skeie, E. Zahavi, and G. Shainer, "Exploring the Scope of the InfiniBand Congestion Control Mechanism," in *2012 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, B. Werner, Ed. IEEE Computer Society, 2012.

[26] J. Escudero-Sahuquillo, E. Gran, P. J. García, J. Flich, T. Skeie, O. Lysne, F. J. Quiles, and J. Duato, "Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks," in *Proc. Int. Conf. Parallel Processing*, 2011.

[27] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.

[28] W. Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *Proc. of 6th Hot Interconnects*, 1998, pp. 41–50.

[29] Y. Tamir and G. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol. 41, no. 6, June 1992.

[30] T. Nachiondo, J. Flich, and J. Duato, "Buffer Management Strategies to Reduce HoL Blocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 739–753, 2010.

[31] W. Olesinski, H. Eberle, and N. Gura, "Scalable alternatives to virtual output queueing," in *Proc. IEEE International Conference on Communications*, 2009.

[32] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, and J. Duato, "An Efficient Strategy for Reducing Head-of-Line Blocking in Fat-Trees," in *LNCS Series. Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy*, september 2010, pp. 413–427.

[33] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "OBQA: Smart and cost-efficient queue scheme for Head-of-Line blocking elimination in fat-trees," *J. Parallel Distrib. Comput.*, vol. 71, no. 11, pp. 1460–1472, 2011.

[34] ——, "Cost-effective queue schemes for reducing head-of-line blocking in fat-trees," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 17, pp. 2235–2248, 2011.

[35] W. Dally, "Virtual-Channel Flow Control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, pp. 194–205, 1992.

[36] W. L. Guay, B. Bogdanski, S.-A. Reinemo, O. Lysne, and T. Skeie, "vFtree - A Fat-tree Routing Algorithm using Virtual Lanes to Alleviate Congestion," in *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium*, Y. Xin, Ed. IEEE Computer Society Press, 2011, pp. 197–208.

[37] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. Nachiondo, "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks," in *Proceedings of the 11th Symposium on High Performance Computer Architecture (HPCA)*, 2005.

[38] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Efficient, Scalable Congestion Management for Interconnection Networks," *IEEE Micro*, vol. 26, no. 5, pp. 52–66, September 2006.

[39] G. Mora, P. J. García, J. Flich, and J. Duato, "RECN-IQ: A Cost-Effective Input-Queued Switch Architecture with Congestion Management," in *Proc. ICPP*, 2007.

[40] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "Cost-Effective Congestion Management for Interconnection Networks Using Distributed Deterministic Routing," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010), Shanghai, China*, december 2010.

[41] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1. Technical Report HPL-2008-20," Hewlett-Packard Development Company, Tech. Rep., April 2008.

[42] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, March 2006.

[43] R. J. Bucki, J. S. Atwal, J. S. Barnes, K. Bernstein, and E. Robinson, "Compact Multi-port CAM Cell Implemented in 3D Vertical Integration," Patent US 8 343 814 B2, January, 2013.