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

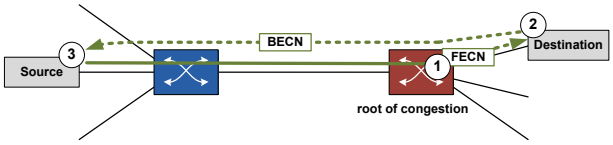


Figure 1: Congestion control in InfiniBand.

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.

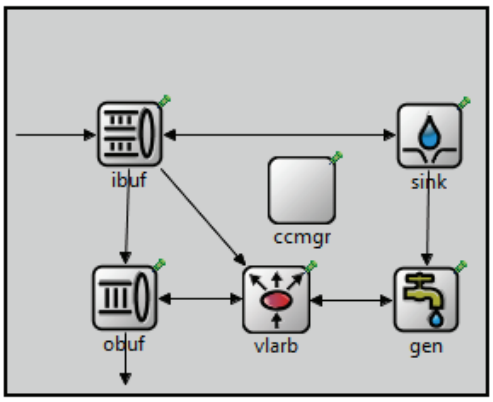


Figure 2: The HCA compound module of the IB model.

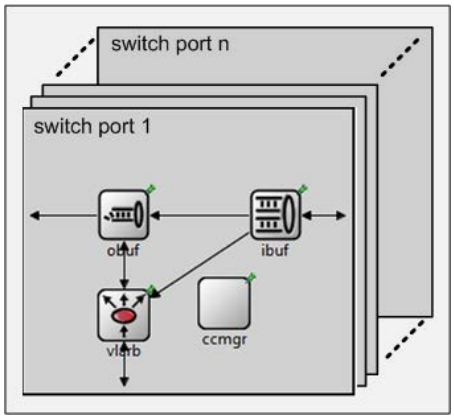


Figure 3: The switch compound module of the IB model.

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].

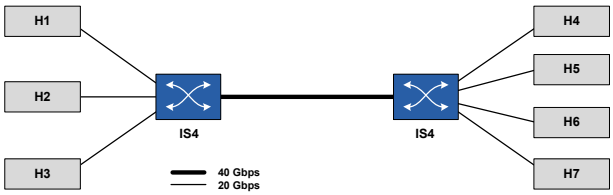


Figure 4: The test bed.

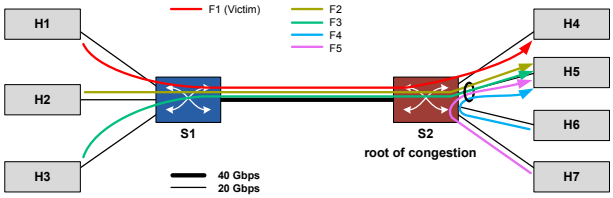


Figure 5: Flow configuration in scenario 1.

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.

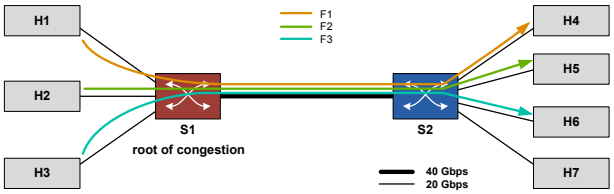


Figure 6: Flow configuration in scenario 2.

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.

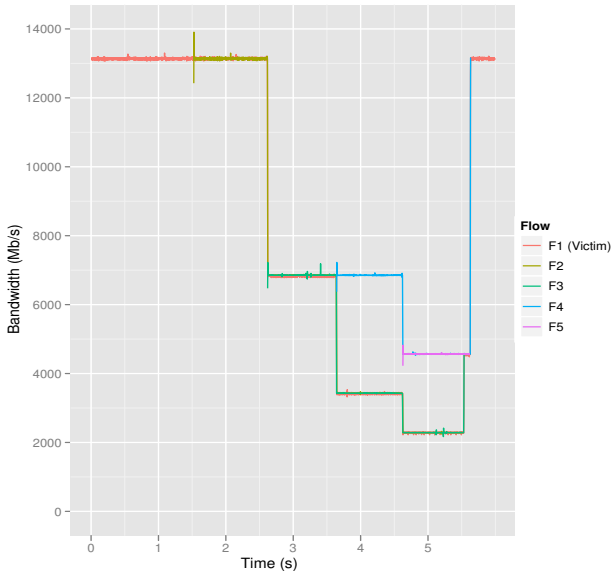
If we turn *on* congestion control in the hardware and repeat our scenario 1 experiment, we got the results presented in figure 7(b). Now, both the HOL blocking and the parking lot problem are removed. The flow $F1$ is sent through the network independently of the other flows, and all the contributors 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.

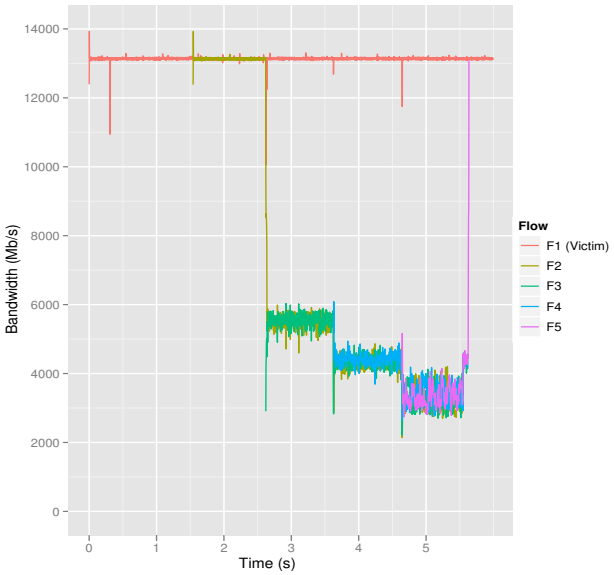
There is one noticeable difference in the two figures 7(b) and 7(d) though: In the one second time interval from 2s to 3s, the flow $F1$ experiences some oscillation during a simulation, an oscillation that is not present in the hardware experiment. This difference in behavior is related to the aggressiveness 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

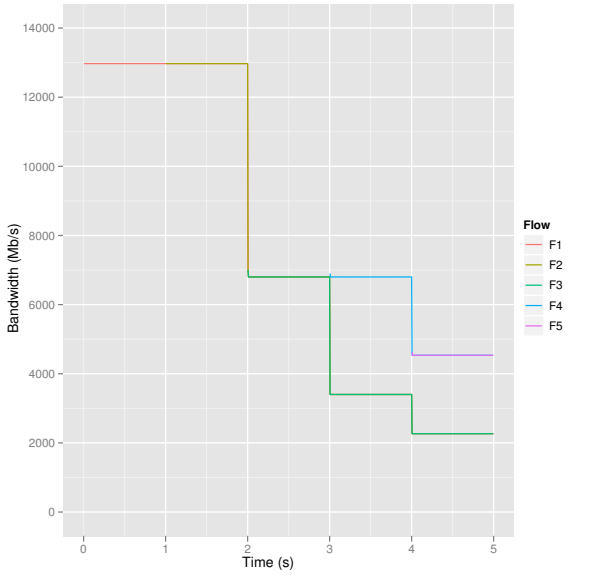
In figure 8(a) and 8(b) we see the results from the hard-



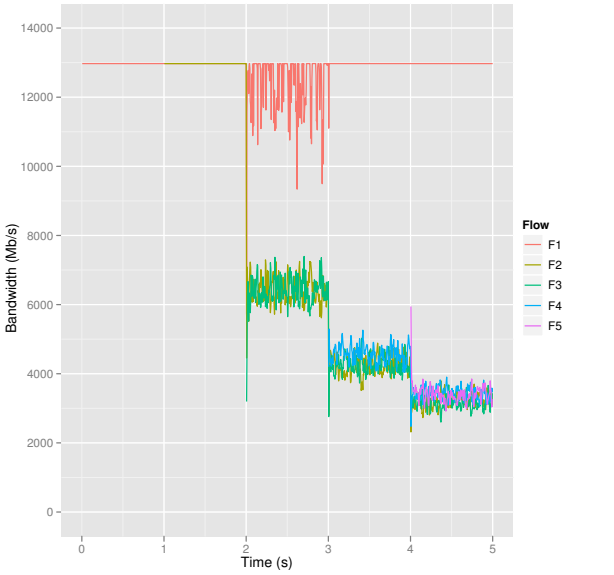
(a) Hardware. Congestion Control turned OFF.



(b) Hardware. Congestion Control turned ON.

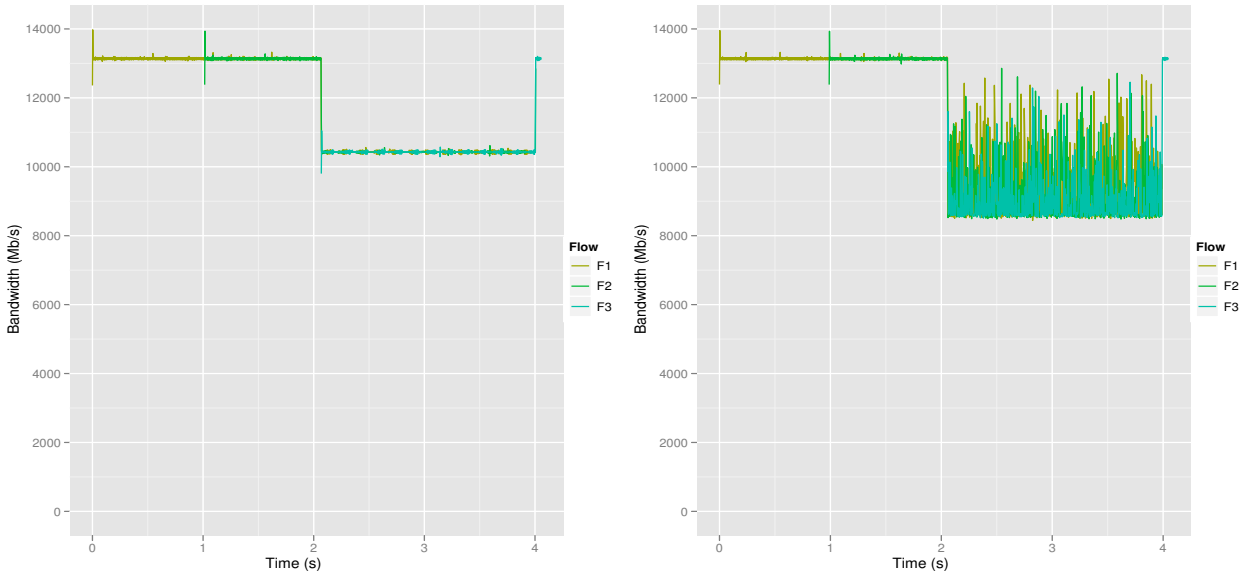


(c) Simulation. Congestion Control turned OFF.



(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.

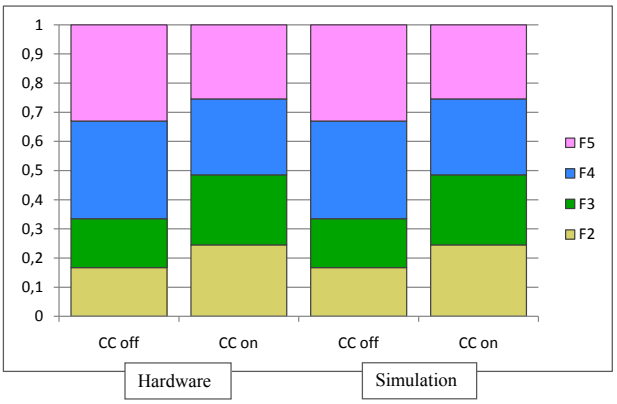


Figure 9: Fairness between the flows contributing to congestion.

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.