Maintaining Quality of Service with Dynamic Fault Tolerance in Fat Trees

Frank Olaf Sem-Jacobsen[1,2] and Tor Skeie[1,2]

[1]Department of Informatics
University of Oslo
Oslo, Norway

[2]Networks and Distributed Systems
Simula Research Laboratory
Lysaker, Norway

Abstract. A very important ingredient in the computing landscape is Utility Computing Data Centres (UCDCs), large-scale computing system that offers computational services to concurrently running applications. In a UCDC, virtual servers containing a subset of the available resources are dynamically created to fulfil user demands. Typically, each virtual server will have its own service level agreement, which should to the largest extent be unaffected by the behaviour of the all other virtual servers in the system. As UCDC systems increase in size and the mean time between failure decreases, it is becoming an increasingly important challenge to expediently tolerate failures (dynamically), while distributing the effects of the failure amongst the virtual servers according to their service level agreements. In this paper we propose and evaluate a strategy for offering predictable service in fat trees experiencing faults, by reprioritising packets. The strategy is able to distribute the effect of network faults in order to satisfy a number of quality of service demands. These may include guaranteeing that high-priority packets not encountering the fault are unaffected by the fault event, guaranteeing high network throughput for all high-priority traffic, or ensuring that the negative effects of the fault are evenly and fairly spread throughout the network. We find that which demands to favour depends on the computer system and the characteristics of the applications it is running, and that in the presence of a moderate number of faults it is to some degree possible to meet the demands.

1 Introduction

The application of supercomputer systems is increasing. Traditionally there are the"single application" supercomputers where single applications run exclusively for a certain amount of time. More recently, we see the emergence of Utility Computing Data Centres (UCDC) where multiple applications are run in parallel on the same supercomputer, separated into virtual servers. This brings forth the necessity of being able to partition, or virtualise, the supercomputer in order to separate the different applications from each other. In this manner they may run on a dedicated set of resources without interfering with other applications, and receive predictable service. Typically, many network resources must be shared between the applications. To provide quality of service some kind of differentiating between the applications/virtual servers must be undertaken relative to existing Service Level Agreements (SLA). This places severe demands on job scheduling and resource allocation, but also on the interconnection network and routing algorithm used to direct packets through the system. The network must be able to reliably forward packets such that each application is guaranteed a portion of the capacity in the network, even if parts of the network should cease to function for any period of time.

In order to support the emerging requirements, many technologies include additional features such as quality of service (QoS) and fault tolerance. Quality of service enables the network to provide differentiated service to different types of traffic, usually by implementing virtual channels (VC) with different priorities in the network as in Infiniband [3] and Advanced Switching (AS) [9].

There are two basic approaches to provide differentiated service to network traffic. The most fine-grained control and guarantees can be achieved by a mechanism such as IntServ [23]. Resources are reserved for each individual flow in every node the flow passes throughout the network. However, this method introduces some severe scalability issues. These are solved in the DiffServ model [23] where traffic is classified into a moderate number of classes and provisions are made for each of the classes throughout the network. This is the approach best catered for by mechanisms present in today's interconnect architectures. The applicability of this approach in cut-through switched networks like Infiniband and AS is confirmed in [11].

The objective of network fault tolerance is to keep the routing algorithm connected as network faults occur in order to ensure that traffic from any source is able to reach any destination. How this is achieved depends on the network topology, the types of faults that must be considered, and the nature of the network applications which dictates the speed at which the fault tolerance mechanism must tolerate the fault.

As opposed to quality of service which is built into the network technology, fault tolerance is often considered an "add-on" mechanism as it may be achieved by having an appropriate routing algorithm to reconfigure the network upon fault events independently of the interconnect technology. Deadlock freedom is guaranteed either through halting the network and removing all traffic while it is reconfigured, or e.g. reconfiguring the network using a different set of virtual channels and shifting all traffic to these channels once reconfiguration is completed. This is known as static and dynamic *reconfiguration* respectively. Endpoint and local dynamic *rerouting*, on the other hand, preconfigures the network with alternative paths around elements that may fail, either on an end-to-end scale with different paths between source/destination pairs or on a local scale around single elements. It is therefore more closely integrated with the interconnect technology as network endpoints or switches must have the ability to reroute packets around faults. This allows network faults to be tolerated much faster than having to reconfigure network after the fault has occurred. Reconfiguration and rerouting therefore affects traffic in the network in different ways. While reconfiguration affects the entire network and separates the configurations either in time (halting and draining the network) or space (moving the traffic to a different set of virtual channels) and thus affecting all traffic in the network, dynamic rerouting affects only a subset of the traffic, perhaps only the path of a single flow is changed because of a fault.

Although many interconnection networks employ both fault tolerance and quality of service mechanisms, little work has been done on combining those two system demands. A reason for this might be that fault events are rare when compared to the lifetime of network flows. Also, static reconfiguration need not necessarily consider quality of service issues since the entire network, or most of the network, is reconfigured, thus changing the conditions for all traffic in the network. However, as interconnection networks are used in high-speed

systems supporting a heterogeneous set of applications, as is the case with UCDCs, there is a need for local dynamic rerouting algorithms which are able to quickly tolerate the fault while losing as few packets as possible. At the same time it is important that traffic in the network that is not affected by the fault maintains its perceived quality of service, while the traffic affected by the fault receives as good service as possible without degrading the service of other traffic. It is therefore necessary to make the dynamic fault tolerance mechanism QoS-aware.

The purpose of this paper is to propose and evaluate a strategy for maintaining quality of service for flows in the network, both for flows that are affected and unaffected by the fault, while using a local dynamic rerouting algorithm which routes packets around network faults locally. We will evaluate the ability of the strategy to satisfy the various quality of service requirements put forth by UCDC systems, namely ensuring high network utilisation, isolating the effect of faults to the directly affected flows, and preserving quality of high-priority traffic, if necessary at the expense of low-priority traffic. The QoS mechanism under consideration is based on the DiffServ model where each link is divided into several virtual channels with a given priority. We will focus on the fat tree topology [7] since this is a widely used topology for interconnection networks employed in UCDCs.

The rest of this paper is organised as follows. We will first give an overview of previous work in the field of dynamic rerouting fault tolerance algorithms and quality of service in Section 2. We will then describe the local dynamic rerouting algorithm we will make QoS-aware in Section 3, present in our strategy for achieving this in Section 4, and discuss the targets for our quality of service mechanism in Section 5. The strategy will be evaluated in Section 6 and the paper is concluded in Section 7.

2 Previous Work

Both fault tolerance and quality of service have received much attention from the academic world. With regards to fault tolerance much work has been done on improving the fault tolerance of existing network topologies, either by adding extra hardware in terms of switches and links [20], routing the packets through the network in multiple passes [6,4], or combining the two approaches [19]. Similar work has been done on the orthogonal fat tree [21], which is a fat tree variation designed to maximise the number of endpoints, by adding additional links and switches to increase the number of available paths end-to-end [22].

The above approaches only provides reconfiguration or endpoint dynamic rerouting. However, similar techniques may be used to create network topologies supporting local dynamic rerouting. By adding additional links and switches several MIN topologies supporting local dynamic rerouting have been created, e.g. the Quad Tree [17], a modified Omega network [18], and the Siamese-twin fat tree [16].

Recently, a local dynamic rerouting algorithm for fat trees has been developed, both for adaptive [12], deterministic [14], and source routed fat trees [13]. This is well suited for our purpose and is the local dynamic fault-tolerant routing algorithm we will employ in this paper. We will describe the algorithm in greater detail in the next section.

Concerning quality of service, much work has been done on exploring the possibility of the quality of service mechanisms provided by Infiniband. The core of the quality of service mechanism in Infiniband is the arbitration tables dictating how large part of the total link bandwidth each virtual channel may receive. Alfaro et al. [1] propose mechanisms to compute these arbitration tables based on bandwidth requirements, and shows how the tables may be configured to serve time sensitive traffic [2]. This has also been done for advanced switching [8].

Although much work has been done in the two fields of quality of service and fault tolerance separately, to the best of the authors knowledge no work has been published considering the effect of fault tolerance mechanisms on quality of service.

3 Dynamic Fault Tolerance

Before we go into the possible methods of maintaining quality of service with network faults, we must first present the dynamic fault tolerant routing algorithm we will be using and the topology of choice. We will be using a recently developed routing algorithm providing local dynamic rerouting (routing around faulty elements) for fat trees [14]. We give a brief outline of the most important properties of the algorithm, for further details we refer to [14].

The fat tree is a tree topology where processing nodes are connected to the bottom of the tree as leaves, and packets traverse the tree from a processing node upward towards a root in the upward phase, and then downward to their destinations in the downward phase. The distinguishing feature of fat trees is that the aggregate capacity of the switching stages is constant. In other words, the aggregate capacity of the top stage of switches is the same as the bottom stage of switches. Although this can be realised by having increasing link and switch capacities in the higher stages, most practical implementations employ several tree roots to maintain the capacity, e.g. as the k-ary n-tree [10] where k is number of switch ports in the upward or downward direction, and n is the number of switching stages (Figure 1).

Packet forwarding is divided into two phases, upwards and downwards. Packets forwarded in the k-ary n-tree may use any of the upward links to advance towards their destination in the upward phase, but in the downward phase there is only a single deterministic path. Thus, achieving local dynamic rerouting around link faults in the upward phase is trivial: if the original upward link is faulty, simply choose another upward link. In the downward phase we must resort to non-minimal paths if the link towards the destination has failed. A packet encountering a faulty link in the downward phase must choose an alternative downward link not leading towards its destination, to what becomes a *U-turn Switch*. Once this link is traversed normal shortest path routing may commence, first directing the packet upwards one stage from the u-turn switch over a different link from where it arrived, and then downwards. If this downward path is also faulty the packet is returned to the U-turn switch, which may select a different upward link. The upward and downward path following the U-turn switch must take place in a deadlock freedom layer, an additional virtual layer (a specific virtual channel on each link) in the network, to ensure deadlock freedom.

The path of a packet encountering a link fault first in the upward phase and then in the downward phase with a subsequent rerouting is displayed in Figure 1. The bold, dashed

links represents faulty links on the original path, and the unbroken bold links is the actual path of the packet when avoiding the faults. Note that over the two subsequent links from the U-turn switch that are coloured gray, the packet is forwarded in the deadlock freedom layer.

It was shown in [15] that it is possible to achieve $k-1$ dynamic link fault tolerance in a k-ary n-tree with one additional virtual layer in the network for deadlock freedom. In the next section will explore how quality of service guarantees may be maintained when using this algorithm to tolerate multiple link faults.

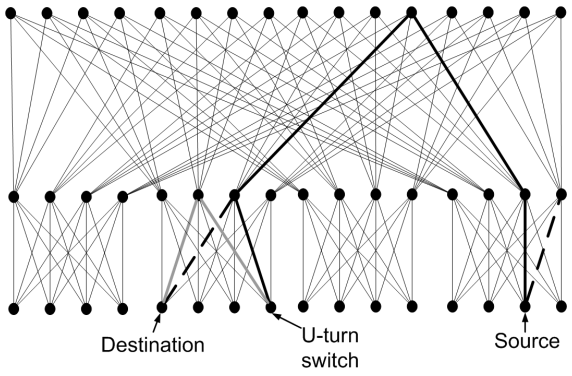


Fig. 1. A fat tree (4-ary 3-tree) consisting of radix 8 switches with two link faults. The faulty link are marked as bold, dashed lines, and the bold line describes the path of a packet from its source to its destination. Note how the packet is misrouted via the U-turn switch and that the two subsequent links are grey indicating that the packet is in the deadlock freedom channels.

4 Maintaining Quality of Service with Dynamic Fault Tolerance

Given the routing algorithm presented in the previous section will now identify at which point is possible or necessary to consider quality of service requirements. Even though we in this section focus on the presented routing algorithm, the strategies we propose and results we gather are applicable to other routing algorithms. Specifically, routing algorithms that rely on extra virtual channels to ensure deadlock freedom will have to take our results into consideration. We assume that each link in the network is configured with two or more virtual channels of different priority, i.e. virtual channels with higher priority is guaranteed a larger portion of the link bandwidth. One method of achieving this is to assign a weight to each virtual channel corresponding to the fraction of the link bandwidth assigned to that channel. An arbitrator may then use the weight of each virtual channel to determine how many packets it may transmit when selected, somewhat similar to how Infiniband works assuming all virtual channels are in the low-priority arbitration table (Infiniband supports two arbitration tables, one low and one high-priority). Virtual channels are usually served in a round-robin fashion.

Each time a packet encounters a faulty link on its path through the network it will be misrouted and thus deviate from its original path. Packets encountering faults in the

upward phase will follow a different path through large parts of the network, while packets encountering a fault in the downward phase will cross three links not part of their original path. In other words, packets encountering faults will add load to other parts of the network, thereby interfering with traffic not directly affected by the fault. This shifting of load may degrade the quality of service experienced both by traffic being misrouted, and the traffic already occupying the links of the new paths. The added load will decrease the achieved throughput of the individual flows, and increase latency because of increased queue lengths.

There is one basic way of handling this misrouted traffic with respect to quality of service, namely change its priority. However, what priority to change to, and when/if to change back to the original priority gives rise to a large number of possible approaches.

There are three scenarios in which it is natural to consider changing the priority of a packet. The first scenario is when a packet encounters a fault. When a packet must be misrouted it may be switched to a lower or higher priority for the time its path differs from its original path, or remain at the original priority. The second scenario is when a packet's misroute path merges with its original path, the path it would have taken if no faults were encountered. At this point it may be switched back to its original priority or continue to its destination using its misroute priority. The third and final scenario is when packets are switched to the deadlock freedom channels. Whereas the two first scenarios are optional, packets encountering faults in the downward phase must necessarily be switched to a deadlock freedom channel. These channels are also required to have a weight in the arbitration tables, and thus a priority. Consequently, packets encountering faults in the downward phase will all be forwarded through at least two virtual channels of the same priority, regardless of whether they are high or low-priority packets in the first place. The priority given to these deadlock freedom channels will therefore have a significant impact on traffic of all priorities. This is further aggravated by the fact that interconnection networks rely on link level flow control with a backpressure mechanism to ensure that no uncorrupted packets are lost. Any slowdown or speed-up of the misrouted packets will consequently affect all packets upstream from the point where the change occurs.

In the next section we will discuss how misrouted traffic might be handled, and which combinations of the different alternatives may yield the desired results.

5 Managing Quality of Service in Supercomputers and UCDCs

The optimal way of distributing the effect of network faults differs depending on the application of the supercomputer. For supercomputers which traditionally run a single application at a time the entire interconnection network is used by the application, with some management traffic which is usually given a higher priority to keep the system operating smoothly. Thus, it is clear that any handling of quality of service around the fault should be designed to maximise network efficiency while maintaining the high priority of the management traffic since all network resources are used towards maximising performance of the single application.

The situation is much more complex in UCDCs since there are several applications competing for the same network resources. We assume that jobs are allocated using virtual channels to achieve separation between the various applications/virtual servers. Each virtual

channel may be assigned a priority to give certain applications higher bandwidth/lower latency access to key resources, for instance, based on how much the customers have paid to run the application (manifested in a SLA). In this context the best way of distributing the effect of network faults is quite different from the single application case. Even though it is beneficial for the system as a whole to maximise in efficiency of the entire network, the consideration for the different applications and their SLAs plays an important role. High-priority jobs should receive maximum priority around the faults at the expense of low-priority jobs. On the other hand, it could be argued that faults occurring in a part of the network primarily used by a single application should only affect that single application, without degrading the service of other applications. However, this may not result in the best overall performance.

The question of how quality of service should be handled when misrouting around network faults becomes a question of how one wishes to distribute the effects of the fault in the network, i.e. slow down a single application significantly versus slowing down multiple applications, but where each application is only marginally affected.

Will now discuss the possible strategies given the available mechanism consisting of changing the priority of misrouted traffic. For simplicity, we assume that we have three priority levels, one high-priority level, one medium-priority level, and one low-priority level. In a UCDC context we may assume that the high-priority levels are used by a high-priority application, and the medium and low-priority level are used by a medium and low-priority application respectively. Traffic in the high-priority level should not be affected by medium or low-priority traffic. The low-priority traffic may be viewed as best effort, without any guarantees and it should not interfere with high or medium-priority traffic. Each of these three levels is mapped to their own virtual channel on every link. Additionally, there is the deadlock freedom channels where all virtual channels have a specific priority/weight which varies depending on the alternative approach under consideration.

This allows us to propose numerous approaches that can be broadly divided into two main categories, namely one where misrouted traffic is given high priority and another where misrouted traffic is given low-priority. Within these two categories there are multiple alternatives for what is done with the packet after being misrouted and the priority of the deadlock freedom channel, but it all comes down to whether the priority of misrouted packets should be increased or decreased.

Decreasing the priority of the misrouted traffic is intuitively a good approach. By decreasing the priority of the traffic being misrouted, its impact on other traffic in the network not affected by the fault is minimal. However, as we will see in the next section, the backpressure nature of interconnection networks will cause this reprioritised traffic to severely impact other traffic in the network. On the other hand increasing the priority of misrouted traffic will ensure that it is expediently handled to possibly make up for the fact that it has a longer path to travel in the presence of the fault, but it might cause a degradation of the service to high-priority traffic.

If we generate all combinations of increasing and lowering packet priority when misrouting upwards and downwards, as well as having high and low-priority deadlock freedom channels and returning to the original priority after the misrouting is complete or continuing to the

destination with the misroute priority, we get 14 possible combinations. In the next section we present and evaluate these 14 different combinations and discuss which one may be most suited to fulfill quality of service demands on the basis of what we have discussed here.

6 Evaluation

To evaluate the behaviour of network traffic of different traffic classes in the presence of link faults we have performed an extensive set of simulations.

6.1 Simulation Parameters

The simulations are performed in a simulator based upon j-sim [5] and developed in-house at Simula Research Laboratory. The network is configured with three traffic classes (TC1, TC2, TC3), each assigned to a virtual channel in the network corresponding to low (VC1), medium (VC2), and high (VC3) priority traffic. Additionally, there is a fourth virtual channel (VC4) for use for deadlock freedom when misrouting around link faults, which will alternately have the same priority as VC1 and VC3 for the different scenarios. The fat tree topology of choice for the simulations is a 4-ary 3-tree, consisting of radix eight switches interconnected in three tiers. This is sufficiently small to allow the simulations to be terminated within reasonable time. Furthermore, when scaling to large network sizes, also of the relative differences of the different approaches may decrease, the overall conclusions will remain the same.

Traffic is generated following a Poisson distribution. The destination address distribution is such that all paths are of equal length when there are no link faults, i.e. all possible destinations for any given source lies in the other half of network, forcing all traffic through the top stage switches. The network is allowed to stabilise before statistics are recorded and faults are introduced. Thereafter the simulations are run for 10 000 simulation cycles.

The relevant setup for the 14 different alternatives is summarised in Table 1. $VC1{-}4$ gives the percent of bandwidth reserved for traffic in the respective VCs, $VC_{misroute}$ determines which VC the packet is moved to when it is misrouted and "How far" describes whether the packet is returned to its original VC after it joins its original path (local), or if it otherwise retains the new VC until the end (to end). Up and Down indicates whether the packet changes VC when it encounters a fault in the upward and downward direction respectively.

The network is configured with three traffic classes and four virtual channels. There is a one-to-one mapping between traffic class and virtual channel which is listed in Table 2, along with the load offered to each traffic class.

6.2 Simulation Results

Following from Section 3 the 4-ary 3-tree is guaranteed to be connected with up to and including three link faults ($k = 4$). In this evaluation we present the difference in throughput and latency for the traffic classes when comparing a fault free network to a network with three link faults. This comparison is done for 4 different load scenarios corresponding to the vertical lines in Figure 2. Figure 2a) shows the throughput (y-axis) of the three traffic

Table 1. A list of the different experiments run.

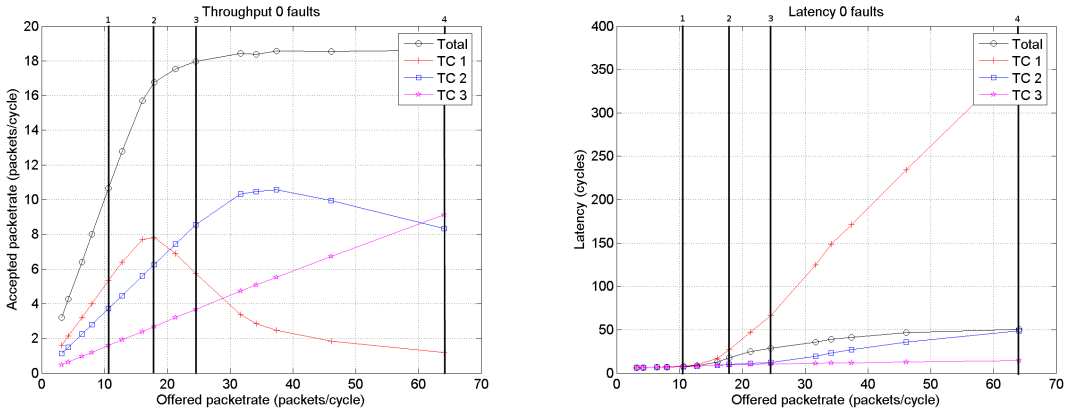
S	VC1	VC2	VC3	VC4	$VC_{misroute}$	How far	Up	Down
1	1	35	54	10	VC1	local	X	X
2	1	35	54	10		local		
3	1	35	54	10	VC3	local	X	X
4	1	20	40	39		local		
5	1	20	40	39	VC1	local	X	X
6	1	20	40	39	VC3	local	X	X
7	1	35	54	10	VC1	to end	X	X
8	1	35	54	10	VC3	to end	X	X
9	1	20	40	39	VC1	to end	X	X
10	1	20	40	39	VC3	to end	X	X
11	1	35	54	10	VC1	local		X
12	1	35	54	10	VC3	local		X
13	1	20	40	39	VC1	local		X
14	1	20	40	39	VC3	local		X

Table 2. The TC - VC relationship

Traffic class	Virtual channel	Percent of total offered load
TC1	VC1	40%
TC2	VC2	35%
TC3	VC3	25%

classes in the fault free network for an increasing traffic injection rate (x-axis). Similarly, Figure 2b) shows the latency (y-axis) for the same simulations. The 4 vertical lines mark the load cases selected for detailed analysis. The figures clearly show how the throughput of the lower priority traffic classes reduces as the network saturates in favour of the high-priority traffic class TC3.

Figures 3 a) and b) show the total throughput and latency respectively for the 14 test cases presented above and the four load cases marked in the previous figure. Every bar in the plot indicates the reduction in throughput when comparing the network with three link faults against the fault free network. The bars closest to us represents load case 1, while the bars furthest back represents load case 4. Note that the base of the bars is at zero at the top, and they stretch downwards indicating the throughput reduction in percent. For load case 1 the throughput reduction is insignificant, but as the load increases the impact of link faults becomes more pronounced. When looking at either of the load cases it is clear that the approaches 4, 5, 6, 9, 10, 13, 14 give similar results and the lowest throughput reduction. The common factor of these approaches is that the priority of the deadlock freedom channel VC4 has the same priority as the high-priority channel VC3. Within this set approach 10 provides the smallest reduction for load case 2, while approach 5 gives the smallest reduction for load cases 3 and 4 (the two saturated cases). Approach 10 gives misrouted packets high priority from the moment they encounter any fault, either upwards or downwards, until they reach their destination. As we will see later on this affects the performance of other high-priority traffic in the network. The other highest performing approach, approach 5, switches misrouted packets to the low-priority VC1 both when encountering fault upwards



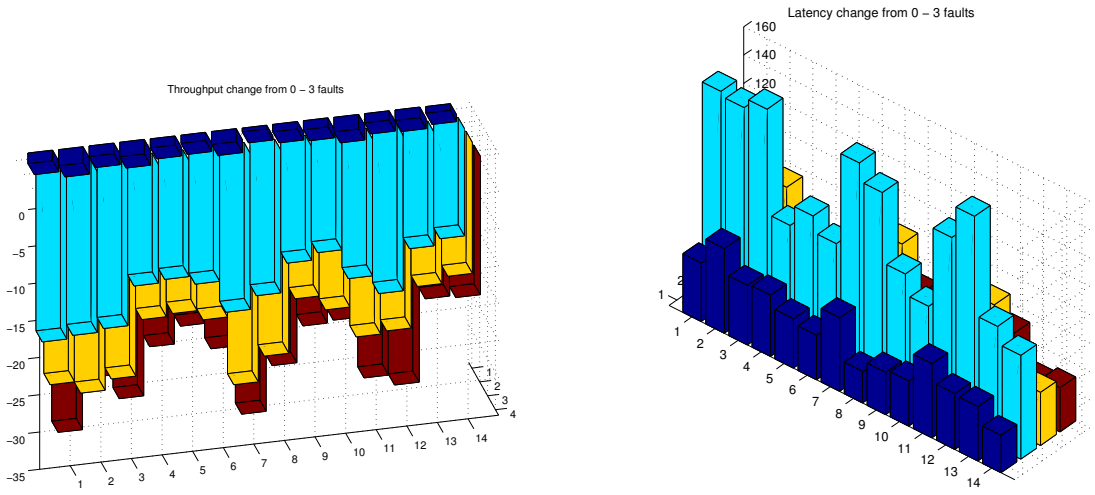
(a) Throughput (b) Latency

Fig. 2. Throughput of the network without faults

and downwards, but they are returned to their original VC when they return to their original path. In this manner, traffic deviating from its original path in any way is given the lowest priority (except when in the deadlock freedom channels) allowing the highest utilisation of the network overall.

The throughput results are mirrored in the latency plot where the approaches utilising a high-priority deadlock freedom channel provide the lowest increase in latency. Note that the bars go from the bottom upwards indicating a latency increase given in percent on the y-axis.

Figure 4 shows the change in latency for several categories of packet flows. Figure 4a) shows the change in latency for high-priority TC3 traffic which does not encounter any faults and does not share links with any other traffic encountering faults. As before, and indeed for all remaining plots, load case 1 is the frontmost row of bars, while load case 4 is the row of bars furthest back. The y-axis is the change in packet latency in percent. The latency for this traffic should ideally not show any change when faults are introduced as this traffic should be unaffected. However, the figure clearly shows that for all approaches the traffic experiences a latency increase. To explain this, recall that interconnection networks employ link level flow control, so should any packet be slowed at any point in the network (due to head of line blocking, etc.) all upstream traffic from this packet using the same VC as it will to some degree be affected through having to wait somewhat longer in some buffers, thus increasing the latency. Hence, even though this traffic does not share any links with packets that are a misrouted somewhere along their path, they will share links with packets that share links with packets directly affected by the fault, creating a complex indirect dependency chain. This effect is clearly visible in the figure as it shows the same distinction between the approaches using a high-priority deadlock freedom channel and those using a low-priority deadlock freedom channel. It is fairly obvious that ensuring expedient handling of misrouted traffic by switching it to high-priority channels is important to maintain high-



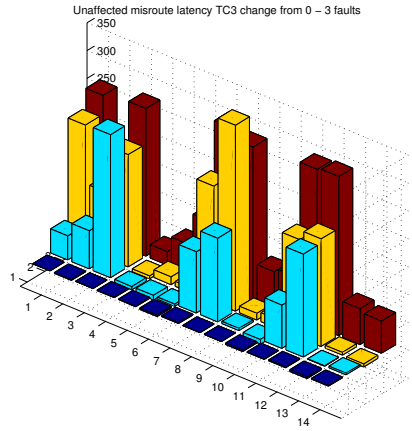
(a) Throughput (b) Latency

Fig. 3. Change in throughput and latency from 0-3 faults for 4 loads

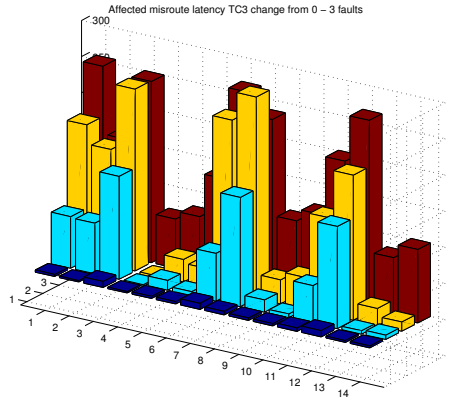
priority guarantees, using a low-priority deadlock freedom layer causes throughput-reducing feedback throughout the network.

There is, however, a trade-off. Transferring all misrouted traffic to the high-priority VC3 has an adverse affect on native TC3 traffic as is evident from the load case 4 where latency increases for approaches 6, 10, and in part 14. This last one, 14, does not show such large increase in latency as the other two, since packets are only switched to the high-priority VC3 when encountering faults in the downward phase, as opposed to both the upward and downward phases in the other two approaches. The optimal solution to this trade-off is only giving high-priority to the deadlock freedom channel VC4, without further changing the priorities of packets, represented by approach 4. In this way, TC3 packets do not get reduced priority when encountering faults and are expediently processed, and low-priority TC1 and TC2 packets do not get increased priority when encountering faults to such a large extent as to reduce the service experienced by TC3 traffic.

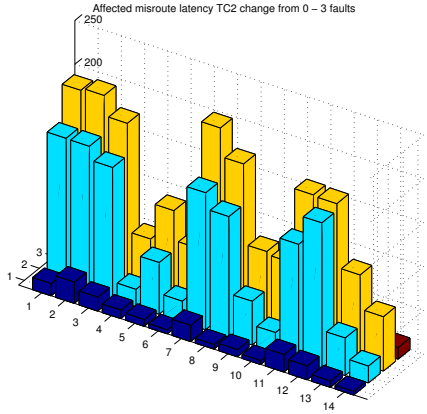
Let us then focus on the change in latency for traffic sharing links with other traffic directly affected by the fault, but not encountering the faults itself, for traffic classes TC3, TC2, and TC1 in figures 4 b,c,d) respectively. For TC3 we see that the latency increases in much the same manner as for traffic not affected by the faults, except that the latency increases are generally a bit larger, especially for load case 1. Comparing this to the latency change for the other two traffic classes, we see that there is a marked difference for the highest load, load case 4. For TC2 traffic the effect of the different approaches are much the same as for TC3, while for TC1 traffic the different approaches have little impact on the latency. The reason for the small change in load case 4 latency for TC2 (the bars are not visible behind load case 3) is that its corresponding virtual channels are heavily saturated and latency is thus given by the buffer sizes and hop count. Increasing the path length by misrouting around faults will therefore not increase latency any more than the buffer time of the extra path length. The same argument is relevant for TC1 traffic which is even more saturated,



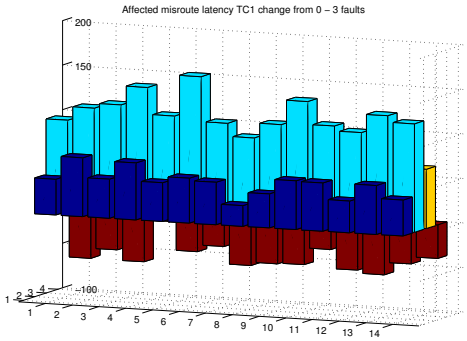
(a) Unaffected, TC3



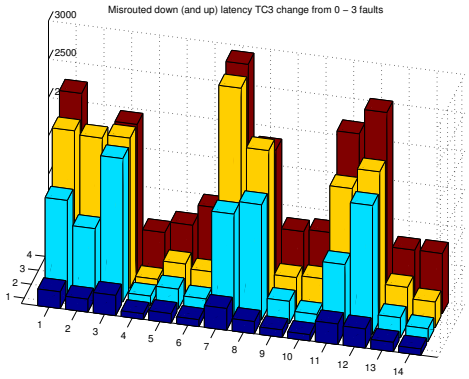
(b) Affected, but not misrouted, TC3



(c) Affected, but not misrouted, TC2



(d) Affected, but not misrouted, TC1



(e) Misrouted in the downward phase, TC3

Fig. 4. Change in latency from 0-3 faults for 4 loads, various traffic types

but there is also an added effect. Even when the deadlock freedom channels are configured with the lowest priority used for these in the simulations, theire weight in the arbitration table is still 10 times higher than for VC1, so any misrouting may actually decrease latency

by allowing it the same priority of all other traffic misrouted around the fault. This effect will to some degree be present for the other loads as well, but it is not as apparent since VC1 is more saturated.

Finally we analyse the performance of TC3 traffic directly affected by the fault through having to be misrouted around failures. This is depicted in figure 4 e). On a general note we see that the latency increase for this traffic is significantly larger than for traffic not directly encountering the faults, with the maximum latency increase close to 3000%. Again we see the same classification as previously, with the approaches with high-priority deadlock freedom channels giving the by far best results. Similarely to the other plots, approach 4 yields the lowest increase in latency also for high-priority traffic directly affected by the fault.

Let us now summarise the results. Intuitively it would seem that giving misrouted traffic low priority in order to minimise its impact on other traffic in the network would yield best results. However, because of the backpressure nature of the flow control mechanism, combined with head of line blocking, reducing the priority of packets over one single link will reduce throughput in large parts of the network. This is evident from the presented results where none of the approaches using a low-priority deadlock freedom channel were able to maintain a modest latency increase in the presence of faults. Furthermore, it seems that the priority of the deadlock freedom channel has the largest impact on latency performance. The other mechanism such as changing the priorities of packets on the entire misroute path has only a minor impact.

The results also show that maintaining high network efficiency generally works well together with maintaining guarantees for high-priority traffic not directly encountering the faults. We are in other words able to effectively isolate the effects of link faults to traffic directly affected by the fault. The other traffic in the network experiences only a moderate service degradation, depending on to which degree it shares its path with misrouted traffic. Configuring the deadlock freedom channel with a high-priority is consequently sufficient to maintain high network utilisation, while at the same time ensuring minimal impact on high-priority traffic in the network.

7 Conclusion

Maintaining quality of service guarantees in the presence of network faults is difficult assuming dynamic fault tolerance, as any fault will move traffic around in the network disrupting the service provided to the applications. This is especially difficult in Utility Computer Data Centre (UCDC) systems consisting of several virtual servers, each with their own Service Level Agreements. We have shown that it is important to consider traffic priorities when configuring a dynamic rerouting fault tolerance mechanism. Lack of consideration for such properties may in the worst-case lead to severely degraded network performance for high-priority traffic with faults in the network. We have presented a strategy for reprioritising traffic encountering network faults and evaluated a large number of combinations of the possibilities within this strategy. We found that by correctly changing priorities of traffic that is affected by the fault it is to large degree possible to satisfy the same Service Level Agreements

as before the fault occurred. However, finding the ultimate configuration is difficult since requirements of the solutions vary greatly between applications. We have demonstrated that we are able to satisfy a large number of these requirements, and surprisingly most of the requirements could be satisfied by giving misrouted traffic higher priority, even for maintaining high-priority guarantees for high-priority traffic.

References

1. F. J. Alfaro, Jose L. Sanchez, José Duato, and Chita R. Das. A strategy to compute the infiniband arbitration tables. *Proceedings of International Parallel and Distributed Processing Symposium*, April 2002.
2. Francisco J. Alfaro, Jose L. Sanchez, and José Duato. A strategy to manage time sensitive traffic in infiniband. *Proceedings of Workshop on Communication Architechture for Clusters (CAC)*, April 2002.
3. InfiniBand Trade Association. *InfiniBand Architecture. Specification Volume 1. Release 1.0a.* Available at http://www.infinibandta.com, 2001.
4. Suresh Chalasani, C.S. Raghavendra, and Anujan Varma. Fault-tolerant routing in MIN based supercomputers. In *Supercomputing '90: Proceedings of the 1990 conference on Supercomputing*, pages 244–253. IEEE Computer Society Press, 1990.
5. J-sim. http://www.j-sim.org/, May 2006.
6. Tsern-Huei Lee and Jin-Jye Chou. Some directed graph theorems for testing the dynamic full access property of multistage interconnection networks. *IEEE TENCON*, 1993.
7. C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.
8. Raul Martinez, Francisco J. Alfaro, and Jose L. Sanchez. Decoupling the bandwidth and latency bounding for table-based schedulers. In *Proceedings of the 2006 International Conference on Parallel Processing*, pages 155–163, 2006.
9. David Mayhew and Venkata Krishnan. Pci express and advanced switching: Evolutionary path to building next generation interconnects. In *11 th Symposium on High Performance Interconnects*. IEEE, 2003.
10. Fabrizio Petrini and Marco Vanneschi. K-ary N-trees: High performance networks for massively parallel architectures. Technical Report TR-95-18, 15, 1995.
11. Sven-Arne Reinemo, Tor Skeie, and Olav Lysne. Applying the diffserv model on cut-through networks. *Proceedings of the 2003 International Conference of Parallel and Distributed Processing Techniques and Applications*, 2003.
12. F. O. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato. Dynamic fault tolerance with misrouting in fat trees. In Wu chi Feng, editor, *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 33–45. IEEE Computer Society, 2006.
13. Frank Olaf Sem-Jacobsen, Olav Lysne, and Tor Skeie. Combining source routing and dynamic fault tolerance. *Submitted to ICPP*, 2006.
14. Frank Olaf Sem-Jacobsen, Tor Skeie, and Olav Lysne. A dynamic fault-tolerant routing scheme for fat-trees. *In Proceedings of the PDPTA conference*, 2005.
15. Frank Olaf Sem-Jacobsen, Tor Skeie, and Olav Lysne. Dynamic fault-tolerance in multistage interconnection networks. *Submitted to IEEE Transactions on Computers*, October 2007.
16. Frank Olaf Sem-Jacobsen, Tor Skeie, Olav Lysne, Ola Tørudbakken, Eivind Rongved, and Bjørn Johnsen. Siamese-twin: A dynamically fault tolerant fat tree. *Proceedings of the 19th IPDPS*, 2005.
17. Jyotsna Sengupta and P.K. Bansal. Fault-tolerant routing in irregular MINs. In *TENCON '98. 1998 IEEE Region 10 International Conference on Global Connectivity in Energy, Computer, Communication and Control*, volume 2, pages 638–641, December 1998.
18. Jyotsna Sengupta and P.K. Bansal. High speed dynamic fault-tolerance. In *Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology, 2001. TENCON.*, volume 2, pages 669–675, 2001.
19. N.K. Sharma. Fault-tolerance of a MIN using hybrid redundancy. In *Proceedings of the 27th Annual Simulation Symposium*, pages 142–149, April 1994.
20. T. Skeie. A fault-tolerant method for wormhole multistage networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, pages 637–644, 1998.
21. M. Valerio, L. E. Moser, and P. M. Melliar-Smith. Recursively scalable fat-trees as interconnection networks. *Proceeding of 13th IEEE Annual International Phoenix Conference on Computers and Communications*, 1994.

22. M. Valerio, L.E. Moser, and P.M Melliar-Smith. Fault-tolerant orthogonal fat-trees as interconnection networks. *Proceedings 1st International Conference on Algorithms and Architectures for Parallel Processing*, 2:749–754, 1995.
23. Xipeng Xiao and Lionel M. Ni. Internet qos: A big picture. *IEEE Network*, pages 8–18, March 1999.