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.

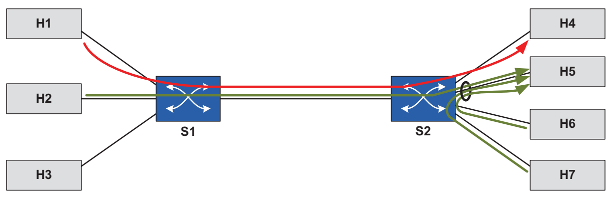


Figure 1. Congestion in an interconnection network.

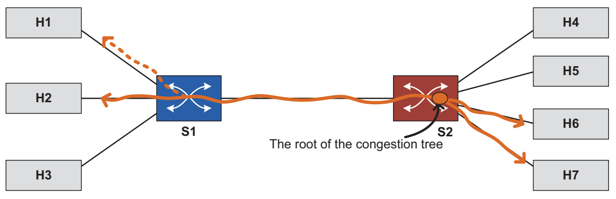


Figure 2. A congestion tree in an interconnection network.

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.

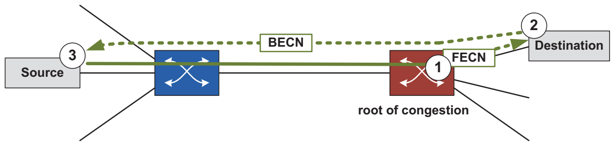


Figure 3. Congestion control in InfiniBand.

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.

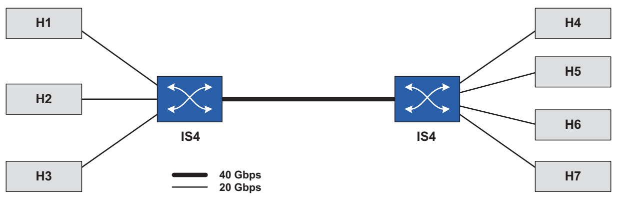


Figure 4. The test bed.

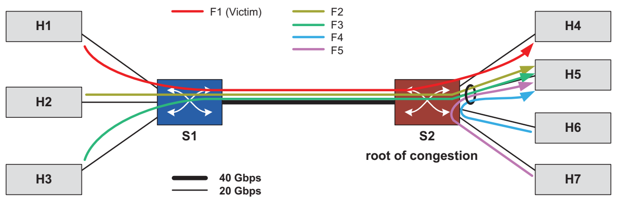


Figure 5. Flow configuration in scenario 1.

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.

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

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

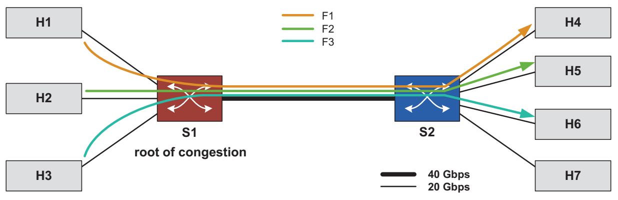


Figure 6. Flow configuration in scenario 2.

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.

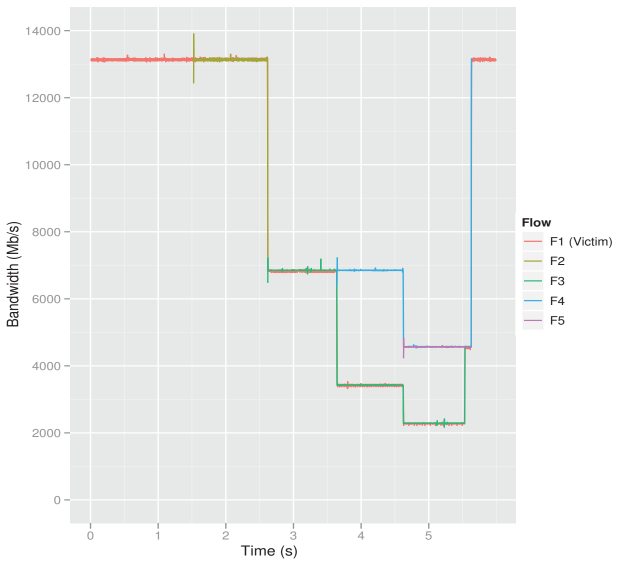
The addition of flow *F2* does not affect *F1* as the two flows only share the link between the two switches, a link with twice the capacity of the switch-to-host links. Therefore they both achieve a throughput of 13 Gbit/s. Now adding the flow *F3*, we observe a major drop in throughput for all three 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.

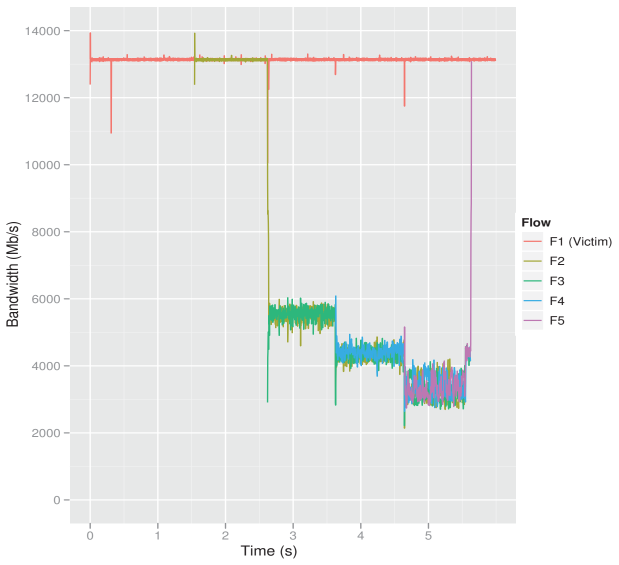
Adding the last flow, *F5*, the same pattern repeats. *F1* (victim), *F2* and *F3* is reduced to approximately 2 Gbit/s, while *F4* and *F5* is reduced to approximately 4.5 Gbit/s. Again, *F1* suffers even more from the HOL blocking, while we still see an unfairness among the flows headed for H5.

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.

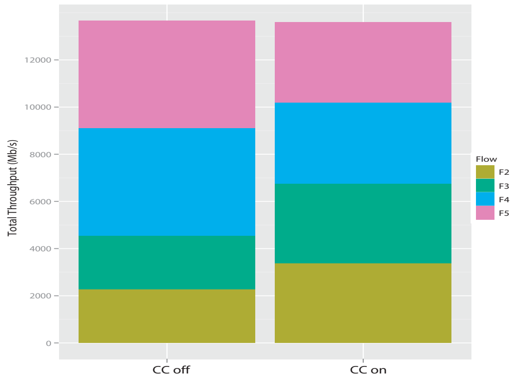


Figure 8. Fairness between the flows contribution to congestion.

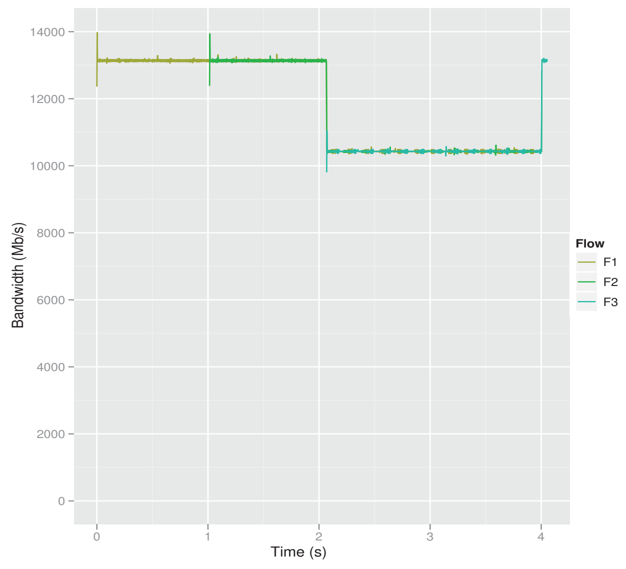
share the last 1/3 of the capacity. This skew is shown in the first bar in fig. 8. The CC treats all contributors to congestion in a fair way. If one contributor occupies more than its fair share of the congested resources at the root of the congestion tree, it will receive a correspondingly high share 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.

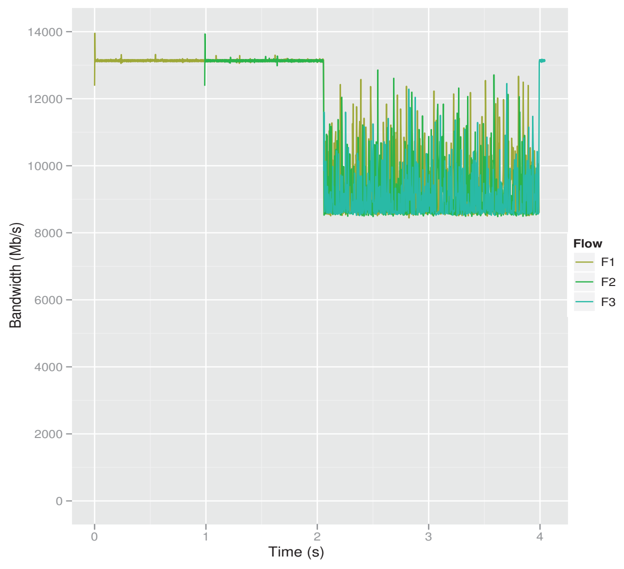
B. *Results from Scenario 2*

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.

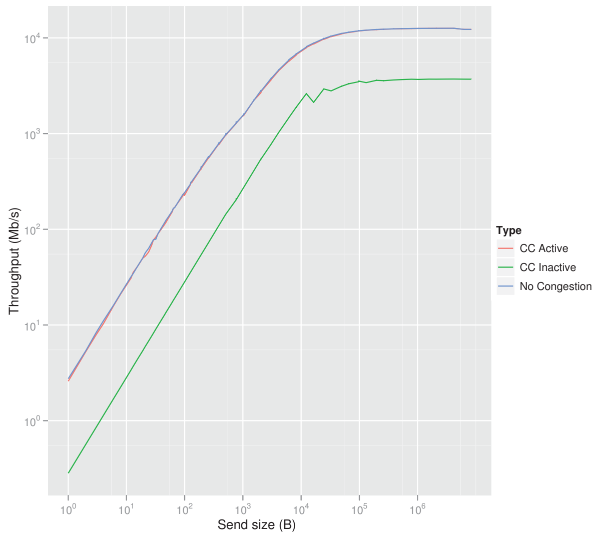


Figure 9. Throughput of the victim flow as a function of packet size.

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

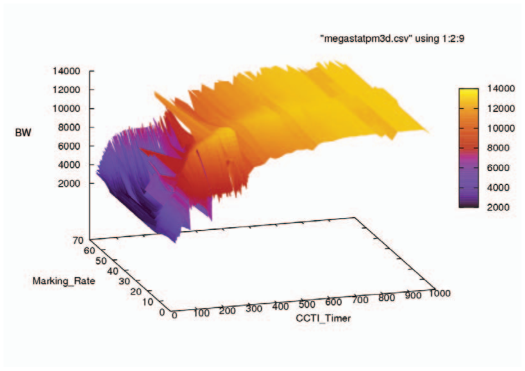


Figure 11. Average throughput of the *victim* flow as a function of the $CCTI\_Timer$ and the $Marking\_Rate$.

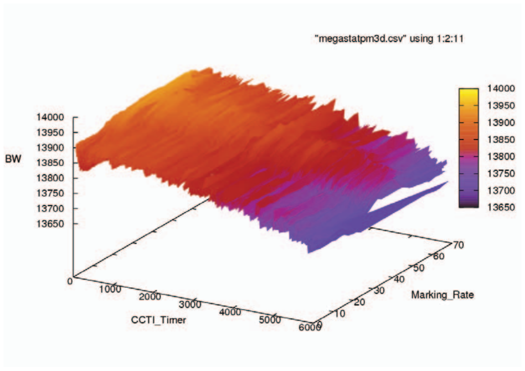


Figure 12. Average throughput of the four contributors to congestion as a function of $CCTI\_Timer$ and $Marking\_Rate$.

$CCTI\_Index$ 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].

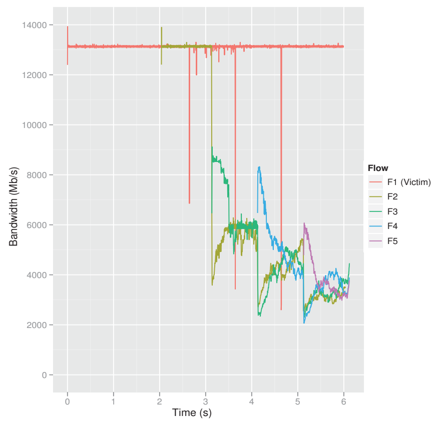


Figure 13. The scenario 1 experiment with a $CCTI\_Timer$ value of 2000 μs.

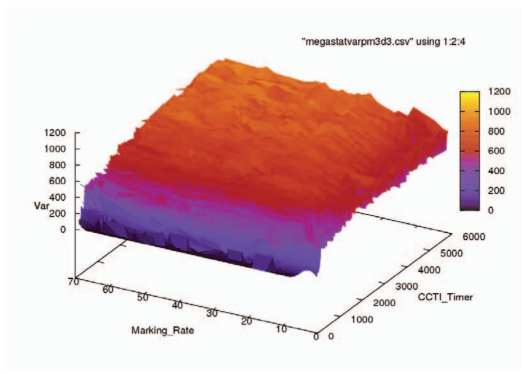


Figure 14. The *treatment variation variable* Var as a function of $CCTI\_Timer$ and $Marking\_Rate$.

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/.