DOI: 10.14529/jsfi160201

A Dynamic Congestion Management System for InfiniBand Networks

Fabrice Mizero[1]***, Malathi Veeraraghavan***[1]***, Qian Liu***[2]***, Robert D. Russell***[2]***, John M. Dennis***[3]

While the InfiniBand link-by-link flow control helps avoid packet loss, it unfortunately causes the effects of congestion to spread through a network. Even flows that do not pass through congested ports can suffer from reduced throughput. We propose a Dynamic Congestion Management System (DCMS) to address this problem. Without per-flow information, the DCMS leverages performance counters of switch ports to detect onset of congestion and determines whether-or-not victim flows are present. The DCMS then takes actions to cause an aggressive reduction in the sending rates of congestion-causing (contributor) flows, if victim flows are present. On the other hand, if there are no victim flows, the DCMS allows the contributor to maintain high sending rates and finish as quickly as possible. The value of dynamic management of a switch congestion-control parameter called `Marking_Rate`, which is responsible for how quickly contributor flows can be throttled, is evaluated in an experimental testbed. Our results show that dynamic congestion management can enable a network to serve both contributor flows and victim flows effectively. The DCMS solution operates within the constraints of the InfiniBand Standard.

Keywords: InfiniBand, Congestion control, Link-by-link flow control, Cascading rate reductions, Dynamic parameter setting.

Introduction

InfiniBand (IB) is widely used in high performance computing (HPC) systems. Among other factors, InfiniBand owes its growing adaptation to its high link rates, low latency and low packet loss. The InfiniBand protocol specification supports a credit-based link-by-link flow control to avoid packet loss and a congestion control system based on explicit congestion notifications.

As noted in prior work [1], the presence of link-by-link flow control causes the effects of congestion to spread backwards in the network. When an output port P1 of a switch becomes congested, the input-side buffer of another port P2 on the same switch, through which a congestion-causing (contributor) flow enters the switch, will fill up causing a reduction in the rate at which flow-control credits are granted by port P2 to port P3 of an upstream switch. This causes a reduction in the effective rate of port P3. Such a rate reduction could cascade backwards and reduce the effective rates of many ports. Bulk-data flows passing through victim ports (ports with reduced effective rates) become victim flows (suffer reduced throughput), even though their own paths do not traverse the congested port. The problem statement of this work is to address the spreading effects of congestion.

We propose a dynamic Congestion Management System (DCMS) that (a) monitors switch-port counters to determine if victim flows have been created by a congestion event, and (b) if there are victim flows, the DCMS dynamically modifies a switch congestion-control parameter to dissipate the congestion event rapidly. A key consideration is the tradeoff between the creation of victim flows vs. a reduction in the throughput of contributor flows.

Experiments were conducted on a two-switch, multi-host InfiniBand testbed. First, the impact of switch congestion-control parameters were studied to determine the default settings

[1]University of Virginia, Charlottesville, USA; {fm9ab, mv5g}@virginia.edu
[2]University of New Hampshire, Durham, USA; {qga2, rdr}@unh.edu
[3]National Center for Atmospheric Research, Boulder, USA; {dennis}@ucar.edu

that would allow contributor flows to enjoy high throughput as long as no victim flows are created. Next, our DCMS proof-of-concept prototype was executed and a switch congestion-control parameter was modified dynamically in a manner that caused senders of contributor flows to reduce their packet injection rates if the DCMS detected the presence of victim flows. When victim flows and/or contributor flows that created victim ports end, or a duration threshold is crossed, the DCMS resets the switch congestion-control parameter back to its default setting.

The novelty of this work lies in our proposal of a dynamic congestion management system (DCMS). The solution is InfiniBand compliant in that the DCMS works in conjunction with off-the-shelf switches requiring no modifications. The importance of dynamic parameter control to enable the network to serve both contributor and victim flows is demonstrated through experiments.

Section 1 offers the reader background on InfiniBand flow control and congestion control. Section 2 describes the spreading effects of a congestion event using a new approach based on a concept of cascading rate reductions. Section 3 describes the DCMS algorithm. Section 4 describes our experiments with a DCMS prototype. Related work is reviewed in Section 5, and Section 5 presents our conclusions.

1. Background

The InfiniBand protocols [2] include link-layer flow control and transport-layer congestion control. Link-layer flow control is used to ensure 0 packet loss due to buffer overflows. A transmitter is permitted to send packets onto a link only when it has received sufficient credits to do so from the receiving end of the link. A Flow Control Packet (FCP) is sent from the receiving side to the transmitting side of a link to explicitly provide information on the amount of space left in the receive buffer.

The transport-layer congestion control is based on Explicit Congestion Notification (ECN), wherein once congestion is detected at a switch, the contributing sources are notified to reduce their packet injection rates. Coordinated actions are required at the (i) switch that detects congestion on one of its ports, (ii) destination Host Channel Adapters (HCAs) of flows that traverse the congested port, and (iii) source HCAs of those flows.

Specific details of how a switch decides that one of its ports is congested are left to vendor implementation. In one approach described by Gran and Reinemo [3], when the fill-ratios of input-port buffers holding packets destined to a particular output port exceed a set threshold, the switch will consider the output port to be congested. A parameter called `Threshold` controls how quickly a switch reacts to congestion, with a value 15 indicating the fastest reaction to congestion onset, and a value 0 for disabled congestion control. The switch then sets a bit called Forward ECN (FECN) in the transport-layer header to 1 for a fraction of the packets transmitted onto the output port. The value of the fraction is determined by a configurable switch parameter called `Marking_Rate`. The `Marking_Rate` is the mean number of unmarked packets sent between consecutive marked packets, where "marking" refers to the setting of the FECN bit. Therefore, the higher the `Marking_Rate`, the lower the rate of generation of FECNs.

When a destination HCA receives a marked packet, the HCA sets the Backward ECN bit (BECN) in an Acknowledgment (ACK) or a data-carrying packet sent from the destination to the source, or the destination HCA generates an explicit Congestion Notification Packet (CNP) to the source of the flow.

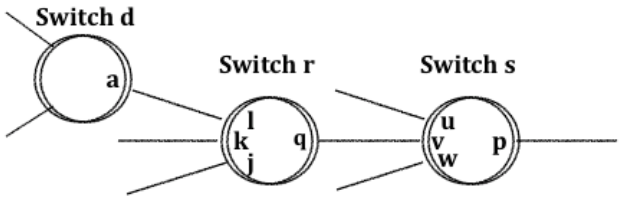


Figure 1. Illustrative InfiniBand network

When a source HCA receives a BECN-marked or CNP for a flow, the source HCA reduces the packet injection rate for that particular flow in the following manner. One Congestion Control Table (CCT) is maintained per port, and one CCT Index (CCTI) can be maintained per flow (queue-pair) or Service Level (SL). A queue-pair is comparable to TCP source and destination port numbers, while SL is a parameter that allows applications to indicate the desired service type. Each entry in the CCT specifies a packet injection delay number, which means that the injection delay between packets for a flow is determined by the CCTI associated with the flow. The HCA maintains two configurable parameters, `CCTI_Increase` and `CCTI_Timer`, per port per SL. When a source HCA receives a BECN-marked packet or CNP from the destination for a particular flow, it increases the CCTI of the flow by the `CCTI_Increase` value for the corresponding port/SL. On every reset of the `CCTI_Timer`, the CCTI of all flows associated with that port/SL are decreased by 1.

In summary, multiple configurable parameters in switches and in HCAs jointly determine how quickly congestion is detected, and how quickly the sending rates of contributor flows are throttled or restored. Throttling contributor flows could have a positive impact on victim flows. On the other hand, throttling contributor flows could have the negative impact of lowering the throughput of these flows.

The purpose of this work is to develop and evaluate schemes that can manage the above-described tradeoff through dynamic modifications to the Congestion-Control (CC) parameters at switches. The only HCA CC parameter that is set on a per-flow basis is CCTI, and therefore it could potentially be modified dynamically. However, it is complex to deploy an external management server that can determine per-flow characteristics and then take actions to dynamically modify the CCTI of a flow at its source. In IP networks, NetFlow and other similar mechanisms are built into routers to reconstruct flow characteristics from sampled or unsampled packets. To our knowledge, there is no comparable feature in InfiniBand switches, which makes it more complex to develop external solutions for flow reassembly.

2. Causes, modes, and effects of congestion

This section describes the causes of congestion, two different modes of congestion, and the effects of congestion.

Congestion occurs at a port p of a switch s when the aggregate arrival rate of packets destined to port p exceeds its capacity. This is the main *cause* of congestion, and port p of switch s is referred to as the *root port* of a congestion event. For example, consider the network shown in Fig. 1. If the aggregate rate of packets arriving at ports u, v, and w, and destined to port p of switch s, exceeds the capacity of port p, then port p will enter a state of congestion.

Formally, we say that port p of switch s is congested at time t if

$$\sum_{v \in \mathbf{P}_s} f_{svp}(t) > C_{sp}, \tag{1}$$

where $f_{svp}(t)$ is the rate of arrival of packets at port v of switch s that are destined to port p of switch s, $\mathbf{P}_s$ is the set of all ports of switch s, and C_{sp} is the link capacity of port p of switch s. When congestion occurs at port p of switch s, input-side buffers of ports with incoming packets that are destined to port p will start filling up. If the configured thresholds are crossed the switch will detect the congestion event and take the actions described in Section 1.

There are two *modes* of congestion as illustrated by an example in the network of Fig. 1. Assume that the port p of switch s enters the state of congestion. If there is a flow that traverses ports j and q of switch r and ports v and p of switch s, then the buffer on the incoming side of port v of switch s will start to fill up, if the instantaneous arrival rate of packets on this flow exceeds the share of the congested-port p bandwidth available to this flow. When this buffer fills up, the rate at which FCPs are generated by port v of switch s to port q of switch r to offer the latter credits for packet transmission will decrease. Effectively, the rate of port q of switch r becomes reduced.

Formally, if port p of switch s enters congestion at time t, then at time $t + \varepsilon$, where ε is a small interval (on the order of nanoseconds in high-rate InfiniBand links),

$$R_{rq}(t + \varepsilon) < C_{rq}, \tag{2}$$

where $R_{rq}(t + \varepsilon)$ is the rate at which data is sent by the transmitter at port r of switch q at time $(t + \varepsilon)$, and port q of switch r is connected to some port v of switch s. Thus, in our usage, "rate" of a port is time-variant, while "capacity" of a port is time-invariant.

A reduction in the sending rate of a switch port can have cascading effects at neighboring switches. In the above example, the presence of a flow traversing port q of switch r and the congested port p of switch s is the cause of rate reduction of port q of switch r. Further, if a flow enters switch r at port l and is destined for port q, then packets from this flow will be served at a lowered rate by port q. Consequently, the input buffer at port l could fill up, causing FCPs to grant credits at a lowered rate to port a of switch d. This causes a rate reduction of port a of switch d. Events that cause these types of "Cascading Rate Reductions" are referred to as *CRR congestion events*. Ironically, this kind of cascading rate reductions occurs because of IB's zero-packet loss policy, which is enforced by the link-by-link flow control. In Ethernet networks, as there is no link-by-link flow control, a transmitter can simply send packets. If a switch buffer is full, packets will simply be dropped. Therefore, a congestion event is handled locally, and its effects do not spread to other switches, as with cascading rate reductions in InfiniBand networks.

On the other hand, if there was no flow passing through neighboring switch ports that also pass through the congested port, then the congestion event could be localized in that all rate reductions are limited to HCA ports. If switch ports are not affected, CRRs will not occur. For example, assume that three hosts are connected to ports u, w, and p of switch s. If the hosts connected to u and w initiate flows destined to the host connected to port p, then port p could get congested. This may in turn cause FCPs to limit credits to the HCAs at hosts connected to ports u and w. However, the rate reductions of these HCA ports cannot propagate to other ports, and therefore, we refer to this mode of congestion as *localized*.

The main *effect* of congestion is the creation of victim flows. A "victim flow" is a flow whose path does not traverse a congested port, and yet (i) does not enjoy the full spare capacity on its path, or (ii) its packets are subject to additional delays. Flows that share one or more links with contributor flows can become victim flows. Consider a flow V in our above example that traverses ports k and q of switch r and ports v and w of switch s. Assuming all links in our example network are Single Data Rate (SDR) links (i.e., 10 Gbps), but the inter-switch link is a Quad Data Rate (QDR) link (i.e., 40 Gbps), the flow V should enjoy SDR rate, if there are no other flows sharing its links. However, because the rate of the QDR inter-switch link will be determined by the rate at which FCPs grant credit to port q of switch r to send packets, the rate of flow V will be limited to this FCP-dictated rate rather than the spare capacity on its path. Hence flow V could become a victim flow.

In summary, congestion is *caused* by a packet arrival rate that exceeds link capacity. There are two *modes* of congestion: CRR and localized congestion. Both modes of congestion could have the *effect* of creating victim flows, which could be delay-sensitive flows or bulk-data flows.

3. Dynamic Congestion Management Solution (DCMS)

Table 1. Notation

Symbol	Meaning
s	switch index
p	port index
$\mathbf{S}$ and $\mathbf{P}_s$	Set of all switches, and all ports on switch $s \in \mathbf{S}$, respectively
$\mathbf{H}$ and $\mathbf{P}_h$	Set of all hosts, and all HCA ports on host $h \in \mathbf{H}$, respectively
$\mathbf{I}$ and $\mathbf{J}$	Set of all switch ports $\mathbf{I} = \bigcup_{s \in \mathbf{S}} \mathbf{P}_s$ and set of all host ports $\mathbf{J} = \bigcup_{h \in \mathbf{H}} \mathbf{P}_h$
$N : (\mathbf{S} \times \mathbf{I}) \rightarrow \{(\mathbf{S} \times \mathbf{I}) \cup (\mathbf{H} \times \mathbf{J})\} \cup \emptyset$	Mapping function that shows the neighbor switch/host and port to which each switch's port is connected, if present; $N(s,p) =$ null if $p \notin \mathbf{P}_s$
$W(s,p,t)$	PortXmitWait counter value of port $p \in \mathbf{P}_s$ at time t
$D(s,p,t)$	PortXmitData counter value of port $p \in \mathbf{P}_s$ at time t
$C(s,p,t)$	PortXmitCongTime counter value of port $p \in \mathbf{P}_s$ at time t
$\Delta X(s,p,t)$	$X(s,p,t) - X(s,p,(t-\tau))$, where $X \in \{W,D,C\}$, where τ is the inter-sweep interval
$\mathbf{V}(s,p)$	Set of victim ports (r,q) for which $\{(s,p),(r,q)\} \in \mathbf{O}_1$
$\mathbb{M}(s,p)$	`Marking_Rate` of port $p \in \mathbf{P}_s$ of switch s; {`Low`, `Default`, `High`}
$\mathbb{S}(s,p)$	State of $p \in \mathbf{P}_s$ of switch $s \in \mathbf{S}$; {`Low-MR`, `Default-MR`}
$\mathbb{I}(s,p)$	Interval count for low value of $\mathbb{M}(s,p)$
T_C	Congestion threshold
T_W	Rate-reduction threshold
T_D	Utilization-change threshold
T_I	Low-MR (`Marking_Rate`) duration threshold

We propose a Dynamic Congestion Management System (DCMS) to modify the `Marking_Rate` parameter in switches to cause reductions in the sending rates of contributor flows if there are victim flows. If there are no victim flows, contributor flows are allowed to send at high rates so that they can finish their transfers quickly to avoid creating victim flows.

The DCMS determines whether-or-not victim flows have been created by a congestion event by reading values of three types of switch port counters, namely, `PortXmitCongTime`, `PortXmitWait`, and `PortXmitData`. `PortXmitCongTime` is the amount of time a port has spent in a congested state. `PortXmitWait` indicates the amount of time a port has data to send but lacks flow-control credits. `PortXmitData` is the amount of data transmitted on the port (in 32-bit words). Specifically, DCMS uses the `perfquery` tool to gather values of above stated counters. Since `perfquery` uses General Management Packets (GMPs) that are subjected to flow control, DCMS uses a separate SL for these packets so that they are given higher priority than user-data packets, and not subject to congestion control.

The default settings for switch CC parameters are presented first, and then an algorithm for making dynamic modifications is described.

3.1. Default values for switch parameters

To begin with, we recommend that the CC `Threshold` in all switches be set to 15 [4], so that even at the slightest hint of congestion on one of its ports, a switch will start increasing the corresponding `PortXmitCongTime` counter, allowing the DCMS to react in a manner that prevents or mitigates the problem of cascading rate reductions.

We recommend setting the switch `Marking_Rate` to a high value so that few FECNs and corresponding BECNs are sent when a congestion event occurs. The fewer the BECNs, the smaller the injection rate reduction at the sending HCA. In other words, when `Marking_Rate` is high, contributor flows will continue to send data at high rates. The rationale is that the DCMS will determine whether a congestion event has created victim flows using the algorithm described in the next section, and if there are victim flows, the DCMS will reduce `Marking_Rate` to create more FECNs and BECNs, which in turn will cause contributor flows to throttle their injection rates. On the other hand, if the congestion event does not cause any victim flows, then the DCMS will allow the contributor flows to enjoy high throughput by not changing the `Marking_Rate` from its default high setting. The sooner a contributor flow ends, the lower the probability of it causing a victim flow.

3.2. Algorithm for dynamic modification of `Marking_Rate`

Algorithm 1 Dynamic Congestion Management

1: Read switch counters and compute $\Delta W(s,p,t), \Delta C(s,p,t), \Delta D(s,p,t), \forall s \in \mathbf{S},$ and $p \in \mathbf{P}_s$ in each sweep
2: Call Algorithm 2
3: Call Algorithm 3
4: Sleep until sweep timer τ expires; **go to** 1

The basic concept of the algorithm is as follows. If a congestion event causes a rate reduction in a neighboring switch port, the `Marking_Rate` parameter of the congested (root) port is lowered significantly so that the senders of contributor flows throttle their sending rates aggressively and congestion dissipates quickly. The DCMS monitors the affected neighboring switch port(s) to check if their link rates recovered after the lowering of the root-port `Marking_Rate`. If such a recovery is in evidence on even one affected neighboring switch port, the root-port `Marking_Rate` is kept low. A count is maintained for the number of inter-sweep intervals for

Algorithm 2 Check for newly congested ports and new victim ports

for each port (s, p) where $s \in \mathbf{S}$, $p \in \mathbf{P}_s$ **do**,
  if $(\Delta C(s,p,t) > T_C) \vee ((N(s,p) \in \mathbf{J}) \wedge (\Delta W(s,p,t) > T_W))$ **then** ▷ Is the port congested?
    for each port (r, q) where $(r \in \mathbf{S}, q \in \mathbf{P}_r, N(r,q) = (s,v), v \in \mathbf{P}_s)$ s.t. $(r,q) \notin \mathbf{V}(s,p)$ **do**
      if $\Delta W(r,q,t) > T_W$ **then** ▷ Is there a victim port?
        Add (r, q) to set $\mathbf{V}(s,p)$ ▷ Add port to set of victim ports for the congested port
        if $\mathbb{S}(s,p) \neq$`Low-MR` **then**
          ▷ Another victim port could have previously caused this state change
          $\mathbb{M}(s,p) \leftarrow$ `Low` ▷ A low setting will lead to a reduction in sending rates of congestion-causing flows
          $\mathbb{S}(s,p) =$`Low-MR`
          $\mathbb{I}(s,p) \leftarrow 1$ ▷ Interval count to limit maximum duration for low `Marking_Rate` setting
        end if
      end if
    end for
  end if
end for

Algorithm 3 Monitor ongoing CRR congestion events and restore default operation

for each port (s, p) for which $(|\mathbf{V}(s,p)| \neq 0) \wedge (\mathbb{I}(s,p) \neq 1)$ **do**
  Increment $\mathbb{I}(s,p)$ by 1
  if $(\mathbb{I}(s,p) \leq T_I)$ **then** ▷ Low-marking-rate maximum duration not yet reached
    for each port $(r, q) \in \mathbf{V}(s,p)$ **do**
      if $(\Delta D(r,q,t-\tau) - \Delta D(r,q,t)) > T_D$ **then**
        ▷ Link utilization dropped signaling absence or completion of victim flows
        Remove port (r, q) from $\mathbf{V}(s,p)$
      end if
    end for
  end if
  if $(|\mathbf{V}(s,p)| == 0) \vee (\mathbb{I}(s,p) == T_I)$ **then**
    ▷ No more victim ports, or Low-marking-rate duration threshold is reached
    $\mathbb{M}(s,p) \leftarrow$ `Default`
    $\mathbb{S}(s,p) =$`Default-MR`
  end if
end for

which the `Marking_Rate` is kept low, and a low-MR (`Marking_Rate`) duration threshold is used to limit the maximum number of intervals for which the congested-port `Marking_Rate` is kept low. Given the dynamic nature of flows, the DCMS cannot know for certain whether-or-not all victim flows have ended, and therefore it restores the root-port `Marking_Rate` to its high default value when the Low-MR duration threshold is crossed. If some of the victim flows are still ongoing, a second cycle of low `Marking_Rate` and congestion-event monitoring will be started. The cycle will be repeated multiple times if needed.

The DCMS procedure is described in pseudode in Algo. 1. Periodically, the DCMS reads the three types of switch-port counters described earlier. This operation is referred to as a "sweep." For simplicity, Line 1 states that the counters are read for all ports of all switches (see Table 1 for notation), but in practice, the DCMS can build up historical information on ports that experience congestion, and limit its reading of the `PortXmitCongTime` to just those ports that suffer from congestion. Line 1 of Algo. 1 shows that changes in `PortXmitCongTime` (ΔC), `PortXmitWait` (ΔW), and `PortXmitData` (ΔD) counters are computed by the DCMS. Algo. 1 then calls Algo. 2 and Algo. 3 as shown in the pseudocode. Algo. 1 is executed after each sweep, which is conducted at time intervals of τ, a configurable parameter.

Algo. 2 identifies congested ports, and then examines the `PortXmitWait` counter of ports in neighboring switches to determine if the congestion event is localized or a CRR event. Multiple ports in neighboring switches could suffer from a rate reduction due to a congestion event, i.e., there could be multiple victim ports for a single congestion event. Therefore a set $\mathbf{V}(s,p)$ is created to store the identifiers of the victim ports caused by congestion of root port p of switch s (henceforth the notation (s,p) will be used to denote this port). As soon as the first victim port is identified, the `Marking_Rate` $\mathbb{M}(s,p)$ is immediately set to `Low`, so that the senders of contributor flows decrease their injection rates rapidly. A state variable $\mathbb{S}(s,p)$ is used in the DCMS to track the `Marking_Rate` value set for the port in order to avoid sending unnecessary messages from the DCMS to a switch. An interval count $\mathbb{I}(s,p)$ tracks the number of inter-sweep intervals since the `Marking_Rate` of port (s,p) was set to `Low`. Details are provided in the pseudo-code of Algo. 2.

The purpose of Algo. 3 is to monitor counter values during ongoing CRR congestion events, and to restore the `Marking_Rate` of a congested port. The challenge lies in determining when to increase the `Marking_Rate` of a congested port back to its `Default` value. The key point is that a victim port may or may not have victim flows. A contributor flow, i.e., a flow that traverses the congested port, could make a neighboring switch port through which it passes a victim port because of link-by-link flow control as explained in Section 2. Therefore, a victim port does not necessarily need to have a victim flow. If there is no victim flow, the `Marking_Rate` of the congested port should be rapidly restored to its `Default` value. On the other hand, if there is a victim flow through a victim port, the `Marking_Rate` of the congested port should be held `Low`.

Since the DCMS does not have per-flow information, it makes conjectures about the presence or absence of victim flows through victim ports. Our solution is based on an observation that when the `Marking_Rate` of a congested port is set to `Low`, the contributor flows will suffer from a rapid rate reduction, which in turn will cause the neighboring victim port to return to its full-capacity state, allowing bulk-data victim flows to enjoy a rapid increase in throughput. Therefore, the DCMS observes the changes in the `PortXmitData` counter of victim ports for an ongoing CRR congestion event. An increase in the observed utilization of a victim port is assumed to indicate the presence of a victim flow, because if only contributor flows passed

through the victim port, dropping the `Marking_Rate` of the congested port to `Low` would cause a rate reduction in the contributor flows, and correspondingly in the utilization of the victim port. On the other hand, a decrease in the observed utilization of a victim port is interpreted by the DCMS as an absence or completion of victim flows passing through the victim port.

The DCMS will reset the `Marking_Rate` of congested ports back to the high `Default` value if there are no more victim ports or if the duration for which a port's `Marking_Rate` was held low exceeds a threshold T_I, which is the Low-MR duration threshold (see Table 1). Details are provided in the pseudo-code of Algo. 3.

In summary, the default setting of switch `Marking_Rate` is chosen to be a high value so that if a congestion event is localized, the contributor flows are allowed to enjoy high throughput so that they end quickly. If, on the other-hand, the congestion causes CRRs in neighboring switch port rates, then the DCMS reduces the `Marking_Rate` of the congested (root) port to reduce the impact of contributor flows on victim flows. The DCMS then plays a guessing game of whether-or-not victim flows are passing through victim ports, when victim flows end, when ports in neighboring switches stop being victim ports, and when a congestion event ends. The DCMS needs to trade off the negative impact of a congestion event on victim flows, while simultaneously ensuring that contributor flows are provided with the opportunity to send data at high rates and thus end quickly.

A limitation of this solution is sweep overhead, as was noted in the dFtree solution [5]. To reduce sweep overhead, the DCMS could rely on historical data to limit its reading of switch counters to just those ports that typically suffer from congestion (e.g., ports connected to disk subsystems). Further the DCMS could be configured to use two inter-sweep intervals, a longer interval until a congestion event is detected, and a shorter interval while a congestion is in progress. A short interval is required for an effective assessment by the DCMS about whether-or-not victim flows exist after a `Marking_Rate` reduction. The initial longer interval would most likely cause the DCMS to miss short-duration congestion events. But we reason that if a congestion event is of short duration, its impact on other flows is necessarily limited, which could justify no DCMS action. On the other hand, long-lasting contributor flows should be throttled as they have more time in which to adversely affect other flows, and therein lies the value of DCMS.

4. Experiments

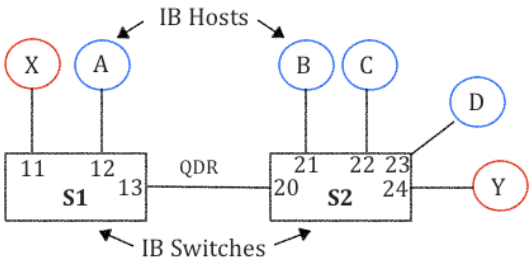


Figure 2. Experimental testbed; All HCA links are SDR

Two experiments were conducted to study CRR congestion events. Experiment I was used to study the impact of the default setting of `Marking_Rate`. In Experiment II, our DCMS prototype was executed to dynamically modify the `Marking_Rate`, and results were collected to study the impact of low-MR duration threshold in the DCMS solution.

Section 4.1 describes the testbed setup used for the experiments. Section 4.2 describes the execution and results of Experiment I. Section 4.3 describes the execution and evaluation of Experiment II.

4.1. Experimental configuration

The experimental setup consists of 6 hosts interconnected by two InfiniBand switches as shown in Fig. 2. Each host has an Intel Xeon E5/Core i7 CPU, 64 GB RAM, PCIe-2 bus, and a Mellanox MT4099 HCA. All hosts run OFA OFED-3.5-2 on top on Scientific Linux v6. In all experiments, OpenSM v3.3.18 runs on one of the hosts.

Each node's HCA was configured to operate at 4xSDR rate (approx. 8 Gbps). The inter-switch link rate was set to 4xQDR rate (approx. 32 Gbps). For simplicity, we refer to 4xSDR as SDR and 4xQDR as QDR in the rest of the paper.

A custom software program called `blast` was used to create high-throughput, memory-to-memory transfers. Each flow sent 140 messages, each of size 128 MiB.

4.2. Experiment I: Default setting

The goal of this experiment was to determine a default value for switch `Marking_Rate`. Since reassembling packets into flows to identify senders of contributor flows inside the network is a compute-intensive operation, the DCMS does not know the senders of contributor flows and hence cannot modify the HCA CC parameters. Nevertheless, the values of these HCA CC parameters will influence congestion-recovery time. Therefore we study the impact of `CCTI_Timer`.

Our main finding is that for the testbed used, if `CCTI_Timer` is set to values in the 75-300 range at the HCAs, then a default value of 64 or 128 for switch `Marking_Rate` is sufficient to allow contributor flows to enjoy high throughput if the congestion event is localized.

The `Marking_Rate` was varied from 0 to 2048, and the `CCTI_Timer` was set to 75, 150, and 300. The other CC parameters were set as follows (i) At each HCA: `CCTI_Increase`: 1, `CCTI_Limit`: 0, and (ii) At each switch: `Threshold`: 15.

`Blast` flows were started sequentially as follows: (i) X-Y flow, (ii) B-D flow at 8s, (iii) C-D flow at 13s, and (iv) A-D flow at 28s (See Fig. 2). Ideally, the X-Y flow should not be affected by the other flows, and the remaining flows should each receive a one-third share of the SDR rate of port (S2, 23). The (s, p) notation is used to identify port p of switch s, and the reader is referred to Fig. 2 for switch and port numbers. The X-Y flow did enjoy throughput close to SDR (which was the rate of the HCAs) under some values of `Marking_Rate` but not others, as seen in our discussion of the results below.

Fig. 3 shows graphs corresponding to three settings of the `CCTI_Timer` and `Marking_Rate`: (i) 75 and 0, respectively, (ii) 300 and 0, respectively, and (iii) 75 and 2048, respectively. We use the shorthand notation 75-0, 300-0, and 75-2048 for these three cases. In the first two cases, where `Marking_Rate` was 0, the X-Y flow enjoyed unhindered SDR throughput; however, in the third case, the X-Y flow was limited to one-third SDR. This finding is explained below.

Fig. 3a shows that the sum total of the throughput values for the B-D, C-D and A-D flows was only 3.1 Gbps from time 40.32 s to 301.8 s in the 75-0 case. This number is even lower at 1.98 Gbps for the 300-0 case as seen in Fig. 3b. At the higher value (300) of the `CCTI_Timer`, the CCTI of flows will be decreased at a lower rate, and therefore inter-packet injection times stay high leading to a lower sending rate. This behavior explains why the total throughput for

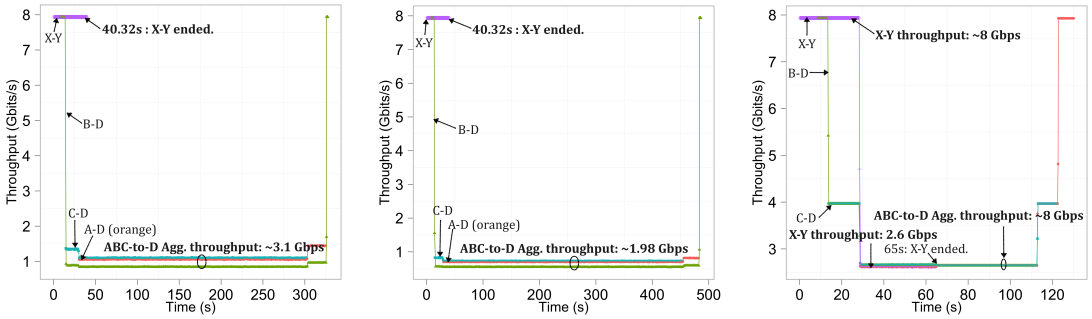
the three flows is lower in the 300-0 case. In these two cases, 75-0 and 300-0, the X-to-Y flow ran unhindered at 7.9 Gbps (close to SDR), and completed its transfer in 40.32 s. Since the aggregate throughput of the B-D, C-D and A-D flows was less than the 8 Gbps rate of port (S2, 23), the congestion dissipated quickly. Since the inter-switch link was QDR, the sum total throughput of the X-Y and A-D flows was less than the inter-switch link rate. No congestion was recorded in the `PortXmitCongTime` counter of port (S1, 15). This is because the sum total of the input-flow rates destined for port (S1, 15) was less than the QDR rate of this port.

In the third case, 75-2048, only 1 in 2048 packets were marked by switch S2 in the contributor flows (A-D, B-D, C-D). This results in a low BECN arrival rate at the senders, A, B and C, causing these senders to maintain fairly high sending rates. Therefore, the sum total throughput of the B-D, C-D, and A-D flows was close to the SDR theoretical maximum rate of 8 Gbps, as seen in Fig. 3c. On the other hand, the X-Y flow was impacted when the A-D flow was initiated at 28 s. The X-Y flow throughput dropped from 7.9 Gbps to 2.6 Gbps at time 29.5 s, as seen in Fig. 3c, and stayed at this rate until the X-Y flow ended at 65 s. In this case, the X-Y flow is a victim flow.

An explanation of why the X-Y flow received only 2.6 Gbps (one-third SDR) lies in the rate-reduction of port (S1, 15) caused due to a lack of flow-control credits from port (S2, 20) (see Fig. 2). Unlike in the 75-0 and 300-0 cases, when the `PortXmitWait` counter at port (S1, 15) was 0, in the 75-2048 case, the `PortXmitWait` counter recorded 2.5×10^9 ticks at the end of the flows. As a tick was 22ns in the testbed network, and the duration of the test was 130 s, the transmitter was not allowed to send data for 55 s out of the 130 s duration, which means that the effective link rate was lowered to 13.54 Gbps from the original 32 Gbps QDR link capacity. Hence this rate reduction at port (S1, 15) would have caused the input-side buffers at ports 11 and 12 to fill up, causing these ports to send FCPs with limited credits to the HCAs at X and A, respectively. The `PortXmitWait` at HCAs X and A also built up to 1.9×10^8 ticks and 4.8×10^8 ticks, respectively. The packet scheduler at port (S1, 15) would have served packets of the X-Y and A-D flows from ports 11 and 12, respectively, in round-robin mode, and therefore both these flows get the same 2.6 Gbps throughput. To determine a suitable default setting, we repeated the experiment for other settings of `Marking_Rate` besides 0 and 2048. Fig. 4 shows the results for `Marking_Rate` values in powers of 2, i.e., 8, 16, $\cdots$, 128. As seen earlier, when the marking rate was 0, the X-Y flow duration was 40.32 sec under all three settings of the `CCTI_Timer` (75, 150, and 300), which is the same duration as that of an unhindered X-Y flow. When `Marking_Rate` was 0, the `PortXmitCongTime` of (S2, 23) reached only 330K, 216K, and 130K, for the three values of `CCTI_Timer`, 75, 150, and 300, respectively. However, for other values of `Marking_Rate`, starting from 8 to 128, the `PortXmitCongTime` of (S2, 23) reached approximately 850K. In other words, when `Marking_Rate` is set to 0, the congestion dissipates faster. The completion time of the X-Y flow was close to 65 for higher `Marking_Rate` values, reaching this level sooner for the smaller 75 setting of the `CCTI_Timer` as seen in Fig. 4a.

Fig. 4b shows that when the marking rate was 0, the total throughput of the A-D, B-D, C-D flows added up to only 3.18, 2.46, and 2.1 Gbps with `CCTI_Timer` values of 75, 150 and 300, respectively, all of which are well below the SDR rate. This illustrates that a marking rate of 0 is needed to avoid the effects of congestion on victim flows, but that this advantage is achieved by sacrificing throughput of contributor flows. With a `Marking_Rate` of 64, the aggregate throughput of the three contributor flows was 8 Gbps for all three values of `CCTI_Timer` at a cost to the victim flow.

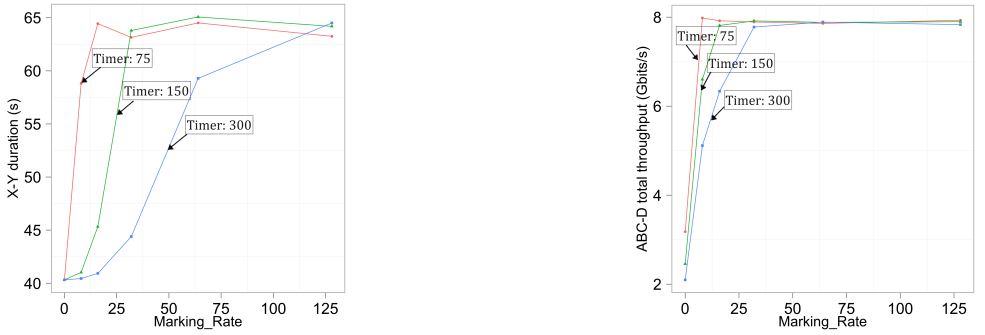
In summary, this experiment shows us that a default value of 64 or 128 can be used for switch `Marking_Rate` if HCAs `CCTI_Timer` values lie in the range 75-300. Further, it showed that the `Low` value of the `Marking_Rate` used by DCMS should be 0.



(a) 75-0 (b) 300-0 (c) 75-2048

Figure 3. Per-flow throughput as a function of time; `CCTI_Timer-Marking_Rate` is shown below each graph

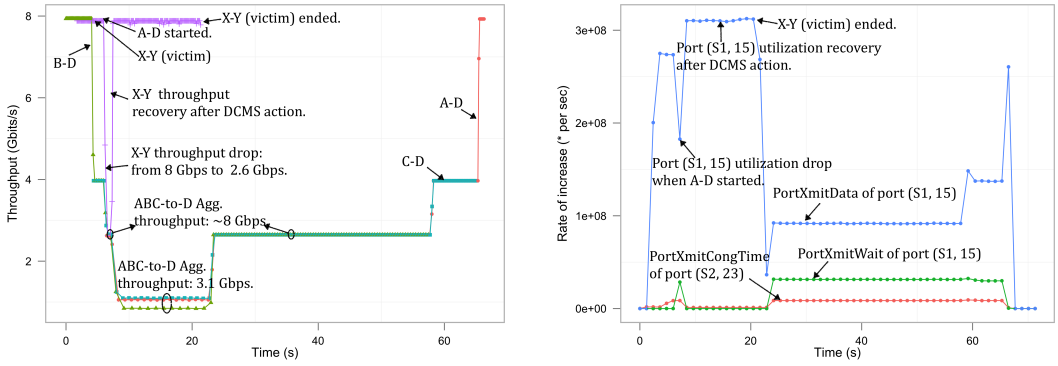
4.3. Experiment II: DCMS



(a) X-Y victim flow duration (b) Aggregate throughput of contributor flows

Figure 4. Illustrates tradeoff between victim-flow and contributor-flow performance

In this experiment, our DCMS prototype was executed to dynamically modify the `Marking_Rate`, and the results were collected to study the impact of low-MR duration threshold in the DCMS solution. Flows were started sequentially in the following order: B-D, X-Y at 2s,



(a) Flow throughput vs. time (b) Port counter values

Figure 5. Scenario 1: Longer Low-MR duration threshold (12 sec)

C-D at 5s, and A-D at 7s. The `CCTI_Timer` was set to 75, and a default value of 128 was used for the `Marking_Rate`. The thresholds T_W and T_C were set to 27.4M and 8M, respectively. Two scenarios were created by varying the Low-MR duration threshold. In the first scenario, the victim flow ended before the threshold was reached, while in the second scenario, the contributor and victim flows continued past the threshold, and therefore, the second reduction in the `Marking_Rate` followed by congestion-event monitoring was required.

Fig. 5a shows the results for Scenario 1. The B-D flow enjoyed 8 Gbps when it started, as did the X-Y flow when it started. At 4.5s, when the C-D flow was started, the B-D and C-D flows each got roughly 4 Gbps, while the X-Y flow got 8 Gbps. As the aggregate rate of the B-D and C-D flows exceeded (S2, 23) port capacity, the rate of increase of `PortXmitCongTime` of port (S2, 23) increased at 4.8s from small values to 6.0×10^6 ticks/s, as seen in Fig. 5b. However, `PortXmitWait` of port (S1, 15) remained unchanged (i.e., rate of increase, as seen in Fig. 5b, was 0) as the A-D flow had not yet started. Meanwhile, `PortXmitData` of port (S1, 15) registered a growth rate of 2.7×10^8 words/s as seen in Fig. 5b, which means X-Y flow was sending packets at a high rate. Despite the increase in `PortXmitCongTime` of port (S2, 23), the controller took no action as the congestion was localized, i.e., there were no victim ports. In other words, the condition of Line 4 in Algo. 2 was not met.

When the A-D flow was started, the throughput of each flow destined to D was one-third of the port (S2, 23) rate (roughly 2.67 Gbps) as seen in Fig. 5a. In addition, the throughput of the X-Y flow also dropped from 8 Gbps to 2.67 Gbps. Fig. 5b shows that the `PortXmitWait` of port (S1, 15) started increasing at 2.8×10^7 ticks/s. Simultaneously, the growth-rate of `PortXmitData` of port (S1, 15) dropped from 2.7×10^8 words/s to 1.8×10^8 words/s. Together, these port counters illustrate the effect of rate reduction at port (S1, 15) caused by the congestion at port (S2, 23). At 7s, the T_C threshold and the T_W threshold were crossed (Lines 2 and 4 of Algo. 2), which caused the DCMS to lower `Marking_Rate` of port (S2, 23) to 0 (Line 8 of Algo. 2). The crossing of both thresholds was an indication that congestion at port (S2, 23) created a victim port (S1, 15).

At 8.5s, as a result of setting `Marking_Rate` to `Low`, the sending rates of flows B-D, C-D, and A-D decreased causing their throughput to drop from 2.67 Gbps to 1.23 Gbps. This action relieved congestion, and the throughput of the victim X-Y flow rebounded from 2.67 to roughly 8 Gbps as seen in Fig. 5a. Fig. 5b shows that at this point, the growth-rate of `PortXmitData` of port (S1, 15) grew to 3.1×10^8 words/s, while the growth rates of `PortXmitCongTime` at port (S2, 23) and `PortXmitWait` at port (S1, 15) dropped back down. From 8.5s to 23s, the controller did not reset the port (S2, 23) `Marking_Rate` to `Default` because the Low-MR duration threshold (12 s) was not crossed. During this time interval, the condition of Line 5 of Algo. 3 was not met, and therefore the `Marking_Rate` of port (S2, 23) was held at 0. This allowed the X-Y flow to complete at 22 s as seen in Fig. 5a.

Soon after, when the Low-MR duration threshold was crossed, the DCMS reconfigured the `Marking_Rate` of port (S2, 23) to 128 (its default value), which allowed the contributor flows to recover. As a result, each of A-D, B-D, C-D flows recovered throughput from 1.23 Gbps to 2.67 Gbps (aggregate of 8 Gbps shown in Fig. 5a). The growth rate of `PortXmitData` of port (S1, 15) remained at 0.91×10^8 words/s until B-D flow ended, at which point the rate of the A-D flow increased, causing the growth rate of `PortXmitData` of port (S1, 15) to increase to 1.37×10^8 words/s. When C-D ended and the A-D flow started enjoying 8 Gbps as seen in Fig. 5a, the growth rate of `PortXmitData` of port (S1, 15) increased to 2.6×10^8 words/s as seen in Fig. 5b.

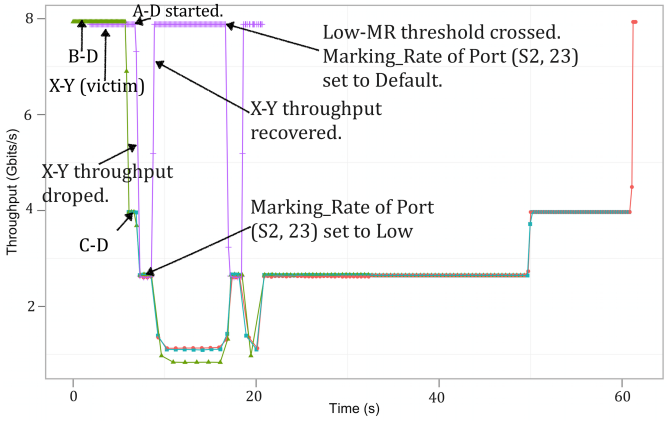


Figure 6. Scenario 2: Shorter Low-MR duration threshold (6 sec)

Finally, when A-D ended and there were no more flows passing through port (S1, 15), the growth rate of PortXmitData dropped back to 0.

Fig. 6 shows Scenario 2 results in which the Low-MR duration threshold was set to 6 sec, which caused the DCMS to reset the `Marking_Rate` of port (S2, 23) to its default value at 17 sec. This operation caused the X-Y victim flow to suffer another drop in its throughput. The DCMS observed a similar build-up of the port counters resulting in a second reduction in the `Marking_Rate` of port (S2, 23), which allowed the X-Y victim flow to recover its throughput. This example illustrates that the DCMS can protect victim flows from the effects of congestion even with no knowledge of flows.

In summary, the above experiments demonstrated the feasibility of deploying a DCMS to manage CC parameters in a dynamic manner such that both victim flows and contributor flows can be served effectively.

5. Related Work

Prior work on InfiniBand congestion control includes simulation and experimental studies [1, 3, 6, 7], recommendations for setting CC parameters [4, 8], and new methods to combat the effects of congestion [9–14].

Our work builds on the above-cited literature by extending our understanding of the effects of link-by-link flow control on congestion in InfiniBand networks. While terms such as congestion spreading [8], and forests of congestion trees [1], capture the effects of link-by-link flow control, in Section 2, we offered a new term cascading rate-reductions to describe the idea that as such links that are behind a congestion point do not themselves suffer from congestion, but rather a reduction in rate as explained in Section 4.2.

With regards to setting CC parameters, our contribution is to determine default values for switch `Marking_Rate`, a parameter that was not considered in the work by Pfister, et al. [8]. The work by Gran, et al. [6] considered switch `Marking_Rate` and HCA `CCTI_Timer` as we did, but stated that the question of how to set CC parameters was a "subject of ongoing research." Our contribution to this subject, which is a dynamic modification of `Marking_Rate`, is a new advance. An adaptive marking rate solution was patented [15], but it requires knowledge about flows.

Of the work on new methods to combat the effects of congestion, Regional Explicit Congestion Notification (RECN) [9, 11] and Destination-Based Buffer Management [12] are effective

but require modifications of the switches as they redirect contributor flow packets to separate queues. Our objective is to improve congestion management in deployed InfiniBand networks and hence a design goal was to require no modifications to InfiniBand switches.

The VOQsw methodology [10], vFtree [13], Flow2SL [14] have the same objective of offering a solution that does not require switch modifications, and leverage InfiniBand's service lane (SL) and virtual lane (VL) features. Our DCMS solution is complementary to these methodologies as it would handle the intra-VL hogging problem.

Our work is the most similar to the dFtree solution proposed by Guay et al. [5] in that the dFtree solution also uses performance counters. However, our congestion recovery uses modifications to switch `Marking_Rate`, while the dFtree solution reassigns hot flows to a slow virtual lane. This paper was written in 2011, and states that since congestion control was newly introduced, it was not available in all switches and HCAs, and therefore, the dFtree solution was designed to work without congestion control.

Conclusions

This work has demonstrated the feasibility of dynamically modifying a switch congestion-control parameter called `Marking_Rate` to enable flows victimized by a congestion event to recover their throughput rapidly. We conclude that even without per-flow information, it is feasible for a Dynamic Congestion Management System (DCMS), which is software running on an external server, to use just the information in switch port counters to make educated guesses about the presence or absence of victim flows during a congestion event. Our experimental work has demonstrated that if the `Marking_Rate` is kept too low, flows that cause congestion would experience severe rate reductions, which would prolong the congestion-event duration and correspondingly increase the probability of creating victim flows. But without a dynamic management system, `Marking_Rate` cannot be kept high because if a congestion event occurs, such a setting could create many victim flows. Thus, the value and feasibility of a dynamic congestion management system has been demonstrated in this work.

Acknowledgments

This work was supported by NSF grants CNS-1116081, OCI-1127340, ACI-1340910, CNS-1405171, ACI-0958998, OCI-1127228, and OCI-1127341. The authors also thank Patrick MacArthur, David Wyman, and Chuck Valenza, University of New Hampshire, for testbed support.

References

1. Gran EG, Reinemo SA, Lysne O, Skeie T, Zahavi E, Shainer G. Exploring the Scope of the InfiniBand Congestion Control Mechanism. In: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International; 2012. p. 1131–1143.

2. InfiniBand Trade Association. InfiniBand Architecture Specification Volume 1, Release 1.3; 2015. Available from: `http://infinibandta.org`.

3. Gran EG, Reinemo SA. InfiniBand Congestion Control: Modelling and Validation. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques.

SIMUTools '11. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering); 2011. p. 390–397.

4. Gran EG, Eimot M, Reinemo SA, Skeie T, Lysne O, Huse LP, et al. First experiences with congestion control in InfiniBand hardware. In: Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on; 2010. p. 1–12.

5. Guay WL, Reinemo SA, Lysne O, Skeie T. dFtree: A Fat-tree Routing Algorithm Using Dynamic Allocation of Virtual Lanes to Alleviate Congestion in InfiniBand Networks. In: Proceedings of the First International Workshop on Network-aware Data Management. NDM '11. New York, NY, USA: ACM; 2011. p. 1–10.

6. Gran EG, Zahavi E, Reinemo SA, Skeie T, Shainer G, Lysne O. On the Relation between Congestion Control, Switch Arbitration and Fairness. In: Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on; 2011. p. 342–351.

7. Santos JR, Turner Y, Janakiraman G. End-to-end congestion control for InfiniBand. In: INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. vol. 2; 2003. p. 1123–1133 vol.2.

8. Pfister G, Gusat M, Denzel W, Craddock D, Ni N, Rooney W, et al. Solving hot spot contention using InfiniBand architecture congestion control. In: Proceedings In High Performance Interconnects for Distributed Computing, Research Triangle Park, NC; 2005. .

9. Duato J, Johnson I, Flich J, Naven F, Garcia P, Nachiondo T. A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In: High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on. IEEE; 2005. p. 108–119.

10. Gomez ME, Flich J, Robles A, Lopez P, Duato J. VOQSW: a methodology to reduce HOL blocking in InfiniBand networks. In: Parallel and Distributed Processing Symposium, 2003. Proceedings. International; 2003. p. 10 pp.–.

11. Garcia PJ, Quiles FJ, Flich J, Duato J, Johnson I, Naven F. Efficient, Scalable Congestion Management for Interconnection Networks. Micro, IEEE. 2006 Sept;26(5):52–66.

12. Nachiondo T, Flich J, Duato J. Buffer Management Strategies to Reduce HoL Blocking. Parallel and Distributed Systems, IEEE Transactions on. 2010 June;21(6):739–753.

13. Guay WL, Bogdanski B, Reinemo SA, Lysne O, Skeie T. vFtree - A Fat-Tree Routing Algorithm Using Virtual Lanes to Alleviate Congestion. In: Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International; 2011. p. 197–208.

14. Escudero-Sahuquillo J, Garcia PJ, Quiles FJ, Reinemo SA, Skeie T, Lysne O, et al. A new proposal to deal with congestion in InfiniBand-based fat-trees. Journal of Parallel and Distributed Computing. 2014;74(1):1802 – 1819.

15. Zahavi E. InfiniBand adaptive congestion control adaptive marking rate. Google Patents; 2010. US Patent App. 12/245,814.