Improving Communication-Phase Completion Times in HPC Clusters Through Congestion Mitigation

Yitzhak Birk
Technion - Israel Institute of Technology
Haifa 32000, Israel
birk@ee.technion.ac.il

Vladimir Zdornov
Technion - Israel Institute of Technology
Haifa 32000, Israel
zdv@tx.technion.ac.il

ABSTRACT

Congestion arises in cluster-based supercomputers due to contention for links, spreads due to oversubscription of communication resources, and reduces performance. We mitigate it using efficient, scalable adaptive routing and explicit rate calculation. We use virtual circuits for in-order packet delivery; path setup is performed by switches locally with no blocking or backtracking. For random permutations in a slightly enriched fat-tree topology, maximum contention is reduced by up to 50% relative to static routing, but only rate control can translate this into actual gain. Unfortunately, TCP's window-based rate control fails because of the low bandwidth-delay product, and small buffers moreover cause congestion spreading even with a single-packet window. InfiniBand's CCA employs multiple parameters, which must apparently be tuned per topology and traffic pattern. Focusing on phase-based applications, we present a distributed explicit rate-assignment algorithm for completion-time minimization of the communication phase (min-max flow completion). Also, a generally applicable packet-injection scheme for a source with different-rate flows that realizes desired rates even with very small switch buffers. Simulations show that adaptive routing alone is ineffective, rate control's effectiveness is limited, yet together they shorten the communication phase by tens of percents. Finally, our explicit rate-calculation algorithm is faster than current reactive schemes.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network management; C.2.2 [**Network Protocols**]: Routing protocols; C.2.1 [**Network Architecture and Design**]: Network topology

Keywords

Congestion, interconnects, InfiniBand, computer clusters, congestion control, adaptive routing.

SYSTOR'09, May 4-6, Haifa, Israel

1. INTRODUCTION

Computer clusters, the architecture of choice for most supercomputers, comprise up to tens of thousands of general purpose computing elements interconnected by a high-speed network [1]. Performance of these parallel machines often greatly depends on the properties of their interconnects.

InfiniBand [2] (24%) and Gigabit Ethernet [3] (58%) are presently the most prominent top-500 supercomputer interconnects. InfiniBand was intended from the outset for high-performance clusters. Ethernet, in contrast, was intended for general purpose LAN and WAN communication, but recently introduced cluster versions of Ethernet are increasingly adopting InfiniBand-like features. We therefore chose InfiniBand as the platform for our work.

1.1 The Congestion Problem

Flows share links, contend for capacity, and their transmission rates are directly affected by the contention and by the rates attempted by the contending flows. Adaptive routing, which can reduce contention, is thus expected to improve overall network performance. Unfortunately, however, even with optimal routing, link sharing may still lead to inefficiencies caused by poor management of shared buffer space. In fact, for any given buffer size, sufficiently long flows will incur the problem.

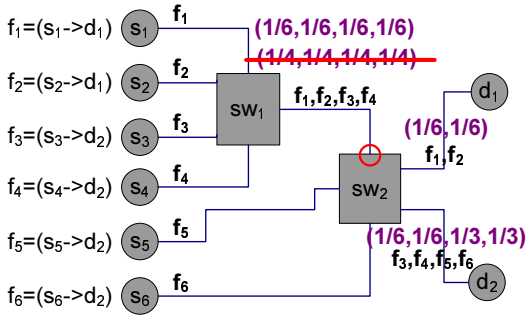

Figure 1: The damage of congestion

Consider, for example, the scenario in Fig. 1. Six flows, $f_1, ..., f_6$, traverse a network with two switches, sw_1 and sw_2. The sources start injecting packets at time $t = 0$. We assume virtual output queuing, and that only hop-by-hop flow control, whose only function is to prevent packet drops, is used. Thus, sources and switches push packets onto links as long as free buffers are available at a link's remote

end. For facility of exposition, *here only*, we assume that output ports serve pending packets using a Round-Robin (RR) policy among input ports. Initially, every flow $f_1, ..., f_4$ is transmitted over link $sw_1 \rightarrow sw_2$ at rate $\frac{1}{4}$. At output port $sw_2 \rightarrow d_2$, f_5, f_6 each get a rate of $\frac{1}{3}$, while f_3, f_4 each get $\frac{1}{6}$. Initially, the packet insertion rate of f_3, f_4 into sw_2 is thus higher than their service rate. Consequently, regardless of buffer sizes, the input buffer at the end of the link $sw_1 \rightarrow sw_2$ becomes full, and the resulting back pressure causes the link to reduce its transmission rate.

A simple simulation reveals that in steady state, $sw_1 \rightarrow sw_2$ transmits only at rate $\frac{2}{3}$. Switch $sw_2 \rightarrow d_2$ is referred to as the *congestion root*, because it is oversubscribed yet transmits at full rate; $sw_1 \rightarrow sw_2$ is a *congestion victim*, since it is not saturated despite the fact that it has more data to push. Basically, congestion roots generate the back pressure, which creates congestion spreading and congestion victims.

The resulting rate vector is $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{3}, \frac{1}{3})$. From the link capacity perspective, this is sub-optimal, because in this situation f_1, f_2 should apparently be able to increase their rates to $\frac{1}{3}$. The assignment is moreover unfair, since f_5's transmission rate is double that of f_4.

Considering finite flows of the same size, we notice that the overall completion time (by which all flows end) suffers as well, despite the fact that once a flow ends others may be able to increase their transmission rates. Suppose that all flows equisized, each requiring 1ms for transmission at line speed. After 3ms, flows f_5 and f_6 end. At this point $sw_1 \rightarrow sw_2$ has transmitted two flows worth of data, and its rate is increased to 1. It will take 2ms to transmit its remaining data, resulting in a 5ms completion time. This is 25% longer than minimum of 4ms (each link is required to pass at most four flows worth of data).

The example demonstrates *congestion spreading*, which is a form of high-order head-of-line blocking. It was first described in [4] as tree saturation. In larger networks, the congestion root can create a tree of full buffers. Every flow crossing one of the buffers in the tree, even if it avoids the root, may be adversely affected.

The fundamental cause underlying congestion spreading is that sources attempt to push more traffic than the network is capable of accommodating. Therefore, an appropriate rate control mechanism that limits the injection rate of the sources is required in order to remedy the problem and ensure fairness. In the example, rate $\frac{1}{4}$ for every flow is optimal.

1.2 Performance Measures and Goals

In this paper, we focus on the case of a an HPC cluster that is used by a single phase-based application that alternates between computation and communication phases, separated by a global barrier. At the beginning of a communication phase, each source knows its destinations and the exact amount of data to be transferred. This setting represents important applications such as wind-tunnel simulation. Our goal is to minimize the duration of the communication phase, determined by the latest flow completion. Flows are assumed to comprise many packets, rendering management overhead reasonable.

Two additional interesting scenarios are discussed in [5]. One, which is the default scenario in the research of networking, is that of independent flows. Here, max-min fair rate allocation is appropriate. Another is that of multiple phase-based application that share a cluster can be viewed as a combination of the foregoing ones: each application cares about the duration of its communication phase, while rates should be allocated fairly among applications.

1.3 Contributions of this Work

In this work, we propose adaptive routing and rate control schemes, and evaluate them both in isolation and together. Since a large number of flows may traverse a switch, storage and manipulation of per-flow state in switches is prohibitive. We therefore restrict our approaches to require a fixed amount of state information per switch.

Our first contribution is a heuristic adaptive routing scheme (Section 4) that relies on local heuristic decisions. Importantly, it guarantees in-order packet delivery by means of a kind of virtual circuit (VC) mechanism, yet avoids the scalability problems of "classical" VC schemes.

Our second contribution is a distributed rate control algorithm for the single-application scenario (Section 5). The algorithm relies on explicit rate calculation, and minimizes communication phase completion time for any fixed choice of routing. (Rates are computed after routes have been established). The algorithm is initially developed and evaluated using a "fluid model", whereby any rate assignment is considered feasible iff it does not violate the capacity constraints of any of the links. This approach ignores the discrete nature of packet networks, queuing issues, and limited buffers.

In order to realize the theoretically feasible rates in practice, we present a packet injection scheme that is an adaptation of *Shaped Virtual Clock* (SVC) [6], and show through simulation that it can do so even with a very moderate size of buffers in switches (Section 6). The injection scheme is designed to suppress bursts of individual flows and the aggregate traffic leaving a source.

Finally, we present simulation results. When adaptive routing or rate control are used individually, minor performance improvement is witnessed. In fact, adaptive routing can even hurt. However, when used together, the results demonstrate a very significant reduction in communication-phase duration. Note also that the rate-calculation algorithms is tailored to the single-application scenario, but the adaptive routing and packet-injection schemes have much broader applicability.

The results were acquired through simulation in OMNeT++ event-driven simulation framework [7]. We created special InfiniBand models for that purpose. Since our goal was to simulate large networks with thousands of nodes, our models operate at the functional, rather than cycle-accurate, level. Although the methods proposed in this work are topology agnostic, we used the k-ary n-tree [8], which is a variant of a practical fat tree [9], as the basis for all our experiments. This topology is popular in modern clusters.

The remainder of this paper is organized as follows. Section 2 states our network model and taxonomy, along with adopted InfiniBand features. Section 3 surveys related work on adaptive routing and rate control. Our proposed adaptive routing is presented in Section 4. Our rate control algorithm is presented in Section 5. Section 6 deals with realization of calculated rates. Finally, Section 7 offers concluding remarks.

2. NETWORK MODEL & TAXONOMY

A network comprises switches, host channel adapters (HCAs), and bi-directional links. We use H and SW to denote the sets of HCAs and switches, respectively. Switches and HCAs are collectively referred to as *network elements*, denoted $E = SW \cup H$. A bi-directional link is a pair of uni-directional links, with *link* referring to the latter. The set of links is denoted by L. (Note the deviation from common graph notation.) InfiniBand allows the use of multiple virtual lanes (VLs) for QoS segregation of traffic. This can be used to grant control packets total priority over data packets. Finally, for clarity of presentation we assume all links in the network to have a fixed capacity $c_l = 1$.

A *connection* is a ("logical sender", "logical receiver") pair. (Any given logical sender or receiver resides in a single physical entity.) Packets sent over the same connection must be delivered in their transmission order. A connection may represent a reliable connection (the InfiniBand equivalent of TCP-like socket association) but is not limited to it. Every source can have multiple concurrent connections, each uniquely identified by a combination of (physical) source address (SLID), destination address (DLID), and a source-unique connection identifier (CID).[1]

Data is transmitted over a connection as a contiguous sequence of *flows*, each comprising a contiguous sequence of packets belonging to said connection. Flow is a management abstraction; it may comprise anything between part of an InfiniBand message and a contiguous sequence of such messages.

Several relevant characteristics of InfiniBand are part of our model: 1) virtual cut-through switching, which greatly reduces the bandwidth-delay product; 2) virtual output queuing in switches, placing the buffers at the inputs of the switch; the size of the buffers is practically very small, designed to hold few MTU-size packets; 3) hop-by-hop flow control is used to prevent the transmission of packets into a full buffer, so no packets are dropped; and 4) switches use oblivious, destination-based routing to forward packets. The combination of oblivious routing with lossless communication has a dramatic impact on the performance and implementation complexity, since together they guarantee in-order packet delivery. As a result, no retransmissions are required unless (rare) physical packet corruption occurs.

From a management perspective, InfiniBand networks have three fundamental traits: 1) they constitute a managed environment, so all network elements (switches, HCAs) can be configured to operate according to some chosen protocol without being concerned about inter-operability or malicious behavior; 2) reliable communication, permitting the formulation of management algorithms under the assumption that every packet eventually reaches its destination while passing through each link exactly once; physical failures may occur, but are rare and may be treated without efficiency concerns; and 3) small network diameter (both in terms of hops and actual distance), high data rates and low-latency switches result in a very low end-to-end packet delay; this reduces the cost of global management operations such as broadcasts and barriers.

As our baseline, referred to as *no control (NC)*, we consider interconnects that operate as follows. Only hop-by-hop flow control is used, whose only function is to prevent packet drops. Thus, sources and switches push packets onto links as long as free buffers are available at a link's remote end. Switches have buffers at input ports, and every output port is allowed to send at most one packet over its link in every time step. An output port serves pending packets from different inputs using a First-Come-First-Served (FCFS) policy, and there is no restriction on the number of packets that can leave an input port's buffer for transmission over different output links in a single time step. (This is known as infinite speedup.) Under this assumption, packets at a given input port's buffer that are destined to different outputs do not suffer from head-of-line blocking, yet they do share the buffer pool.

[1]In InfiniBand *queue-pair number* (QPN) can be used when possible.

3. RELATED WORK

3.1 Adaptive Routing

Standard InfiniBand networks employ oblivious routing, resulting in simple packet routing while guaranteeing in-order delivery. However, such routing cannot react dynamically to the applied traffic pattern. The alternative is adaptive routing. Finding an optimal routing under various optimality criteria generally leads to one of the variants of the Multi-commodity Flow problem, an NP-complete problem [10], so heuristic approaches are used in practice.

Lossless networks are prone to deadlocks that arise whenever a group of packets cannot advance due to a cyclic dependency among them. Therefore, deadlock avoidance is a crucial property of any routing scheme. In [11], a necessary and sufficient condition for deadlock-free routing is formulated. It is used as a basis for several topology agnostic routing schemes [12, 13].

In some cases, deadlock-freedom is provided by the topology of a network. This is particularly true for practical fat trees (Section 4.2), in which no cyclic dependency can arise if minimal paths are used.[2] For fat trees, static routing schemes were proposed in [14, 15], and an adaptive one was examined in [16, 15]. The latter proposed to use packet-level adaptation, routing different packets of the same flow independently. This approach, taken also in [17], is fairly simple to implement, but violates the in-order delivery requirement.

An alternative approach that preserves in-order delivery was proposed in [18, 19, 20]. InfiniBand allows the assignment of multiple addresses (LIDs) to an HCA, so the same destination can be addressed using multiple LIDs. Since the routing for each LID is fixed independently, multiple paths can be defined for each source-destination pair at network configuration. The source selects a specific path by choosing a LID from among those assigned to the destination. This approach preserves packet order, provided that the entire flow is routed using the same LID. Unfortunately, multiple-LID routing does not provide full routing flexibility, because the alternative paths are set once. Moreover, a limited (64K) number of LIDs introduces a tradeoff between cluster size and its routing capabilities.

Flexibility and in-order delivery can be achieved using a virtual circuit (VC) mechanism [21, 22]. The resources, which can include link capacity, buffers, routing entries etc., are reserved during VC *setup*, before the first packet of the

[2]In a practical fat tree, unlike an ideal one, multiple routes exist for the same source-destination pair.

connection is sent, and all data packets follow the same path. When a connection ends, its resources are released in a *tear-down* procedure.

Unfortunately, the requirement to store per-VC information in each switch along a VC's path, combined with the resources in switch routing tables, limits the number of connections that can traverse a switch. VC setup may thus fail, resulting in a *blocked* connection whose setup must be attempted at a later time. In some schemes, alternative paths are tried, but unfortunately those may also be blocked. So, the penalty of blocking may be very high. In Section 4, we propose a scheme that uses virtual circuits to route individual flows (rather than connections). Our circuits reserve only routing resources, and efficiently avoid blocking.

3.2 Rate Control

The goal of rate control is to avoid link oversubscription and the resulting clogging of buffers, while ensuring fair and efficient utilization. We classify existing rate control schemes into four main groups: *reactive window-based*, *reactive rate-based*, *precise explicit rate calculation* and *approximate explicit rate calculation*. To our knowledge, all existing solutions were designed for the independent flows scenario.

In reactive schemes, a source increases the load of a flow optimistically, until it receives some kind of indication that the flow passes through a congested link. It reacts by reducing the applied load until the flow is believed to no longer traverse congested links. Reactive schemes usually never reach a truly steady state; instead, they oscillate about some (not necessarily optimal) set of values.

Reactive scheme can be classified as follows: 1) Whether the load that a flow may apply to the network is determined by the size of a congestion window or using an explicit rate value. 2) Whether the congestion indication is implicit or explicit. Implicit indication is derived from long round-trips or (in lossy networks) from sudden packet loss. In the explicit case, switches play an active role in informing the source, usually by marking data packets traversing the congested link. 3) By the increment/decrement policy, whereby the change in applied load optionally depends on the acquired feedback. With Additive-Increase, Multiplicative-Decrease (AIMD) [23], for example, the new load is only a function of its current state.

The most prominent congestion control mechanism, that of TCP, is a reactive, window-based scheme, and has spawned many variations over the years to suit different network environments [24, 25, 26, 27, 28]. The congestion window controls the number of unacknowledged packets that may be injected into the network.

A typical TCP network is characterized by relatively long round-trip delay (high bandwidth-delay product) and switching elements with large buffers, which makes the congestion window a convenient tool. Cluster networks are different. Virtual cut-through switching makes the bandwidth-delay product very low even for networks with large radii. In fact, in most practical scenarios, the maximum window size should be very few MTU packets per flow, leaving little room for control. Moreover, with very small buffers, congestion spreading can occur even if the window size is fixed to one packet [29]. For these reasons, InfiniBand *Congestion Control Architecture* (CCA) uses *Inter-Packet Delay* (IPD), rather than window size, to set the desired load. Thus, CCA is a reactive, rate-based control scheme.

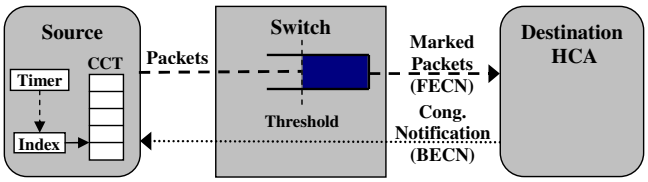

Figure 2: Congestion Control Architecture

The components of CCA are presented in Fig. 2. Switches monitor the number of packets awaiting transmission at each output port. If a port transmits at line speed, yet the number of awaiting packets exceeds a predefined threshold, it is said to be a *congestion root*. If the threshold is crossed, but the port does not operate at full speed (due to the back pressure), it is a *congestion victim*.

When a switch detects one of its ports to be a congestion root, it starts marking packets that use the port with a *Forward Explicit Congestion Notification* (FECN) bit. The marking rate is a parameter. Upon arrival of a marked packet at its destination, a special *Backward Explicit Congestion Notification* (BECN) packet is sent back to the source. Sources maintain a table of increasing IPDs, known as the *Congestion Control Table* (CCT), and an *index* variable. When a source receives a BECN, the index is incremented and the rate is effectively reduced. A *timer* is used to decrease the index (increase rate) over time while no BECNs are received.

The multitude of parameters (threshold, marking rate, CCT values, decrease time) is a serious impediment to correct configuration of the scheme. Our attempts to choose CCA parameters lead to a grim conclusion whereby a setting (set of control parameters, not a specific transmission rate) that is optimized for one traffic pattern can have catastrophic results for another.

Consider for example the "all-to-one" traffic pattern. Here, the link connected to the destination HCA carries a very large number of flows and is a congestion root. Consequently, the switch should best mark every crossing packet in order to provide timely feedback to all flow sources. However, doing so for permutation traffic that exhibits much lower contention would cause sources to over-react, leading to rate oscillation without convergence. Finally, using the low marking rate (that suits the permutation traffic) for all-to-one traffic would not prevent buffer clogging, resulting in congestion spreading and reduced throughput.

Indeed, various attempts to set CCA parameters were not successful. Analytic derivations such as [29] and [14] were never tested in networks of a realistic size with non-trivial traffic, and the systematic simulation method of [30] suffers from partial sample space coverage, typical for this approach. The apparent conclusion is that in order for the interconnect to operate efficiently, CCA requires careful tuning for each particular topology and traffic pattern. This is often impractical.

Unlike their reactive counterparts, explicit rate calculation schemes actively compute the rates to be assigned to flows. The rate assignment is chosen to utilize links efficiently and fairly while preserving feasibility. Explicit rate calculation makes it easier to define clear and flexible design goals that provide firm performance guarantees, and the calculation is usually performed in a distributed man-

ner. *Precise* algorithms [31, 32, 33] do not depend on specific network characteristics, and are guaranteed to converge under very weak assumptions. *Approximate* algorithms [34, 35, 36, 37, 38] are expected to operate faster, but rely on parameter tuning and may exhibit oscillatory behavior.

More details on various rate control techniques can be found in [39, 40]. We note that most existing schemes do not include flow weights as a means of prioritization, whereas we do (Section 5). Even when no weighting is required, current schemes are either approximate or do not function properly (oscillations and/or congestion spreading) because there is no fixed setting of their control parameters that works properly for all traffic patterns and topologies or even for all traffic patterns with any given topology.

4. ADAPTIVE FLOW ROUTING

In this section, we propose a generic, scalable routing scheme based on flow-level VC-like routing. A critical constraint is to require a fixed amount of state information per switch, thereby avoiding the scalability limitations of "classical" VCs that require a switch to store information for every flow that traverses it.

4.1 Generic Scheme

A connection is a sequence of flows, each of which is a sequence of packets. In-order delivery is required among all packets of a connection. Each packet carries a source address (SLID), destination address (DLID), and a source-unique connection identifier (CID), which jointly constitute a *globally unique connection identifier* (GCID). Our approach is to adapt the routing at flow boundaries; all of a flow's packets follow the same path. Transmission of a flow's packets does not commence until the source ensures that all packets of the previous flow of the same connection have arrived.

Data Structures

In large clusters using high-radix switches, numerous flows may traverse any given switch may be very large. With "classical" VCs, this gives rise to a memory scalability issue (per-flow routing information must be stored) and to a possible performance problem (associative lookup in a large table). We solve these problems by intelligent use and extension of the standard InfiniBand routing table (InfiniBand RT). This table has a constant size equal to the maximum allowed number of destinations (defined by the standard), and is addressed (as a regular table, not associatively) by DLID. Therefore, InfiniBand RT is accessed very quickly. Each RT entry holds a port number.

We equip each switch with a single *extended routing table* (ERT), which is an extension of the regular InfiniBand RT. ERT entries have two fields. The first, as before, holds a port number, which we refer to as the *default port* (DP) for the DLID, and the second field contains a list of several *alternate ports* (APs).

We also use VC routing tables, dubbed *route cache* (RC). An RC entry stores a GCID and port numbers, and is accessed associatively by GCID. For scalability while preventing connection blocking (unlike in "classical" VC), some flows traversing a switch do not have an RC entry. Packets belonging to such flows are routed by the switch to the DP for their DLID. RC size is thus irrelevant for correctness, though it may affect routing quality.

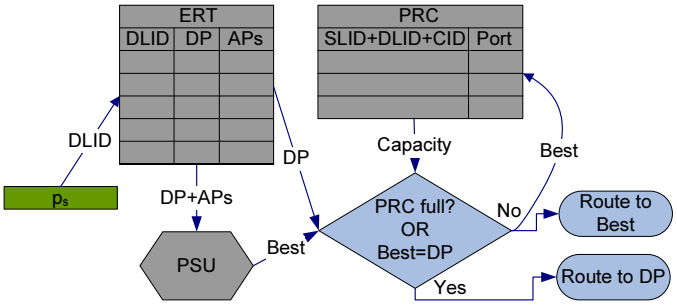

Figure 3: Route setup

Instead of having a centralized RC, we keep a *port routing cache* (PRC) for each input port. Each PRC is still addressed by GCID, but only holds information pertaining to flows traversing one input port, and different PRCs can be accessed in parallel. A PRC can be implemented as a *Content Addressable Memory* (CAM) or as a hash table.

Route Management

When a flow f starts, a control packet p_s is sent to set up the route. Each switch on p_s's path routes the packet, basing its heuristic decision on local information. (See Fig. 3.) Upon arrival of p_s to an input port, both the ERT and the PRC are accessed. The ERT is accessed by DLID, and passes a list of all the matching output ports to a *port selector unit* (PSU). The PSU retrieves the heuristic measure for each port and selects the best available port, based on the heuristic and other optional information. If the PRC is full or the DP is found to be the best output port, then p_s is routed to the DP and no entry is added to the PRC. Otherwise, p_s is routed through the chosen alternate port, and its GCID along with the selected port become a new PRC entry. Note that after being routed to a DP at some switch, p_s can still be adaptively routed by later switches on its path.

Since p_s advances through switches regardless of the state of RC, it should eventually reach its destination. Once this happens, the packet is sent back to the source along a *default path* (defined by default ports). On its way back, p_s does not affect the state of the switches.

Upon p_s's return, the route setup procedure is complete, and the source starts sending data packets. These are routed by switches according to PRC entries that match their GCID, defaulting to their DLID's DP.

When a flow ends, a control packet p_t is sent. It is routed like a regular data packet, and erases the PRC entries matching its GCID along its path, thus tearing it down and freeing up PRC entries. Once p_t returns to the source, a new flow with the same GCID may begin.

The above scheme is routing-selection policy agnostic, but requires that cycle-freedom be guaranteed. (This is also required for deadlock freedom.)

4.2 Application to a k-ary n-tree

A k-ary n-tree [8] (Fig. 4a) is a practical implementation of a fat tree, whereby every internal node of the fat tree is implemented as a set of interconnected switches with unit-capacity links. The switches implementing a single fat-tree node are jointly dubbed a *logical node*. As depicted in Fig. 4a, a minimal route in a k-ary n-tree comprises an as-

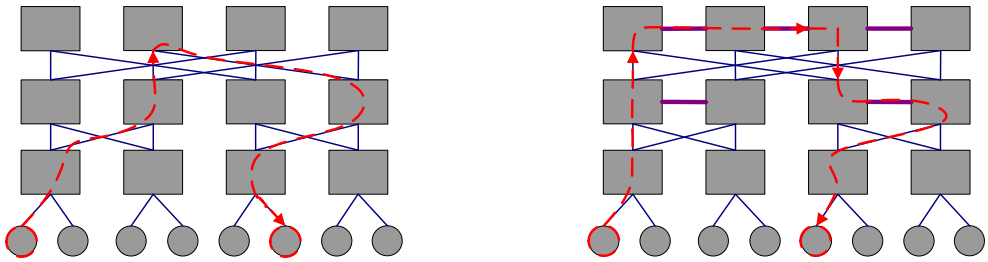

Figure 4: Regular and modified k-ary n-trees

cent to a switch that is a closest common ancestor of the source and the destination, followed by a descent to the destination. Importantly, the ascent path can be chosen arbitrarily. However, once the ascent is completed, the descent path is unique. Indeed, simulation results show that the routing flexibility of such a topology is lacking.

Modified k-ary n-tree

In order to increase routing flexibility, we use a *modified k-ary n-tree*: switches belonging to the same logical node in the ideal tree are connected by horizontal links, as depicted in Fig. 4b. Due to the aforementioned properties of the k-ary n-tree, horizontal movement between such switches preserves reachability of a destination through descent.

Routing in the enriched topology (Fig. 4b) still comprises the ascent and descent phases. The ascent is performed adaptively, exactly as in a regular k-ary n-tree. During its descent, a flow is allowed to take multiple horizontal hops before proceeding to a lower level. The horizontal direction (right/left) is chosen once per level, before the first horizontal hop. It is chosen so as to maximize the permissible number of horizontal hops before reaching the boundary of a logical node. Thus, flows initially arriving to a switch in the left half of a logical node are sent to the right, and vice versa.

It is obvious that the proposed routing in the modified k-ary n-tree avoids cyclical dependency between links, and is therefore deadlock-free. Also, the resulting flow paths do not contain cycles. Correct operation of our generic adaptive routing scheme is thus ensured.

We tested the modified topology and the proposed routing for 1000 random permutations in a 16-ary 3-tree (4096 end-nodes). Here and later, we used the number of flows traversing a link as a heuristic measure of dynamic link quality, and the size of the RC table was assumed to be unlimited.[3] The number of parallel links with unit capacity, referred to as *horizontal width*, was used as a parameter. In every run, we measured the contention experienced by individual flows (maximum over links traversed by a flow). The maximum and the average (over flows) results are summarized in Fig. 5, which presents these measures for both oblivious and adaptive routing.

We draw two important conclusions. First, when horizontal width is set to zero (adaptation only during ascent), adaptive routing has no impact on performance. This is attributed to the fact that good greedily chosen ascent cannot guarantee successful descent. Second, with horizontal width of two or more, the maximum contention is reduced by approximately 50%, while the average contention is reduced by more than 20%. We stick to a horizontal width of two in the rest of our experiments, as it increases the total number of ports by a mere 10%.

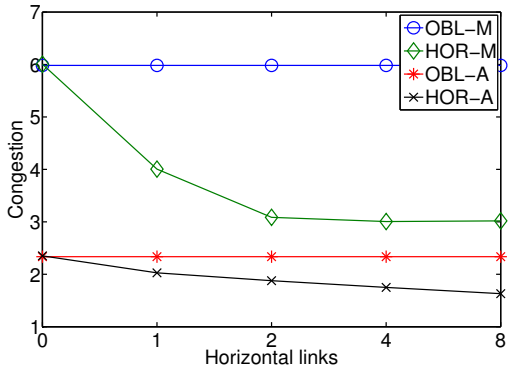

Figure 5: Number of horizontal links versus maximum congestion

Remark: Here and later, we compare the combination of the additional "horizontal" links and adaptive routing (jointly referred to as "adaptive routing") with static routing in an unmodified tree. Also, we did not compensate for the additional links by commensurately increasing the data rates of the links in the unmodified tree, as such an enhancement cannot be implemented in practice. (Apparently, for an unknown traffic pattern, no static routing can make efficient use of the horizontal links anyhow.) For further implementation details of our routing scheme and additional experimental results, refer to [5].

5. RATE CALCULATION

In this section, we present a very simple distributed algorithm, which is designed to operate in an asynchronous, reliable environment and minimizes completion time. Stated differently, it provides a max-min normalized (by flow length) rate assignment to the flows. Importantly, it only requires network elements to store a small, fixed amount of state in-

[3]The traffic patterns used in our simulations did not result in large numbers of flows traversing switches. Nevertheless, the dependence of performance on the RC table size has yet to be examined.

formation, independent of number of flows or network topology. Control-packet size is also fixed.

5.1 The Optimal Completion Time

Let us begin by presenting a precise mathematical model. We assume the routing to be fixed for the duration of the calculation. Denote by F_l the set of flows traversing link l, and by L_f the set of links on the path of flow f. Let every $f \in F$ have an associated weight w_f, equal to its size d_f. Later we will express w_f in units of time, i.e., the time it takes to transmit f at line speed.[4] Let W_l be the aggregate weight of flows traversing $l \in L$. Denote by W_f the maximum W_l among links in L_f. Finally, let W be the maximum link weight in the network. Then,

$$W_l = \sum_{f \in F_l} w_f,\ W_f = \max_{l \in L_f} \{W_l\}, \tag{1}$$

$$W = \max_{l \in L} \{W_l\} = \max_{f \in F} \{W_f\}. \tag{2}$$

Considering sufficiently long flows, we ignore small disparities among the starting times of different flows, and initially assume all flows belonging to the communication phase to start at $t = 0$ (the beginning of the phase). Particularly, we forbid flows to start in the middle of the phase. The control of the network is reduced to assigning each flow $f \in F$, at any given time τ, an instantaneous rate $r(f, \tau)$, without violating the capacity of the links. The amount of flow f data transmitted by time t is

$$D(f, t) = \int_0^t r(f, \tau) d\tau. \tag{3}$$

The phase completion time is defined to be the smallest time by which the data of all flows has been transmitted.[5] Our goal is to find a feasible rate assignment vector r that is constant in time, i.e. $r(f, \tau) = r(f)$, and guarantees the shortest phase completion time. (We will prove that the optimum can indeed be achieved by constant rate assignment.)

First, consider a scenario in which all flows have the same size and weight $w_f = 1$. Giving $r(f) = \frac{1}{W}$ to all flows minimizes completion time. It is furthermore the most resource conserving, as it assigns to each flow the minimum rate required for minimizing completion time. Unfortunately, however, it requires some kind of global coordination mechanism, as every flow must know the minimum of the assigned rates.

We next make two important observations: 1) giving flows higher rates than the above minimum does not necessarily impede the progress of other flows, and 2) so doing does not increase the amount of transmitted data, as a completed flow will cease to consume bandwidth. With this in mind, we next propose a simpler rate assignment.

Single Application Assignment (SAA): $\forall f \in F : r(f) = \frac{1}{W_f}$. SAA is a feasible rate assignment. Moreover, since $\frac{1}{W_f} \geq \frac{1}{W}$, the optimum total completion time is still guaranteed. When flows of different size are considered, we define the normalized rate to be $\bar{r}(f) = \frac{r(f)}{w_f}$. Theorem 1 shows that setting $\bar{r}(f) = \frac{1}{W_f}$ results in an optimal assignment, so SAA is optimal by our measure.

THEOREM 1. *Given a set of flows (size and route), assigning to each $f \in F$ a rate $r(f) = \frac{w_f}{W_f}$ (equivalently $\bar{r}(f) = \frac{1}{W_f}$) is feasible and achieves a completion time W, which is the minimum.*

PROOF. $\forall l \in L, \forall f \in F_l : r(f) \leq \frac{w_f}{W_l}$. Therefore, $\forall l \in L : \sum_{F_l} r(f) \leq 1$, so the assignment is feasible.

The link bearing W needs at least W units of time to transmit all of the applied data, so the lower bound on the completion time is W.

Finally, $\forall f \in F$, the completion time is given by $\frac{w_f}{r(f)} = W_f \leq W$, so the lower bound is achieved. □

COROLLARY 1. *A routing that minimizes W, combined with SAA, achieves the globally optimal completion time.*

Although our adaptive routing (Section 4) does not achieve the minimum W, the optimal rate assignment achieves the "topological limit" of any given traffic pattern and chosen routes.

Before presenting a distributed algorithm that calculates SAA rate assignment, we note the following:

1. SAA does not maximally exploit link capacity; i.e., some flows may be able to increase their rates without affecting others.

2. The algorithm is easily adapted to operate in a network with links of varying capacity by redefining link weights to be:

$$W_l = \frac{\sum_{f \in F_l} w_f}{c_l}. \tag{4}$$

5.2 Distributed SAA Algorithm

In SAA, the rate of flow f depends on a single parameter, W_f. Our goal is therefore to enable every flow to derive its W_f value. In order to do so, links must track their weights (similarly to the proposal for adaptive routing in Section 4), and flows must periodically check link weights along their paths. Note that a single probing at the starting time of a flow does not suffice, as flows are allowed to start at slightly different times.

The detailed behavior of flows and links is summarized in Algorithm 1. The subroutines are used for communication between a flow f and links in L_f. In fact, it is the source of f that communicates with network elements on f's path, but we find it convenient to use the flow-link terminology.

Each subroutine entails sending a control packet p to the destination of f, carrying information from f to links in L_f. At each link, p's arrival triggers action that affects the state of the link, and p potentially collects some data from the link. Once p reaches the destination, it is sent back to the source along an arbitrary (e.g., default) path with the collected data, not affecting link states. The collected data is returned to f at the end of the subroutine. Only a single control packet is sent per invocation of a subroutine. Consequently, there is at most a single outstanding control packet per flow.

[4]We assume potential disparities in units to be solved by appropriate constant coefficients.

[5]Flow size and interconnect speed allow us to neglect the difference between transmission and arrival time.

Algorithm 1: SAA

1: **Initialization (at network setup):**
2: $\forall l \in L : W_l \leftarrow 0$
3: **Upon the start of flow** f**:**
4: $W_f \leftarrow$ `AnnounceStart`(w_f)
5: $r(f) \leftarrow \frac{w_f}{W_f}$
6: **Periodically:**
7: $W_f \leftarrow$ `ProbeLinks`()
8: $r(f) \leftarrow \frac{w_f}{W_f}$
9: **Upon the end of flow** f**:**
10: `AnnounceEnd`(w_f)

`real` ***AnnounceStart***(w_f)
1: `send` $p : p.w_f \leftarrow w_f, p.W_f \leftarrow 0$
2: **for all** $l \in L_f$ `upon receipt of` p **do**
3: $W_l \leftarrow W_l + p.w_f$ /*Update weight*/
4: $p.W_f \leftarrow \max\{p.W_f, W_l\}$ /*Collect $\frac{1}{W_f}$*/
5: **return** $p.W_f$ `when` p `returns to the source`

`void` ***AnnounceEnd***(w_f)
1: `send` $p : p.w_f \leftarrow w_f$
2: **for all** $l \in L_f$ `upon receipt of` p **do**
3: $W_l \leftarrow W_l - p.w_f$ /*Update weight*/

`real` ***ProbeLinks***()
1: `send` $p : p.W_f \leftarrow 0$
2: **for all** $l \in L_f$ `upon receipt of` p **do**
3: $p.W_f \leftarrow \max\{p.W_f, W_l\}$ /*Collect $\frac{1}{W_f}$*/
4: **return** $p.W_f$ `when` p `returns to the source`

At the beginning of a communication phase, each flow (independently) executes `AnnounceStart()`, which updates the weight of links in L_f and collects an initial W_f. This procedure may use the first packet of the flow, and can thus be combined with establishing the route of f adaptively. When f ends, it updates the links again by executing `AnnounceEnd()`. In between, the flow samples the relevant weights using `ProbeLinks()`, which may be piggy-backed on data packets.

The use of ACKs to return the collected weights can sometimes be obviated, as follows. Let each data packet carry two weight fields $p.W_f^c$ and $p.W_f^u$. The first holds W_f as it was known to f when p left it, and the second collects the updated maximum encountered link weight. The destination returns an ACK with the weights only if $p.W_f^c \neq p.W_f^u$.

Calculation Time

Once all flows have traversed their paths and made their presence known, it only takes each flow f a single probing to acquire the correct rate, which is guaranteed to stay unchanged until f ends. To see this, note that flows traversing $l \in L_f$ for which $W_l = W_f$ cannot end before f. As a result, f is assigned the correct rate throughout its transmission.

The probing frequency can be tuned (statically or dynamically), and control packets can be piggy-backed on data packets. The probing takes a single round-trip delay, which comprises propagation and queueing. (Control packets may delay one another even if given absolute priority over data packets.) The queueing delay, however, is greatly reduced by the fact that there is at most a single outstanding control packet per flow.

It is interesting to try and estimate the flow length beyond which rate-calculation overhead is sufficiently low. We estimate that in a 4096-node InfiniBand cluster employing an extended fat tree topology and adaptive routing, the end-to-end delay, including calculation in switches by dedicated hardware, should be under $10\mu s$. Contention among control packets may increase this delay; however, as these follow the same path as the data packets, the resulting rates assigned to the flows will be smaller, thus commensurately prolonging their transmission time. Therefore, the queuing-free estimate can serve in conjunction with transmission time at line speed to estimate the flow size whose transmission time equals the rate calculation time. Assuming $10\mu s$ and a line speed of 10Gb/s, this size is 100 kilobits.

It should furthermore be noted that the calculation time of SAA is by far lower than that existing schemes, which require multiple round trips to converge. We also note in passing that if the traffic pattern remains the same for several communication phases, rate assignments can be reused. Finally, in certain applications the rate assignment for the communication phase can be calculated during the preceding computation phase, thereby completely hiding the rate calculation latency.

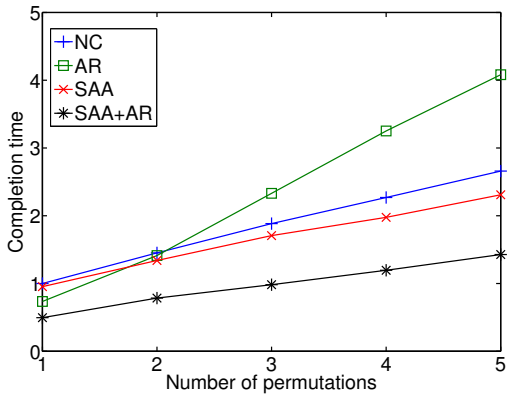

Figure 6: SAA performance

5.3 Simulation Results

We tested the effect of SAA and adaptive routing on the duration of the communication phase in a modified 16-ary 3-tree (4096 end-nodes). The application was assumed to send equisized flows that jointly constitute a superposition of random permutations. Thus, the number of flows sent and received by every end node equals the number of permutations, but link loads vary. Switches' input buffer sizes were set to eight MTU packets, sufficient for realizing the calculated rates (Section 6).

Fig. 6 depicts the completion time (in normalized units, averaged over 50 runs) for different control schemes. Interestingly, SAA alone (without adaptive routing) provides a small improvement of up to 13% over the baseline NC (no rate control, no adaptive routing), and adaptive routing alone (AR) even increases the completion time with three or more permutations. However, when the two are combined (SAA+AR), they yield a major improvement of up to 50%.

We explain this apparently counter-intuitive behavior as follows. Adaptive routing reduces the maximum contention in the network. However, it significantly increases the path lengths of some flows. Those flows have an increased prob-

ability of being affected by congestion spreading, and are likely to prolong the communication phase. Finally, when adaptive routing is used in conjunction with rate control (SAA), congestion spreading is avoided, and the reduced contention causes a major improvement in the total completion time. Thus, for a single permutation, SAA+AR reduces the completion time twofold compared to NC, thus fully exploiting the twofold reduction in maximum contention achieved by the adaptive routing.

6. REALIZATION OF CALCULATED RATES

6.1 Discrete vs. fluid model

The discussion so far, including simulations, was based on a "fluid model", whereby it is assumed that any feasible set of flow rates can be implemented. While facilitating the formal reasoning, this does not accurately reflect reality for three main reasons. First, the packet injection policy at sources determines the size and frequency of bursts of injected traffic. (We focus on a policy for individual sources, since it is impractical to coordinate injection of multiple sources.) Second, buffering can smooth quantization-related phenomena. Unfortunately, as mentioned in Section 1, InfiniBand switches typically have relatively small buffers. Finally, the scheduling of packets contending for the same output port may affect the ability of the fabric to keep up with the injected traffic.

In this section, we show that our simplifying assumption is justified. To this end, we developed a packet-injection scheme and used it in simulations that do capture the discrete nature of the network, including finite buffers and scheduling.

The goal of our injection scheme is to restrain the burstiness of the traffic in order to reduce the dependence on buffer smoothing. Note that because long flows are considered, there is no point in allowing flows to have small bursts (as in Leaky Bucket [41]) anyway. Therefore, we created an adapted version of *Shaped Virtual Clock* (SVC) injection scheme [6]. Basically, we let sources transmit packets periodically (according to the aggregate rate of all their flows), while applying a selection (among flows) mechanism for every transmission.

The injection scheme is used only by sources; these, unlike switches, are aware of the rates of their flows. Output ports of switches are assumed to employ FCFS policy on the packets arriving from different input ports. In addition, we assume that any number of packets can move simultaneously from an input buffer to *different* output ports ("infinite speedup"), yet an output port can only transmit a single packet at any given time.

6.2 Injection Scheme

Consider a single flow f with packets of fixed size d_p. The source can set the inter-packet delay (IPD) to $\frac{d_p}{r(f)}$ and transmit a single packet every IPD. This approach can be trivially extended to deal with several flows having the same rate. However, if flows have different rates, even with equal packet sizes, their effective multiplexing onto a single output link is challenging.

Let $D(f,t)$ denote the amount of data sent by the source for a flow f until time t. In SVC, whenever a source ends transmission of a packet, the next packet is chosen from a flow f that satisfies the following conditions: 1) f is "eligible" to send a packet; i.e., the time that passed from the last transmission of f's packet is at least $\frac{d_p}{r(f)}$ (the transmission of the new packet would not violate f's rate), and 2) $f = \arg\min\left\{\frac{D(f,t)}{r(f)}\right\}$ among eligible flows. If no eligible flow is found, the transmission of the next packet is delayed until one of the flows becomes eligible. Note that while SVC prevents bursts of individual flows, it allows bursts of aggregate traffic leaving the source.

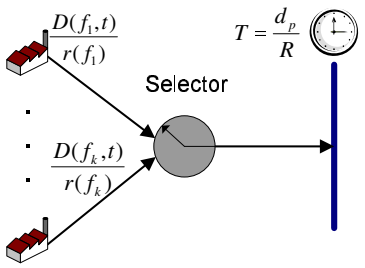

Figure 7: Periodic Selection injection scheme

We propose the *Periodic Selection (PS)* injection scheme (Fig. 7). Here, unlike with SVC, the source transmits a single packet every $\frac{d_p}{R}$. Yet again, the transmitted packet is (logically) selected just before the transmission. The flow to provide a new packet at time t is $f = \arg\min\left\{\frac{D(f,t)}{r(f)}\right\}$. In this manner the aggregate traffic leaving the source has a periodic nature as well.

If a source cannot transmit due to back pressure, a packet's transmission is delayed until free buffer space on the receiving side is available again. Once possible, the source will resume its periodic operation, counting the period from the time of the actual transmission, without attempting to compensate for the lost work. Therefore, such waiting will postpone the completion of the delayed source's flows.

The PS scheme can be implemented in two different practical ways. The simple implementation performs selection of the next packet by comparing all flows when a transmission of a current packet begins. If dedicated hardware is used for that purpose, the comparison can be performed in logarithmic time on a comparison tree. Otherwise, the flows must be compared serially, which might increase the delay considerably. Another alternative is to hold flows in a sorted list structure. The head of the list provides the next packet, and when a new packet is removed for transmission, the head is moved to a new place as its $\frac{D(f,t)}{r(f)}$ changes. (This ratio remains unchanged for other flows.) For more details on the injection scheme, see [5].

6.3 Simulation results

We tested our injection scheme in the same experimental setup as described in Section 5. The injection scheme was employed only by sources that, as opposed to switches, are aware of the rates of their flow. Output ports of switches were assumed to employ FCFS policy on the packets arriving from different input ports. We also assumed that any number of packets can move from input buffer to *different* output ports ("infinite speedup"), yet an output port can only transmit a single packet at any given time.

Table 1: Ratio of minimum measured and calculated rates

n \ b	2	4	8	16
1	0.38	1.00	1.00	1.00
2	0.19	0.64	1.00	1.00
3	0.18	0.31	1.00	1.00
4	0.18	0.25	1.00	1.00
5	0.20	0.39	1.00	1.00

All flows consisted of 2KB packets. The number of flows per source n, and the size of the input buffers in switches b (in units of 2KB), were used as test parameters. In all tests, our PS injection scheme was used to try and generate the rates assigned by SAA+AR, and the actual outcome was measured. All results were averaged over 50 runs.

Since the completion time is dictated by the lowest-rate flow, in each run we collected the ratio between the minimum measured and calculated rates (ratio of the minima). The collected results (averaged over 50 runs) are presented in Table 1. We see that with eight or more buffers per input port, the ratio is constantly 1, indicating that the achieved completion time effectively equals the optimal one.

7. CONCLUSIONS

Congestion in high-speed computer-cluster interconnects cannot be overcome effectively with current schemes. In this paper, we considered the important HPC scenario of a single phase-based application for which the completion of the last flow matters.

We proposed novel adaptive routing and rate calculation algorithms. On a slightly augmented 16-ary 3-tree implementing a 4096-node fat tree, adaptive routing alone was shown to be effective at mitigating the "topological" congestion (reduce by some 50% in return for a 10% increase in the number of switch ports). However, due to the possibility of oversubscription to communication resources and the resulting congestion spreading, it was shown to be completely ineffective at shortening the communication phase. Explicit rate calculation alone is of also of limited benefit. Interestingly, the combination of the two was shown to be very effective, reducing the duration of the communication phase by some 50%.

The 16-ary 3-tree topology used with 4096 nodes used in the simulations is highly representative of current computer clusters, and the slight topological extension proposed for effective adaptive routing only entails a 10% increase in the number of switch ports. Finally, InfiniBand, whose architecture was used as the base model, is the leading technology in current clusters.

Our proposed PS packet-injection scheme furthermore showed that the proposed schemes are practical. We showed that the calculated injection rates can be closely approximated even with a very limited number of buffers in switch ports.

The applicability of explicit rate calculation depends on the convergence time of the algorithms relative to flow durations. Our algorithm provides a correct output within a single round-trip time (a few microseconds) after all flows enter the network, which is by far faster than reactive schemes that moreover result in sub-optimal rates. We also note in passing that whenever control messages are slowed down by contention among themselves for a shared link, the same will be true for the flows. Thus, the range of flow size for which the overhead is sufficiently low is insensitive to "topological" congestion and perhaps even benefits from it. Calculating rates before the start of communication phase, e.g., during the previous computation phase, can hide the rate calculation latency altogether.

Finally, we point out that while the SAA rate-assignment algorithm is tailored to a single phase-based application, the adaptive routing and packet injection schemes are broadly applicable.

The promise of the proposed schemes calls for further investigation. Topics for further research include: 1) a deeper examination of the implementation details, 2) tests with real-life benchmarks, 3) application of the adaptive routing in other topologies, 5) testing the realizability of calculated rates under finite speedup and arbitration restrictions.

Acknowledgment. The authors are grateful to Mellanox Technologies, and particularly by Eitan Zahavi, for stimulating discussions.

8. REFERENCES

[1] "http://www.top500.org."

[2] InfiniBand^{TM} Trade Association, "Infiniband^{TM} architecture specification, release 1.2," InfiniBand Trade Association, Oct 2004.

[3] IEEE Computer Society, "IEEE Std 802.3^{TM}-2005," Dec 2005.

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

[5] V. Zdornov and Y. Birk, "Mitigating congestion in high-speed interconnects for computer clusters," Technion – Israel Institute of Technology, CCIT 723, Mar 2009.

[6] D. Stiliadis and A. Varma, "A general methodology for designing efficient traffic scheduling and shaping algorithms," in *Proc. IEEE INFOCOM*, 1997, pp. 326–335.

[7] "http://www.omnetpp.org."

[8] F. Petrini and M. Vanneschi, "k-ary, n-trees: High performance networks for massively parallel architectures," in *Proc. 11th Int'l Parallel Processing Symp. (IPPS)*, 1997, pp. 87–.

[9] C. E. Leiserson, "Fat-trees: Universl networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. c-34, no. 10, Oct. 1985.

[10] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proc. 16th Annual Symp. on Foundations of Computer Science (SFCS)*, 1975, pp. 184–193.

[11] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.

[12] T. Skeie, O. Lysne, and I. Theiss, "Layered shortest path (lash) routing in irregular system area networks," in *Proc. 16th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2002, pp. 162–169.

[13] A. Mejia, J. Flich, J. Duato, S. A. Reinemo, and T. Skeie, "Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori," in *Proc. 20th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Apr 2006, p. 10pp.

[14] S. Yan, G. Min, and I. Awan, "An enhanced congestion control mechanism in infiniband networks for high performance computing systems," in *Proc. 20th Advanced Information Networking and Applications (AINA)*, vol. 1, 2006, pp. 845–850.

[15] C. Gomez, F. Gilabert, M. E. Gomez, P. Lopez, and J. Duato, "Deterministic versus adaptive routing in fat-trees," in *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS)*, Mar 2007, pp. 1–8.

[16] J. Kim, W. J. Dally, and D. Abts, "Adaptive routing in high-radix clos network," in *Proc. ACM/IEEE Conf. on Supercomputing (SC)*, 2006, p. 92.

[17] J. C. Martínez, J. Flich, A. Robles, P. López, and J. Duato, "Supporting fully adaptive routing in infiniband networks," in *Proc. 17th Int'l Symp. on Parallel and Distributed Processing (IPDPS)*, 2003, p. 44.1.

[18] J. Liu, A. Vishnu, and D. K. Panda, "Building multirail infiniband clusters: Mpi-level design and performance evaluation," in *Proc. ACM/IEEE Conf. on Supercomputing (SC)*, 2004, pp. 33–33.

[19] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A multiple lid routing scheme for fat-tree-based infiniband networks," in *Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS)*, 2004.

[20] A. Vishnu, M. Koop, A. Moody, A. R. Mamidala, S. Narravula, and D. K. Panda, "Hot-spot avoidance with multi-pathing over infiniband: An mpi perspective," in *Proc. Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, 2007, pp. 479–486.

[21] B. V. Dao, S. Yalamanchili, and J. Duato, "Architectural support for reducing communication overhead in multiprocessor interconnection networks," in *Proc. 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA)*, 1997, p. 343.

[22] Y. Turner and Y. Tamir, "Connection-based adaptive routing using dynamic virtual circuits," in *Proc. International Conference on Parallel and Distributed Computing and Systems*, 1998, pp. 379–384.

[23] D. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, 1989.

[24] 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, no. 8, pp. 1465–1480, 1995.

[25] S. Floyd, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, 1999.

[26] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. ACM SIGCOMM*, 2002, pp. 89–102.

[27] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proc. IEEE INFOCOM*, vol. 4, 2004, pp. 2514–2524.

[28] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 37–48, 2005.

[29] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-end congestion control for infiniband," in *Proc. IEEE INFOCOM*, vol. 2, 2003, pp. 1123–1133.

[30] G. Pfister, M. Gusat, and D. Craddock, "Solving hot spot contention using infiniband architecture congestion control," in *Proc. High Performance Interconnects for Distributed Computing Workshop*, 2005.

[31] A. Charny, D. Clark, and R. Jain, "Congestion control with explicit rate indication," in *Proc. IEEE Int'l Conf. on Communications (ICC)*, 1995, pp. 1954–1963.

[32] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan, "An efficient rate allocation algorithm for ATM networks providing max-min fairness," in *Proc. HPN*, 1995, pp. 143–154.

[33] A. Charny, K. K. Ramakrishnan, and A. Lauck, "Time scale analysis scalability issues for explicit rate allocation in atm networks," *IEEE/ACM Trans. Netw.*, vol. 4, no. 4, pp. 569–581, 1996.

[34] R. Jain, S. Kalyanaraman, and R. Viswanathan, "The OSU scheme for congestion avoidance in ATM networks using explicit rate indication," in *Proc. WATM First Workshop on ATM Traffic Management*, 1995.

[35] H. Tzeng and K. Siu, "Adaptive proportional rate control for abr service in atm networks," in *Proc. IEEE Computers and Communications*, 1995, pp. 529–535.

[36] Y. Afek, Y. Mansour, and Z. Ostfeld, "Phantom: a simple and effective flow control scheme," in *Proc. ACM SIGCOMM*, 1996, pp. 169–182.

[37] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore, "The erica switch algorithm for abr traffic management in atm networks," *IEEE/ACM Trans. Netw.*, vol. 8, no. 1, pp. 87–98, 2000.

[38] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, 2006.

[39] R. Jain, "Congestion control and traffic management in ATM networks: Recent advances and A survey," *Computer Networks and ISDN Systems*, vol. 28, no. 13, pp. 1723–1738, 1996.

[40] L. Mamatas, T. Harks, and V. Tsaoussidis, "Approaches to congestion control in packet networks," *Journal of Internet Engineering*, vol. 1, no. 1, pp. 22–33, 2007.

[41] J. Turner, "New directions in communications (or which way to the information age?)," vol. 24, no. 10, pp. 8–15, Oct 1986.