Comparative Evaluation of CEE-based Switch Adaptive Routing

Daniel Crisan, Mitch Gusat, Cyriel Minkenberg
IBM Research, Zürich Research Laboratory
Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
{dcr,mig,sil}@zurich.ibm.com

Abstract—**In datacenter networks routing is key to applications' performance, stability and reliability. IEEE Data Center Bridging Task Group is standardizing essential components of Converged Enhanced Ethernet (CEE) for use as universal datacenter technology. As the first adaptive routing proposal for CEE, the recent introduction of Switch Adaptive Routing (Switch AR) has the potential to enable new CEE applications from the realms of Cloud and High Performance Computing (HPC). We compare its performance versus the current state of the art in routing algorithms. We assess three most representative deterministic and load-oblivious schemes, as well as a rate limiter-enhanced version of Switch AR – all customized for CEE networks.**

Using HPC application traces and synthetic benchmarks borrowed from the InfiniBand and the 802.1Qau test suites, we evaluate the performance benefits and drawbacks of each scheme. We prove that Switch AR achieves better performance than deterministic and load-oblivious when asymmetric loads can be encountered on different alternative paths. On the other hand, when the loads are uniform, oblivious load-balancing algorithms deliver the best performance.

I. Introduction

Nowadays datacenter networks and HPC installations are composed of multiple disjoint networks: (i) a Local Area Network – Ethernet or Gigabit Ethernet, (ii) a System Area Network – Myrinet, Quadrics or InfiniBand, and (iii) a Storage Area Network – FibreChannel or InfiniBand. The *convergence* of all these networks into a single one is the solution to reduce cost, complexity and power consumption.

The technology recently promoted by the industry and standardized by IEEE as universal network fabric is the Converged Enhanced Ethernet. CEE provides a unified Layer 2 network that carries all the traffic generated by the applications running in a datacenter using a single physical infrastructure. Essential components of CEE are currently being standardized by the IEEE 802 Data Center Bridging Task Group defining the specifications for per-priority link-level flow control (LL-FC) and traffic differentiation (802.1Qbb), Quantized Congestion Notification (QCN) congestion management mechanism (802.1Qau), and Enhanced Transmission Selection (ETS) (802.1Qaz).

As the first adaptive routing proposal for CEE, the recent introduction of Switch Adaptive Routing (Switch AR) has generated interest, together with the potential to enable new CEE applications from the realms of HPC and Cloud. To fully realize this potential we must comprehensively answer a performance question: (**Q1**) How does Switch AR compare against the state of the art routing schema, currently used in HPC and Cloud? Whereas Switch AR's merits versus QCN have been previously established [1], [2], here we shall conduct a more exhaustive comparison, including most, if not all, of the competing schema, using a wider set of benchmarks. Contingent to Q1 is the related question: (**Q2**) Which are Switch AR's primary counter-candidates? To answer this question we begin with a brief survey of routing for HPC and datacenter interconnects.

Network topologies used for datacenters typically provide path redundancy for fault resilience and higher bisection bandwidth. Datacenter routing schemes exploit this path redundancy to minimize congestion, increase throughput and lower latency. Deterministic routing algorithms aim to distribute the load statically from different sources/destinations over different paths. Oblivious load-balancing routing algorithms exploit path redundancy by distributing the flows from each source across the alternative paths without considering network state or traffic load. Switch AR enables the CEE switches to select alternative paths by snooping the congestion notifications that pass through them. Thus the load is dynamically distributed in response to congestion feedback.

We analyze these routing algorithms and reveal their specific benefits and drawbacks compared to Switch AR. Given the limitations of the available analytical and NS-2/3 models, we use the Venus-ZRL event-based simulation environment [3] for modeling. For evaluation we use HPC application traces and synthetic benchmarks previously employed for 802 and InfiniBand simulations. We focus on fat-trees and standard 10Gbps Converged Enhanced Ethernet.

We make the following contributions: In Section II we review the main features of CEE datacenter networks by presenting the topology, the LL-FC, QCN and ETS mechanisms. First contribution is a brief survey; in Section III we introduce the routing algorithms under evaluation. As second contribution, we enhance the original Switch AR by adding QCN-compliant rate limiters. Section IV describes our simulation environment and the traffic benchmarks. Finally, as our third contribution, we perform a rigorous quantitative evaluation of all the routing schema presented and analyze the results in Section V. We present related work in Section VI and conclude in Section VII.

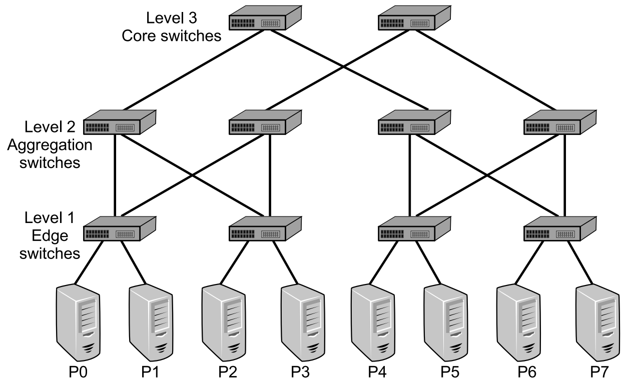


Figure 1. Multi-tiered datacenter with edge, aggregation and core switches.

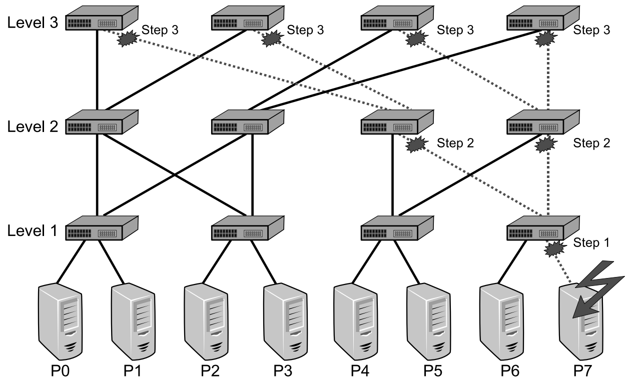


Figure 2. Saturation tree formation in a fat-tree network. End-node P7 is slow in processing incoming packets sent from nodes P0 ... P3. In the first step, congestion appears on the edge link connecting P7 with the corresponding level 1 switch. In the second step, the dashed links from level 2 to level 1 are affected. In the third step, congestion propagates up to root level. Other flows that do not even target P7 are now potentially affected (e.g. P3 $\rightarrow$ P4). If the initial hotspot persists long enough, the domino effect created by the LL-FC continues and the congestion propagates further back to level 2 and level 1 (not drawn). The network experiences a catastrophic loss of throughput affecting all the nodes.

II. Datacenter networks

Future datacenters will typically consist of 1K-100K processing nodes connected by a converged network fabric. The classical topology for datacenter networks follows a tiered approach as in Figure 1. The processing nodes are connected to edge switches that provide the connectivity between the nodes collocated in the same rack. The edge switches in turn are connected to an intermediate layer of aggregation switches that connect different racks together in a cluster. The clusters can be further linked through another layer of core switches [4], [5].

The packets generated by one of the processing nodes have to traverse a few levels of switches before reaching the destination nodes. A common multi-stage interconnect is the fat-tree [6]; hence we will focus on this topology in this paper.

A. Topology: Fat-Trees

A k-ary n-tree consists of $N = k^n$ processing nodes and $n \cdot k^{n-1}$ switch nodes. The switch nodes are organized on n levels, each level having k^{n-1} switches. All switches have the same arity $2k$, excepting the top switches, which have arity k. This type of network has constant bisection bandwidth and path redundancy [7].

The k-ary n-trees and their slimmed versions belong to the family of extended generalized fat-trees as explained in [6]. An example of a 2-ary 3-tree can be seen in Figure 2.

A deadlock free path in a fat-tree is computed by selecting an intermediate switch from the set of Nearest Common Ancestors (NCA) of the source and the destination node [8]. A NCA is a common root of both the source and the destination located at the lowest possible level. Packets are following an up-phase from the source to the NCA, and then a down-phase from the NCA to the destination node.

For example in Figure 1, to send data from source P0 to source P7, there are two NCAs: the two switches on level 3. Packets leaving source P0 travel upward until they reach one of the level 3 switches, then downward to destination P7.

B. Converged Enhanced Ethernet (CEE)

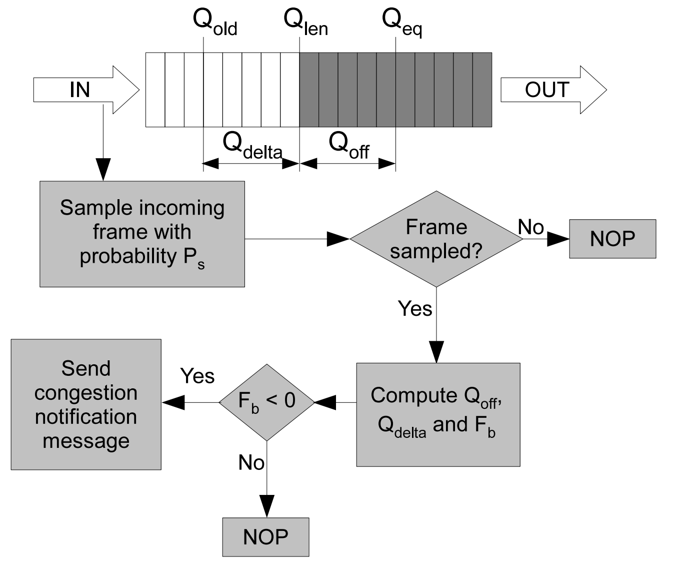
CEE comprises the following improvements made to the traditional Ethernet: (i) per-priority link-level flow-control and traffic differentiation (802.1Qbb) [9], (ii) congestion management (802.1Qau) [10], (iii) enhanced transmission selection (802.1Qaz) [11]. New virtualization efforts are on the way in 802.1.

1) Link-Level Flow-Control (LL-FC): In order to improve performance and reliability, the CEE networks are specifically designed to prevent frame losses, by using the LL-FC mechanism. It works by pausing the transmission on an input link when the corresponding buffer occupancy exceeds a certain threshold, here called high watermark. The transmission is paused using a special control frame sent to the upstream device. There are two effects to this approach. The desired effect is that the congestion information is propagated from the congestion point to the upstream devices. Hence, eventually the core congestion is pushed to the network edge.

On the other hand, when a link is paused, the buffers of the upstream device fill up and new upstream links will have to be paused. This has the potential to continue recursively affecting more and more devices. Therefore if the congestion persists, it can spread from one network device to another forming a congestion tree [12], [13]. Previous studies [12] showed that a congestion tree can fill all the buffers in only a few round-trip times, too fast for software to react. An example about how a congestion tree can form in a network is shown in Figure 2.

This undesired effect of LL-FC can cause a catastrophic loss of throughput of the entire network. To make the situation worse, after the original congestion subsides, the congestion tree dissipates slowly, because all the buffers involved must first drain [14].

The LL-FC mechanism was extended to support multiple priority classes, i.e., Priority-based Flow Control (PFC): each priority class has its own buffer and LL-FC. The traffic is differentiated in 16 priority classes. Congestion generated by



Q_{old}
Q_{len}
Q_{eq}
IN
OUT
Q_{delta}
Q_{off}
Sample incoming frame with probability P_s
Frame sampled?
No
NOP
Yes
Send congestion notification message
Yes
$F_b < 0$
Compute Q_{off}, Q_{delta} and F_b
No
NOP

Figure 3. QCN load sensor mechanism. The sampling rate P_s is a function of the measured feedback F_b. For low congestion levels, one every 100 received frames is sampled. Sampling rate increases linearly with the severity up to one every 10 received frames.

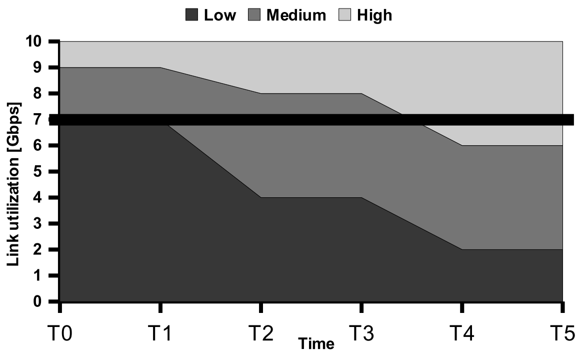


Figure 4. Enhanced Transmission Selection bandwidth allocation for a 10Gbps link with three priorities: low, medium and high. The offered traffic intensity for the low-priority is constant: 7Gbps. Before T1, the intensity of the medium and high-priority traffic allows the low-priority to receive the required bandwidth of 7Gbps. After T1, the intensity of the medium and high-priority traffic increases gradually, thus reducing the capacity left to the low-priority. The low-priority traffic encounters a bottleneck.

a priority class does not influence the other priority classes.

2) Quantized Congestion Notification protocol (QCN): QCN is a congestion management mechanism defined in the IEEE standard 802.1Qau [10]. The final version of the standard provides a set of protocols and procedures for congestion management of long-lived data flows. QCN-compliant switches can detect and signal congestion to the end-nodes. The QCN-capable end-nodes respond to the congestion information by limiting their transmission rate.

QCN consists of two algorithms:

(i) Congestion detection: This is a mechanism used to observe the state of the network and to detect congestion. Each switch samples the incoming frames with a variable and randomized sampling rate (see FIGURE 3). For each frame sampled, the switch measures the output queue occupancy and computes a feedback value F_b. The feedback value is computed as $F_b = -(Q_{off} + w \cdot Q_{delta})$. Q_{off} is the difference between the measured queue occupancy Q_{len} and an equilibrium queue occupancy Q_{eq}, considered normal during the operation: $Q_{off} = Q_{len} - Q_{eq}$. Q_{delta} is the change of the queue occupancy from the preceding sample instant: $Q_{delta} = Q_{len} - Q_{old}$. If the computed F_b is negative, the switch generates a congestion notification message, sent back to the end-point that generated the sampled frame, i.e., culprit source. Hence the congestion notification informs the source about the hotspot, essentially conveying its location via the Congestion Point ID, whereas the feedback value F_b provides a 6-bit quantitative indication of how severe the bottleneck is. When a higher precision is required, the congestion notification also entails two 16-bit values, i.e., the raw queue offset and delta.

(ii) Source reaction: This is a mechanism by which the source limits its transmission rate in response to the congestion notifications received from the QCN-enabled switches. When a notification is received, the source instantiates a Rate Limiter (RL) that adjusts the transmission rate according to the feedback received: the higher the feedback, the higher the rate reduction. The RL also provides a way to recover: if no congestion notification has been received for a certain number of sent frames, it can be assumed that congestion has vanished and the source can increase its transmission rate.

The above description shows that the QCN algorithm matches the transmission rate of an end-node with the available bandwidth in the network. Unlike in earlier proposals (Ethernet Congestion Management – see [15] for a full description) there is no positive feedback in QCN. Hence, the source has to recover the bandwidth autonomously. This rate recovery is performed in three phases. In the first phase (Fast Recovery), a few binary increase steps are performed similar to BIC-TCP [16]. In the second phase (Active Increase), several linear increase steps take place, followed by an optional superlinear increase regime (Hyper-Active Increase) in the third phase.

3) Enhanced Transmission Selection (ETS): The enhanced transmission selection mechanism provides a framework to support bandwidth allocation to traffic classes. This is needed because different applications have different bandwidth and latency requirements. For time-sensitive applications requiring minimum latency, a strict priority scheduling is needed. Active priorities that generate bursty traffic can share bandwidth. When a priority is not using its allocation other priorities can use the bandwidth. An example of the operation of ETS in a datacenter with three priority classes is depicted in FIGURE 4. Note how the low-priority traffic bandwidth slice shrinks under the pressure of the high-priority traffic.

III. DATACENTER ROUTING ALGORITHMS

Routing involves choosing a path from a source node to a destination node. The path is chosen in order to optimize a metric such as hop-count, latency or cost. In fat-trees, due to the high path redundancy, there are multiple shortest paths from source to destination. In this case it is desirable that the routing algorithm also performs the load-balancing between these alternative shortest paths.

According to whether the routing algorithm uses the load status information generated in the network, the routing schemes can be classified as oblivious (ignore state) or adaptive (use state) [17], [18].

In the following paragraphs we will describe the selected routing schemes. We assume that the network has a k-ary n-tree topology.

A. Deterministic routing

Deterministic routing always uses a single fixed path from a given source S to a given destination D. The choice of paths is done such that the load is distributed evenly across the switches that act as Nearest Common Ancestors (NCA) between different sources/destinations.

An extensively studied deterministic routing technique is the modulo-based D-mod-k routing [19], [20], [21] also know as Stage And Destination Priority - SADP [22], [23]. To establish a path $S \rightarrow D$, the algorithm chooses the parent $\left\lfloor \frac{D}{k^{l-1}} \right\rfloor \bmod k$ at the level l in the upwards phase of the routing until a NCA is reached. An example of D-mod-k routing is given in FIGURE 5. The NCA choice is dictated by the destination address. Consequently flows with different destinations use different NCAs and the traffic sent to different destinations is distributed statically over alternative paths.

Another approach uses the source address in the choice of the NCA. This is accomplished by the modulo-based S-mod-k routing that chooses the parent $\left\lfloor \frac{S}{k^{l-1}} \right\rfloor \bmod k$ at every level l in the upwards phase. Using this algorithm the flows with different sources use different NCAs.

Various studies [22], [24] proved that D-mod-k is one of the best performing deterministic routing algorithms. Additionally it has the advantage of in-order delivery, hence no need for resequencing buffers at the destination. Nonetheless, its throughput can suffer because of resource conflicts and head-of-line blocking. It is always possible to find particular traffic patterns under which two or more flows contend on the same link. Such conflicts are unavoidable owing to the static nature of the algorithm [24], [23].

B. Random routing

Random routing [25], [26], [27] uses all available paths from S to D with equal probability. This approach distributes the loads across the different links and switches.

The Valiant routing algorithm [25] states that, in a network with an arbitrary topology, for each packet from S to D, a random intermediate node R must be selected and the packet is then routed along the path $S \rightarrow R \rightarrow D$. In fat-trees the role of the intermediate nodes is taken by the NCAs. To route a packet from S to D, a random NCA is chosen and the packet sent through that NCA. The choice of NCA can be done at

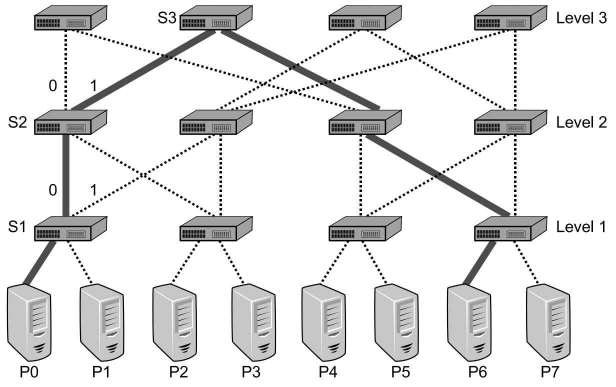


Figure 5. D-mod-k routing in a 2-ary 3-tree ($k = 2$). Source P0 sends a packet for destination P6 ($D = 110$). The packet arrives at switch S1 at level 1, which computes $\left\lfloor \frac{D}{k^0} \right\rfloor \bmod k = 0$ and selects parent #0. Then the packet arrives at switch S2 at level 2, which computes $\left\lfloor \frac{D}{k^1} \right\rfloor \bmod k = 1$ and selects parent #1. The packet reaches the root switch S3, which is NCA for P0 and P6. From the NCA, there is a single downward path available to the destination P6.

the source, as in [26], or at each step of the upward phase as in the Connection Machine CM-5 [27].

Misordering is possible, hence any traffic type that requires in-order delivery needs a resequencing buffer at the destination. The throughput may be reduced by uneven loading of the alternative paths. If one of the NCAs is loaded more than others, still it will receive the same amount of traffic, because the division is statical.

C. Hashed routing

Hashed routing is a special case of Equal-Cost Multi-Path routing detailed in [28]. In hashed routing, each flow from S to the D is characterized by a flow identifier. The source uses a hash function that takes as input a flow identifier and outputs a path selected from the set of alternative paths to the destination. The flow is usually identified by a 5-tuple containing the source and destination address (Layer 2 or 3), the source and destination port, and the protocol number.

For fat-trees, the number of alternative paths is determined by the number of NCAs. Hence the hash function has to select an NCA for each flow identifier given as input. This distributes the flows evenly over different links. Since all packets of a flow follow the same path, as in deterministic routing, no resequencing is required. Care is still needed for a flow level ordering.

Hashed routing performs similar to random routing for sources that generate a large number of “mice” flows, which will select different paths because of hashing, thus the load is distributed across the network. On the other hand, if the number of flows between S and D is small, hashing degenerates into deterministic routing and a single path will be used.

D. Switch-based adaptive routing (Switch AR): The Original and the RL-enhanced Version

Switch AR [1], [2] uses the QCN congestion information to steer the traffic. Switches are QCN-enabled and continuously monitor the status of their outbound queues. If congestion is detected in one of these queues, the switch generates a congestion notification that travels upstream from the congestion point to the originating source of the packets deemed as culprits.

Congestion notifications are snooped by the upstream switches, which thus learn about the downstream congestion. When a switch detects congestion, it can reroute the traffic to alternative paths, to avoid the hotspots, and to allow the congested buffers to drain. In this way, the path diversity is exploited, theoretically better than by the load-oblivious schemes.

If the new path, however, is also congested, the Switch AR algorithm will revert to the original path, hence oscillations are possible. These are likely to appear in networks with multiple hotspots or, when multiple flows contend as observed in [29].

The switch uses the snooped congestion information to annotate its routing table with a congestion level for each port through which a given destination is reachable. When a frame for this destination arrives, it will be routed through the port with the minimum congestion level. By marking the ports as congested with respect to each destination, the switch reorders its routing preferences to favor the uncongested ports. The algorithm must ensure that all the upstream switches learn about congestion. Congestion notifications are routed randomly towards the culprit source. Thus all the upstream switches of all the alternative paths can detect the hotspot.

The Switch AR algorithm is using binary route split ratios. When congestion is detected on one path, the entire traffic flow is switched to an alternative path. The advantage is in simplicity and low-cost. The resources required are minimal because only some additional per-port data needs to be stored [1], [2]. Another advantage is that the switch forwarding logic is unchanged. Only the routing table is updated in response to congestion notifications.

One can argue that the binary split can lead to oscillations. In order to avoid oscillations, fractional route split ratios can be used: in response to congestion only a fraction of the traffic is rerouted to an alternative path. However, this comes at a higher cost, as the switch will have to store per-flow information. Extensive changes to the forwarding logic are required. For example, in order to divert 20% of the traffic to an alternative path, every 10 packets, 2 packet have to be rerouted.

We are currently working to improve Switch AR's stability by a less aggressive load re-routing decision while keeping the hardware requirements as low as possible.

Since Switch AR is a route-only scheme, we also want to test its coupling with a congestion management scheme such as the QCN. Therefore we devise a new version, called Switch AR with Rate Limiters, whereby QCN compliant rate limiters can be instantiated at each source, for each "culprit" flow. These flows are identified solely using the destination address carried by the congestion notifications. When a congestion notification is received, a rate limiter is instantiated at the flow source. The algorithm used for rate limitation is identical with the one described in the 802.1Qau standard.

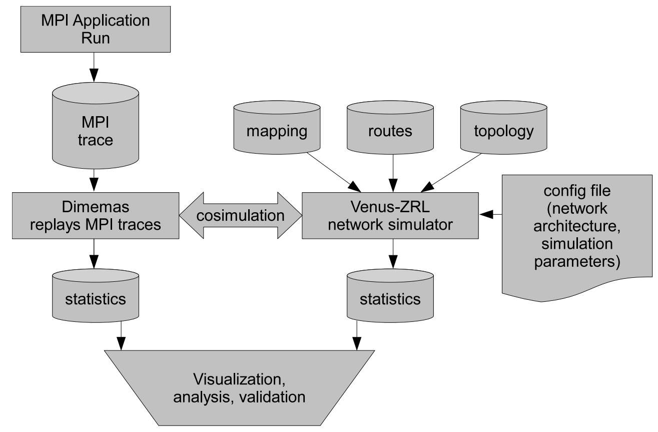

MPI Application Run
MPI trace
mapping
routes
topology
Dimemas replays MPI traces
cosimulation
Venus-ZRL network simulator
config file (network architecture, simulation parameters)
statistics
statistics
Visualization, analysis, validation

Figure 6. The structure of the Venus-ZRL simulation environment. The MPI applications are run on a real parallel machine. Traces of the MPI calls are stored in files, which are replayed by the Dimemas simulator. Venus-ZRL initializes the simulation using the provided topology, routes, mappings and configuration file. The messages generated by Dimemas are transported by the network simulated in Venus-ZRL and eventually returned to Dimemas. Both simulators output statistics that can be visualized with specific tools and used for analysis and validation.

IV. Evaluation methodology

A. *Simulation environment*

To replay network traces and to study the routing effects on the application performance, we used two simulators coupled in an end-to-end simulation environment: Dimemas and Venus-ZRL [3].

Venus-ZRL is an event-driven simulator developed at IBM Research – Zürich, capable of flit level simulations of processing nodes, switches and links. It is based on OMNeT++ [30], an extensible, C++ simulation library. It was developed as an extension of the Mars-ZRL network simulator [31].

Venus-ZRL supports various network topologies such as fat-trees, tori, meshes, and hypercubes. It can simulate different network hardware technologies, such as Ethernet, InfiniBand, and Myrinet. Additionally, it can also model irregular networks topologies and new types of hardware.

Dimemas is a Message Passing Interface (MPI) simulator that models the semantics of the MPI calls. The two simulators communicate through a co-simulation interface. When an MPI message is produced, Dimemas passes the message to Venus-ZRL, which models the segmentation, buffering, switching, routing, reassembly and eventually delivers the message back to Dimemas.

A brief scheme of the simulation environment is shown in FIGURE 6; a more detailed description can be found in [3].

Moreover, Venus-ZRL can operate as a standalone simulator; in this case the traffic is generated by synthetic traffic sources used to simulate various traffic patterns, such as Bernoulli, bursty, on/off or Markov traffic. Also we can simulate communication patterns ranging from simple permutations to more complex sweeping hotspot scenarios.

The simulation environment has already been tested in InfiniBand, Myrinet and 802/CEE simulations.

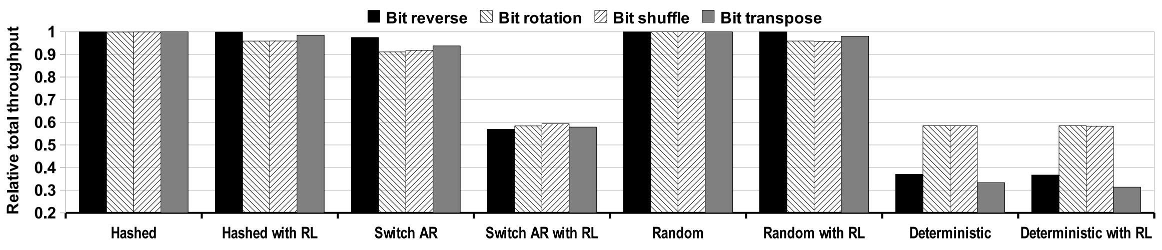


Figure 7. Permutation traffic relative total throughput. As reference we consider the total throughput achieved by the same traffic pattern on an ideal crossbar. Deterministic routing generates conflicts on multiple links. Because of its static nature it cannot avoid them, hence looses up to 70% of the throughput. Switch AR with RLs looses around 40% because sources can not decide whether the received congestion notifications were already used for traffic steering or not; hence, all of them are fed into the rate limiters.

B. Network model

The network we are modeling in Venus-ZRL has the following components:

(1) Processing nodes – The processing nodes are the sources and destinations of the network traffic. They are assumed to have an infinite bandwidth link with the network adapters. In the processing nodes, we gather statistics such as the delay. They are computed as the difference between the time the packet was generated at the source and the time it was received at the destination. Thus we make sure the simulator also accounts for the time spent by the packets waiting before entering the network.

(2) Network adapters – The network adapters are responsible for link-level flow control and the source reaction algorithm for congestion management. Out-of-order arrivals are resequenced in the receive buffer.

(3) Switches – The switches are network devices that transfer packets from their input links to the output links. They are responsible for link-level flow control, contention resolution and congestion detection. In switch-controlled routing schemes, also the routing decisions are made by the switches.

Switches are modeled as ideal input-buffered output-queued switches (IBOQ). When a packet arrives on an input link, it is buffered in the input buffer associated with that link. Simultaneously, an output port is selected according to the routing algorithm in use. The incoming packet is enqueued in the output queue associated with the output port selected. If the output queue is empty, the packet will be sent out immediately. If there is contention on the output port, the packets will be sent out in FIFO order. The switch we model implements a cut-through switching policy.

There are two main differences between the ideal switch we are using and a real switch:

- The input bandwidth in each switch output queue is N times the line rate. Hence, it is possible for all input ports to simultaneously enqueue a packet in an output queue. In a real system, for a high arity switch, this is unrealistic because of physical limitations.
- The size of each switch output queue is only bounded by the sum of all the input buffers for all ports. In a real system, the size of the output queue is bounded to a smaller value than the sum of all the buffers capacities in the device. Hence a single output queue can not use the entire buffer memory in the device.

C. MPI Traces

We have selected nine different MPI applications. Five of them (BT, CG, FT, IS, and MG) are part of the NAS Parallel Benchmark [32] developed by NASA, targeting performance evaluation of highly parallel supercomputers. The other four applications (WRF, NAMD, LISO, and Airbus) are used by the research community for weather prediction and fluid dynamics parallel simulations.

The applications were first run on the MareNostrum supercomputer of the Barcelona Supercomputing Center. While the application was running, the MPI calls were recorded in a trace file. Afterwards, the traces are replayed in our simulation environment.

D. Simulation parameters

As previously explained, the congestion notifications are generated by the QCN-enabled switches. The algorithm used for congestion detection and reaction is QCN 2.2.

The network technology is 10Gbit Ethernet with 1500B maximum transmission unit. TABLE I contains the most important parameters for our simulations.

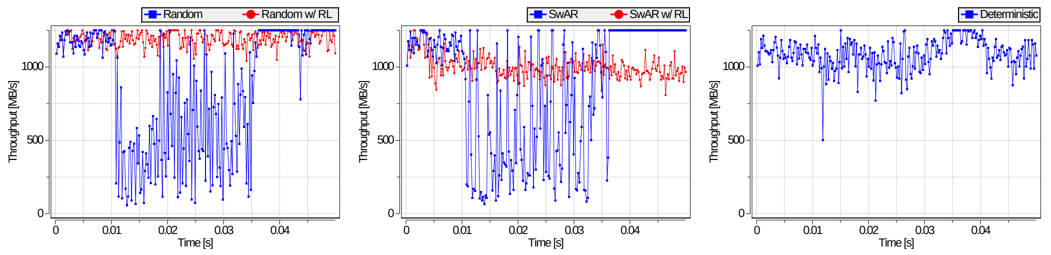
Table I
SIMULATION PARAMETERS

Parameter	Value	Unit	Parameter	Value	Unit
link speed	10	Gb/s	sample interval	150	KB
frame size	1500	B	buffer size/port	100	KB
min. rate	100	Kb/s	fast recovery thr.	5	
Q_{eq}	20	KB	byte count limit	150	KB
W_d	2		active incr.	5	Mb/s
G_d	0.5		hyperactive incr.	50	Mb/s
quantization	6	bit	min. decr. factor	0.5	
RL timer	15	ms	extra fast recovery	enabled	
AR timer	250	ms	LL-FC	enabled	

V. RESULTS

All the routing schemes described in SECTION III were implemented in Venus-ZRL. These schemes are a representative set of common load-oblivious and deterministic routing algorithms.

From the load-oblivious class, we assess random **(Random)** and hashed routing **(Hashed)**. From the deterministic class, we



(a) Tput random. (b) Tput Switch AR. (c) Tput deterministic.

Figure 8. Input generated hotspot at edge links: victim flow throughput. The ideal throughput is 95% of link speed (1187MB/s). Random and switch adaptive schemes without RLs generate large congestion trees. Consequently, the victim flow's throughput drops during the congestive pattern. Deterministic routing limits the expansion of the congestion tree.

evaluate D-mod-k routing **(Deterministic)**. Finally we assess the switch-based adaptive routing algorithm **(Switch AR)**. For all these algorithms, we consider one version *with* rate limiters and one version *without* rate limiters. The rate limited version will bear the suffix "**RL**".

To begin with, we measure the performance using a few synthetic benchmarks, as is typically done in this field. Next, we continue with trace-driven simulations.

A. Synthetic traffic

Some of the synthetic traffic patterns we use in the following subsections are part of the Hotspot Benchmark used by IBM Research and 802 Task Groups. One or more sources can generate a hotspot at a given location in the network by injecting a predetermined amount of (in)admissible traffic for that location. Flows that pass through the hotspot are referred as *hot flows*, while the others are referred as *cold flows*.

Hotspots are classified using the following criteria:

- *Type*
  – *Input Generated* – The inputs (sources) require more bandwidth than available in the network. Typical examples are the patterns when flows from different sources converge into the same link exceeding its capacity.
  – *Output Generated* – An output (network device) is slow in processing incoming packets. For example, a traffic destination can be slowed down because of a CPU overload. Another possible cause can be a switch servicing traffic from different priorities. In this case, output generated hotspots can appear because a part of the available bandwidth is reserved for the higher priorities.
- *Severity* – measures the ratio between offered and accepted traffic (the drain rate of the bottleneck link during congested phase).
  – *mild* – smaller than 2
  – *moderate* – between 2 and 10
  – *severe* – higher than 10
- *Degree* – the fan-in of the congestive tree at the hotspot (i.e. the percentage of all sources that inject hot flows into the hotspot).
  – *small* – less than 10%
  – *medium* – 20% to 60%
  – *large* – more than 90%

We use a network with 32 processing nodes connected by a 2-ary 5-tree network. This network has the round-trip time and average hop count of a large datacenter interconnect. In an average datacenter, the number of nodes is on the order of 10K, but also the arity of switches is much higher (32 to 64 ports). Because of these factors, a large interconnect will still have a small number of levels (3-5), as in our simulation.

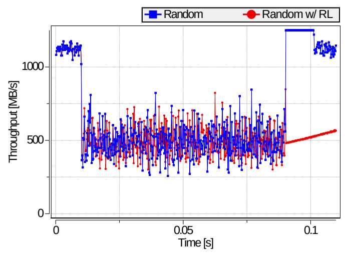
1) Permutation Traffic: We evaluate the routing schemes using both uniform and permutation traffic. All processing nodes inject traffic into the network at 90% of the link capacity. FIGURE 7 shows the results of this evaluation. For each routing scheme, we display the relative total throughput of all the flows in the simulation. As reference we consider the throughput achieved by running the same traffic pattern on an ideal crossbar network, with the same number of processing nodes. A description of the permutation patterns employed can be found in [17] CHAPTER 3.2 or [18] CHAPTER 9.2.

We also applied uniform traffic, which does not generate congestion. All routing schemes can successfully handle it, as the reference does. We omit these results as trivial.

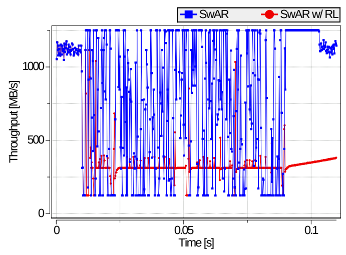
On the other hand, permutation traffic is difficult for the deterministic schemes. Because of its static nature the deterministic routing cannot reroute the traffic if two flows collide. For some permutations this generates massive losses of throughput. Also, switch-adaptive routing with rate limiters suffers a loss of throughput. This happens because in permutation traffic multiple hotspots are generated. Therefore it can happen that a flow that is being rerouted to avoid a congested link is routed to another congested link. Eventually this will activate the rate limiters.

The permutation traffic pattern is the quasi-worst-case scenario, because we do not always expect in normal datacenter operation that the nodes start communicating in synchronized manner – except some HPC and special applications.

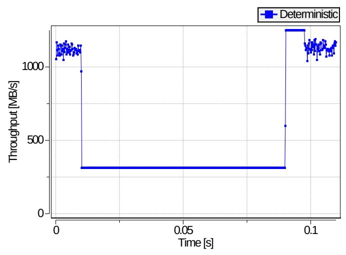
2) Input-generated hotspot at the edge links: The objective of this test is to check whether the routing algorithms are generating congestion trees. We create an input-generated hotspot of mild severity and small degree. To achieve this, we direct 45% of the traffic from 4 different nodes to a single destination node for 20 ms. The entire simulation lasts 50 ms. The congestion tree evolution can be directly visualized by inspecting the length of the queues for different switches



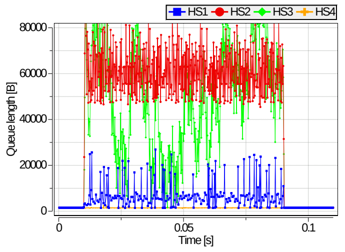
(a) Tput random.



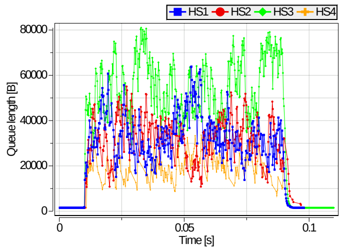
(b) Tput Switch AR.



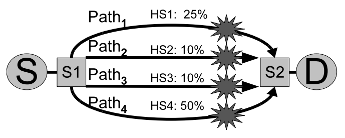
(c) Tput deterministic.



(d) Queue length random.



(e) Queue length Switch AR.



(f) Scenario: Four paths between S and D with different congestion levels.

Figure 9. Output-generated congestive pattern at root links. The ideal throughput is 90% of link speed (1125MB/s). Random routing (a) drops to 40% (500MB/s) because of head of line blocking in switch S1 (f). For random routing queues are unevenly loaded (d) producing out-of-ordering and jitter. Switch adaptive routing achieves on average 60% (750MB/s) and oscillates (b); its queues are more uniformly loaded (e). Deterministic routing uses only $Path_1$, hence reaches only 25% (312MB/s) (c).

during simulation, or indirectly by measuring the throughput of a cold flow.

We adopt the second method; in FIGURE 8 we show the throughput of the "victim" cold flow when using different routing schemes. The hot flows converge only on the edge link connecting the destination with the network. The cold flow does not pass through that link, hence its throughput should not be affected by the bottleneck. However, when a congestion tree is formed, many other links can saturate, as shown in FIGURE 2. Hence the cold flow can be indirectly affected by the secondary hotspots belonging to the same congestion tree.

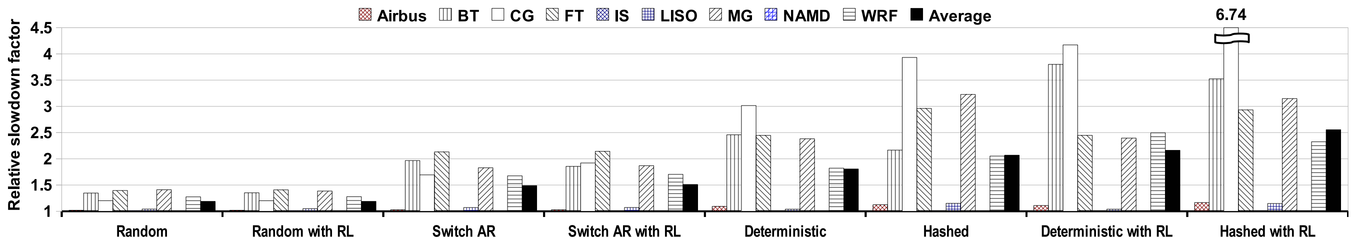
As seen in FIGURE 8(a) and FIGURE 8(b), both random routing without RL and switch adaptive routing without RL can generate congestion trees. On the other hand, activation of rate limiters will eliminate this issue. We observe from FIGURE 8(c) that deterministic routing is less strongly affected. This is because random routing and switch adaptive use of all the available alternative paths. Hence, they tend to spread the congestion. Deterministic routing uses a single path all the time, hence congestion is limited to that path. These results confirm the observations from [23] about the undesired effects of adaptivity.

3) Output-generated hotspots at root links: The objective of this test is to study the effects of multiple hotspots with different severities on the routing algorithms. Multiple hotspots can appear in datacenters running different applications on different priority levels which might employ different routing strategies. We simulate this scenario by reducing the service rate of some links situated at the root level in the fat-tree. At a higher level, this is equivalent to having multiple paths between a source and a destination, each path having a different available bandwidth. As depicted in FIGURE 9(f), there are four paths between the source and the destination; on each path there is a congestion point. The available bandwidths on each path are 25%, 10%, 10% and 50% of the maximal link bandwidth, respectively.

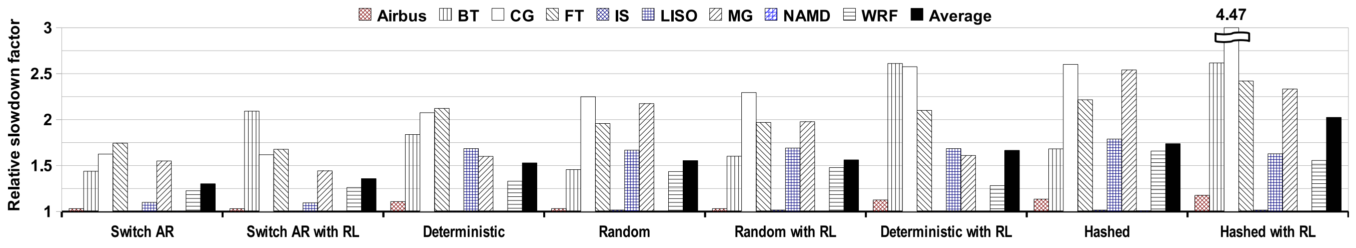
The source injects packets at 90% of the link capacity, and FIGURE 9(a,b,c) shows the throughput at the destination. Deterministic routing, FIGURE 9(c), achieves 25% of the link throughput (312MB/s) because the path selected happens to be the $Path_1$. Random routing, FIGURE 9(a), achieves 40% throughput (500MB/s) owing to head-of-line blocking. Initially the packets are uniformly distributed between the four flows. The severely congested paths slow down the mildly congested ones, and the total throughput stabilizes at approx. 40%. This is confirmed by inspecting the queue lengths at the congestion points in FIGURE 9(d). The queues for $Path_2$ and $Path_3$ are the most congested ones, whereas the other two queues are almost empty. The activation of rate limiters induces no significant changes.

Switch-adaptive routing oscillates between the four paths. This is visible by inspecting the throughput graph in FIGURE 9(b). The oscillation causes a large number of packets to be received out of order. When switch adaptive routing selects a path, the queue associated with the hotspot fills. The switch starts to generate congestion notifications and the upstream switches reroute the traffic on a different path. However, there are still packets that are in the queue of the previous path, which has to drain. This out-of-ordering generates the throughput oscillations.

Switch AR reaches ca. 60% throughput (750MB/s) on average. The 60% throughput is obtained because immediately



(a) Without high-priority traffic. Random routing produces the shortest execution times. Switch adaptive routing looses throughput because of frequent oscillations. Deterministic and hashed routing lack adaptivity. Hashed is strongly affected by the large messages generated by applications such as CG.



(b) With high-priority traffic. Switch AR has the shortest execution time. Adaptive routing can detect the hotspot and avoid it, while random routing continues to send packets through the congested link. Deterministic routing does not use the congested link for half of the connections, on average. Hence it outperforms the random scheme.

Figure 10. Relative slowdowns of HPC workloads (smaller is better). The reference is the runtime needed to execute the trace on a single-hop ideal crossbar. Results are sorted by their average slowdown. In (a) the HPC applications can use the full bandwidth of each link. In (b) we simulate the impact of high-priority traffic by reducing the bandwidth of a single link to 33% (by 66%).

after a flow is rerouted to a new path the buffers along the previously used path are still draining. Hence for a limited time period, multiple paths are in use. Since in FIGURE 9(b) path changes are frequent the average throughput achieved by Switch AR is higher than when using only the least congested path, i.e. 50%. Oscillations here actually increase the throughput, although introduce jitter and misordering.

However, we observe that the queues utilization is quite equal (FIGURE 9(e)). Switch AR manages to partly discover the severity of each hotspot, albeit it is penalized by oscillations.

Subject of ongoing research, Switch AR's instability could be improved by adopting fractional split ratios, instead of switching the full load from a path to another.

B. HPC Workloads

In this subsection we evaluate the routing schemes using the HPC application traces presented in SECTION IV-C. The results of the evaluation are plotted in FIGURE 10(a) and FIGURE 10(b). To facilitate the comparison, we display the relative slowdowns of each routing scheme. The reference is the runtime needed to execute the trace on a single-hop ideal crossbar network. We simulate a datacenter with 128/256 processing nodes connected by a 2-ary k-tree network.

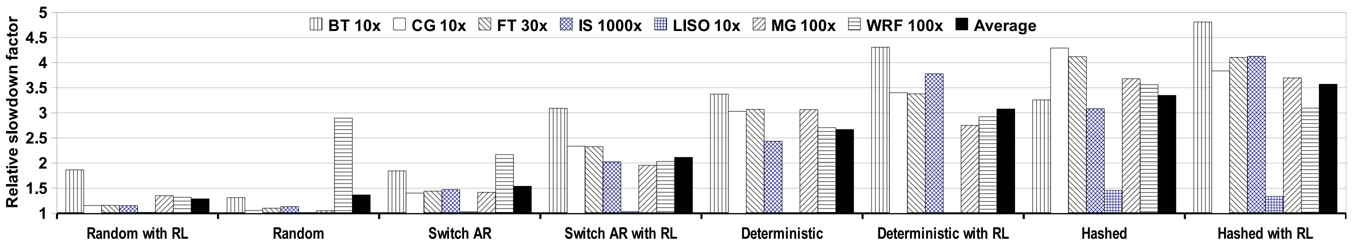
We considered two scenarios. In the first one, we run the application on an empty network, with no other traffic present. The application can use the entire bandwidth of each link.

In the second scenario, we apply a reduction of the bandwidth on a single link. We believe that the second scenario is more realistic. The motivation of this approach is to test how the routing algorithm works for low-priority traffic, assuming Priority Flow Control and Enhanced Transmission Selection are implemented in the real datacenters. The high-priority traffic has a guaranteed fraction of bandwidth allocated. The low priorities can freely use this fraction only when there is no high-priority traffic. On the other hand, when there is high-priority traffic, the low-priority traffic will act as if the links have lower and variable capacities, as modulated by the high priorities active above (see FIGURE 4 for an example).

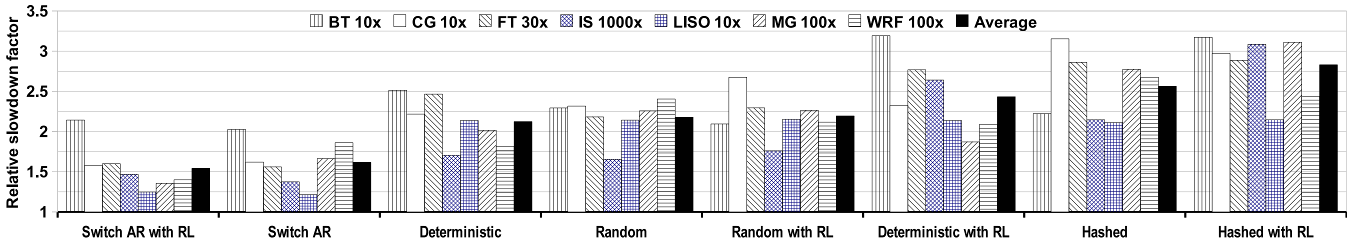
In all simulations, we employed random task placements. In a virtualized datacenter it is realistic to assume that the tasks are usually assigned to processing nodes in an arbitrary way, depending on various parameters such as node load or task priority.

From FIGURE 10(a) we can see that the oblivious random routing algorithm delivers the shortest average execution time. On the other hand, when we introduce an imbalance in the link bandwidth caused by the execution of a high-priority application, the situation changes and the Switch AR provides the shortest execution time (FIGURE 10(b)). The load-oblivious solutions always balance the load evenly between the available paths. When the slightest imbalance appears, they suffer a loss of throughput due to head-of-line blocking: the busy links will slow down the free links.

In FIGURE 10(b) even the deterministic algorithms can sometimes perform better than the oblivious routing. This is also confirmed by the results in [24], which showed that random routing in some cases performs worse than deterministic schemes. The intuitive cause of this behavior is that deterministic schemes can be ”lucky“ for some flows and completely avoid the hotspot, whereas random routing, which



(a) Without high-priority traffic.



(b) With high-priority traffic.

Figure 11. Relative slowdowns of scaled HPC workloads (smaller is better). The reference is the runtime needed to execute the trace on a single-hop ideal crossbar. Results are sorted by their average slowdown. In (a) the HPC applications can use the full bandwidth of each link, while in (b) we simulate the impact of high-priority traffic by reducing the bandwidth of a single link to 33% (by 66%). Same observations as for FIGURE 10 apply. The rate-limited versions or random routing and Switch AR, respectively, produce the shortest execution times because they avoid the end-point congestion generated by scaling.

uses all paths at the same time, is forced to always pass through the hotspot.

The deterministic routing provides long execution times because of the lack of adaptivity and load-balancing. This is particularly visible when running applications that generate long messages such as the CG benchmark that produces 750KB messages. A single conflict between two long messages and the execution time can be substantially increased. From the same reason hashed routing also slows down the application.

C. Scaled HPC Workloads

For some of the applications listed in SECTION IV-C, we observed that they do not put pressure on the communication network. For others such as IS or LISO, the differences observed between the routing schemes were minor. This was due to the fact the applications rarely exchanged small messages. For such applications, contention was infrequent, hence the adaptive routing algorithms could not provide any benefit. This is confirmed by previous work. In [23] the authors showed that the adaptivity does not provide any improvement for some categories of workloads.

To stress the communication network more and to emphasize the differences between the various routing schemes, we scaled the trace files. When a trace is scaled the size of all the messages generated by its execution is multiplied with a given scale factor. We used different scale factors to ensure that the communication demands of the application are high enough to generate contention. TABLE II lists the scale factors used and the average and maximum message size the scaled application uses. For most of the applications the maximum message size is on the order of MB.

Table II
TRACE SCALING FACTORS

Trace	Scale factor	Mean size	Max size
BT	10x	69.4 KB	1.23 MB
CG	10x	1.61 MB	7.5 MB
FT	30x	1.31 MB	3.9 MB
IS	1000x	158 KB	2.06 MB
LISO	10x	40 KB	121 KB
MG	100x	2.96 MB	13.5 MB
WRF	100x	0.93 MB	9.7 MB

The results are in FIGURE 11(a) and FIGURE 11(b). Scaling of traces does not radically changes the ranking of the routing schemes, but will accentuate the differences between them. For empty network the random routing is still the best followed by the Switch AR, deterministic and hashed routing. As in the previous section, for a network where high-priority traffic is present, the Switch AR provides the shortest execution time followed by the random, deterministic and hashed routing.

We can notice that the rate-limited versions of random routing and Switch AR perform better than the versions without RLs. As a side effect of trace scaling end-point congestion is generated in workloads like MG or WRF. The adverse effects of this end-point congestion are eliminated by the use of RLs as explained in SECTION V-A2.

VI. RELATED WORK

For an extensive overview of routing mechanisms, the reader is referred to CHAPTERS 8 to 11 of [17] and CHAPTER 4 of [18].

Deterministic routing was analyzed in [19], [20], [21] and [22], [23]. Random routing was studied in [25], [26], [27]. In [33], an algorithm that attempts to randomize the set of messages chosen for delivery is presented. Hashed routing is presented in [28] as a special case of Equal-Cost Multi-Path routing described in [34]. Switch adaptive routing was introduced in [1], [2].

In [15] an overview of the proposed schemes for CEE congestion management is given. An alternative delay-based congestion management scheme is introduced in [14] and proved to be efficient.

In [35], a connection scheduling algorithm for fat-trees is introduced. It needs a central controller for all switches situated on the same level, which makes it impractical for large datacenters. A better idea comes from [36], where the authors introduce a pattern-aware routing scheme. Their solution applies to HPC environments where the workloads are known in advance and an offline optimization of the routing can be done before the application is launched.

VII. Conclusions

We have presented the main features of a CEE-based datacenter network and have performed a comparative analysis of the operation of the current state of the art deterministic, load-oblivious and adaptive routing schemes, including the recently proposed Switch AR. To quantitatively evaluate their benefits and drawbacks, we have simulated them in detail using synthetic and trace-driven traffic.

The output-generated hotspot scenarios, as well as the HPC workloads, have revealed a drawback of the deterministic schemes, namely the lack of adaptivity to the load conditions. Meanwhile the input-generated hotspot scenarios, and the scaled-up traces, have confirmed the benefits of QCN-like rate limiters when applied to Switch AR, as well as to the other schemes, in preventing saturation trees.

Next, output-generated hotspot scenarios, when combined with HPC traces – with multiple priority traffic – have proved that the Switch AR algorithm achieves better performance on asymmetrically loaded networks. When the loads, however, are symmetrically distributed on all the available paths, the random routing has outperformed the other schemes.

Albeit Switch AR performs well, neither of its two current versions wins across all the benchmarks. Hence, our two guiding questions from Section I have generated a rather nuanced answer, without revealing yet a clear single winner or loser amongst the schemes investigated.

Besides cost, ultimately the decision depends on the traffic patterns expected to prevail in the datacenter. For uniform traffic distributions, the $D\text{-}mod\text{-}k$ deterministic remains the simplest and best performing routing. Its in-order delivery, without resequencing buffers, further lowers the implementation costs. Next, for many Cloud and HPC applications, random and hashing-based schemes deliver a sensible trade-off between load-balancing, latency, throughput, and cost – which may justify their popularity.

Finally, for complex federated datacenters simultaneously using multiple virtual lanes/priorities, with high burstiness and margins for the unknown future applications, the adaptive method outperforms the other schemes. Its higher cost is partially alleviated by the increased deployment of QCN-compliant switches, which readily provide congestion status information for a relatively low cost, expressed in congestion notification traffic overhead. This somewhat unexpected fringe benefit of the 802.1Qau congestion management represents an additional payoff for the efforts invested by academia and industry in standardizing QCN.

As future work, we plan to stabilize the oscillations inherent to the switch-based adaptive scheme, which we primarily attribute to the binary route split ratios. Additionally, we are exploring the merits of source-based adaptive routing, as the counter-part of Switch AR. Finally, we plan to gather more benchmarks and datacenter network patterns and use them for a more comprehensive study.

Acknowledgments

We are deeply indebted to Casimer DeCusatis, Ronald Luijten, Fredy Neeser, Renato Recio, German Rodriguez, Alessandra Scicchitano, and, especially to the 802.1Qau workgroup, for their contributions.

References

[1] C. Minkenberg, M. Gusat, and G. Rodriguez, "Adaptive Routing in Data Center Bridges," in *Proc. 17th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2009)*, New York, NY, USA, August 2009.

[2] C. Minkenberg, A. Scicchitano, and M. Gusat, "Adaptive routing for Convergence Enhanced Ethernet," in *Proc. 2009 International Workshop on High-Performance Switching and Routing (HPSR 2009)*, Paris, France, June 2009.

[3] C. Minkenberg and G. Rodriguez, "Trace-driven Co-simulation of High-Performance Computing Systems using OMNeT++," in *Proc. SIMUTools 2nd International Workshop on OMNeT++*, Rome, Italy, March 2009.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. ACM SIGCOMM 2008 Conference on Data Communication*, Seattle, WA, USA, August 2008.

[5] R. N. Mysore, A. Pamboris *et al.*, "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona, Spain, August 2009.

[6] S. R. Öhring, M. Ibel, S. K. Das, and M. Kumar, "On Generalized Fat Trees," in *Proc. 9th International Parallel Processing Symposium (IPPS 1995)*, Santa Barbara, CA, USA, April 1995.

[7] F. Petrini and M. Vanneschi, "k-ary n-trees: High Performance Networks for Massively Parallel Architectures," in *Proc. 11th International Parallel Processing Symposium (IPPS 1997)*, Geneva, Switzerland, April 1997.

[8] ——, "A Comparison of Wormhole-Routed Interconnection Networks," in *Proc. Third International Conference on Computer Science and Informatics*, Research Triangle Park, NC, USA, March 1997.

[9] *P802.1Qbb/D1.3 Virtual Bridged Local Area Networks - Amendment: Priority-based Flow Control*, IEEE Draft Standard, 2010. [Online]. Available: http://www.ieee802.org/1/pages/802.1bb.html

[10] *P802.1Qau/D2.4 Virtual Bridged Local Area Networks - Amendment: Congestion Notification*, IEEE Draft Standard, 2009. [Online]. Available: http://www.ieee802.org/1/pages/802.1au.html

[11] *P802.1Qaz/D1.2 Virtual Bridged Local Area Networks - Amendment: Enhanced Transmission Selection for Bandwidth Sharing Between Traffic Classes*, IEEE Draft Standard, 2010. [Online]. Available: http://www.ieee802.org/1/pages/802.1az.html

[12] G. Pfister and V. Kumar, "The Onset of Hotspot Contention," in *Proc. International Conference in Parallel Processing (ICPP 86)*, University Park, PA, USA, August 1986.

[13] G. Pfister and V. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 943–948, October 1985.

[14] M. Gusat, R. Birke, and C. Minkenberg, “Delay-based Cloud Congestion Control,” in *Proc. IEEE GLOBECOM 2009*, Honolulu, HI, USA, December 2009.

[15] C. Minkenberg and M. Gusat, “Congestion Management for 10G Ethernet,” in *Proc. Second Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip (INA-OCMC 2008)*, Goteborg, Sweden, January 2008.

[16] L. Xu, K. Harfoush, and I. Rhee, “Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks,” in *Proc. 23rd Conference of the IEEE Communications Society (Infocom 2004)*, Hong Kong, China, March 2004.

[17] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[18] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks. An Engineering Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[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 International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, NM, USA, April 2004.

[20] G. Johnson, D. J. Kerbyson, and M. Lang, “Optimization of InfiniBand for Scientific Applications,” in *Proc. 22nd International Parallel and Distributed Processing Symposium*, Miami, FL, USA, April 2008.

[21] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, “Optimized InfiniBand Fat-tree Routing for Shift All-to-All Communication Patterns,” in *Proc. International Supercomputing Conference (ISC 2007)*, Dresden, Germany, November 2007.

[22] F. Gilabert, M. E. Gómez, P. López, and J. Duato, “On the Influence of the Selection Function on the Performance of Fat-Trees,” in *Proc. 12th International Euro-Par Conference*, Dresden, Germany, August 2006.

[23] C. Gómez, F. Gilabert, M. E. Gómez, P. López, and J. Duato, “Deterministic versus Adaptive Routing in Fat-Trees,” in *Proc. 21th International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, March 2007.

[24] G. Rodriguez, C. Minkenberg *et al.*, “Oblivious Routing Schemes in Extended Generalized Fat Tree Networks,” in *Proc. 2009 Workshop on High Performance Interconnects for Distributed Computing (HPI-DC 2009)*, New Orleans, LA, USA, August 2009.

[25] L. G. Valiant and G. J. Brebner, “Universal Schemes for Parallel Communication,” in *Proc. 13th annual ACM Symposium on Theory of Computing*, Milwaukee, WI, USA, May 1981.

[26] J. Flich, M. P. Malumbres, P. López, and J. Duato, “Improving Routing Performance in Myrinet Networks,” in *Proc. 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, May 2000.

[27] C. Leiserson, Z. S. Abuhamdeh *et al.*, “The Network Architecture of the Connection Machine CM-5,” in *Proc. The 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, USA, June 1992.

[28] C. Hopps, “Analysis of an Equal-Cost Multi-Path Algorithm,” Internet Engineering Task Force, RFC 2992, November 2000.

[29] P. Geoffray and T. Hoefler, “Adaptive Routing Strategies for Modern High Performance Networks,” in *Proc. 16th IEEE Symposium on High Performance Interconnects (HOTI 2008)*, Stanford, CA, USA, August 2008.

[30] A. Varga, “The OMNeT++ Discrete Event Simulation System,” in *Proc. European Simulation Multiconference (ESM 01)*, Prague, Czech Republic, June 2001.

[31] W. E. Denzel, J. Li, P. Walker, and Y. Jin, “A Framework for End-to-end Simulation of High performance Computing Systems,” in *Proc. First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008)*, Marseille, France, March 2008.

[32] D. Bailey, E. Barszcz *et al.*, “The NAS Parallel Benchmarks,” NASA Ames Research Center, Moffett Field, CA, NASA Technical Report RNR-94-007, March 1994.

[33] R. I. Greenberg and C. E. Leiserson, “Randomized Routing on Fat-Trees,” in *Proc. 26th Annual Symposium on the Foundations of Computer Science*, Portland, OR, USA, October 1985.

[34] D. Thaler and C. Hopps, “Multipath Issues in Unicast and Multicast Next-Hop Selection,” Internet Engineering Task Force, RFC 2991, November 2000.

[35] Z. Ding, R. R. Hoare, A. K. Jones, and R. Melhem, “Level-wise Scheduling Algorithm for Fat Tree Interconnection Networks,” in *Proc. 2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL, USA, November 2006.

[36] G. Rodriguez, R. Beivide, C. Minkenberg, J. Labarta, and M. Valero, “Exploring Pattern-aware Routing in Generalized Fat Tree Networks,” in *Proc. 23rd International Conference on Supercomputing (ICS 2009)*, Yorktown Heights, NY, USA, June 2009.