Distributed Adaptive Routing Convergence to Non-Blocking DCN Routing Assignments

Eitan Zahavi, Isaac Keslassy, and Avinoam Kolodny

***Abstract*—With the growing popularity of big-data applications, Data Center Networks (DCN) increasingly carry larger and longer traffic flows. As a result of this increased flow granularity, static routing cannot efficiently load-balance traffic, resulting in an increased network contention and a reduced throughput. Unfortunately, while adaptive routing can solve this load-balancing problem, DCN designers refrain from using it, because it also creates out-of-order packet delivery that can significantly degrade the reliable transport performance of the longer flows.**

In this paper, we show that by throttling each flow bandwidth to half of the network link capacity, a distributed adaptive routing algorithm is able to converge to a non-blocking routing assignment within a few iterations, causing minimal out-of-order packet delivery. We present a Markov chain model for distributed adaptive routing in the context of Clos networks that provides an approximation for the expected convergence time. This model predicts that for full-link-bandwidth traffic, the convergence time is exponential with the network size, so out-of-order packet delivery is unavoidable for long messages. However, with half-rate traffic, the algorithm converges within a few iterations and exhibits weak dependency on the network size. Therefore, we show that distributed adaptive routing may be used to provide scalable and non-blocking routing even for long flows in a rearrangeably-non-blocking Clos network under half-rate conditions. The proposed model is evaluated and approximately fits the abstract system simulation model. Hardware implementation guidelines are provided and evaluated using a detailed flit-level InfiniBand simulation model. These results, providing fast convergence to non-blocking routing assignment, directly apply to adaptive-routing systems designed and deployed in various DCNs.

***Index Terms*—Data Center Networks, Big Data, Adaptive Routing**

I. Introduction

A. Background

NEARLY all currently-deployed state-of-the-art Data Center Networks (DCNs) rely on layer-3 Equal Cost Multipath (ECMP) routing to evenly distribute traffic and utilize the aggregated capacity provided by the multi-tier network [1]. ECMP routing is *deterministic and static*, because it is based on constant hash functions of the flow identifier. The obtained bandwidth from these techniques is close to the network cross-bisectional bandwidth as long as flow granularity is small, i.e. the routing algorithm spreads many flows that are either short or of low-bandwidth.

E. Zahavi is with Mellanox Technologies LTD and the Department of Electrical Engineering, Technion, Israel (e-mail: ezahavi@tx.technion.ac.il).

I. Keslassy and A. Kolodny are with the Department of Electrical Engineering, Technion, Israel (e-mails: {isaac, kolodny}@ee.technion.ac.il).

Digital Object Identifier 10.1109/JSAC.2014.1401xx.

In recent years a new challenge has emerged for DCNs: support *"big-data" applications like MapReduce* [2] [3]. In measurements conducted on the Shuffle and Data-Spreading stages of MapReduce applications, it was shown that up to 50% of the run time may be consumed by these stages [4]. In fact, these stages transmit the intermediate computation results with sizes up to 10's of gigabytes between each pair of servers participating in the computation. Therefore, the long and high-bandwidth flows characterizing these phases break the nice traffic spreading provided by the ECMP hash function for many low-bandwidth flows. Under such hash-based traffic spreading, the probability of over-subscription follows the balls-and-bins max-load distribution [5]. The contending flows result in low effective bandwidth [6].

Adaptive routing can provide a solution to this contention problem [7] [8]. In fact, adaptive routing, which changes the routing assignments based on the current load, can reach efficient traffic spreading in the DCN, even when flow granularity is high. In particular, *distributed* adaptive routing systems, in contrast to centrally managed ones, rely on some distributed feedback mechanism which translates network congestion into hints for the involved switches to modify the routing assignments. Based on these hints, the traffic is adaptively steered towards less congested parts of the network and thus throughput is improved. Nevertheless, when using adaptive routing, switches need to know little about the global state of the network or about the states of other switches, and therefore be implemented in DCNs for big-data applications. This property of adaptive routing is key for its scalability for large systems, where global state is large and hard to be up-to-date in all switches.

Unfortunately, adaptive routing also causes high out-of-order packet delivery in long flows, produced by the modification of the forwarding path of different packets of the same message [9]. This out-of-order packet delivery greatly degrades window-based transport protocols like TCP (or any other packet window-based transport protocol) and can result in a significant degradation of throughput and latency [10] [11]. Some studies propose the use of re-order input-buffers and limit the number of in-flight packets to control their cost [7] [10]. However, limiting the number of in-flight packets degrades bandwidth and increases latency.

Another challenge for adaptive routing is that previous network states cannot be used for deciding about the new routing when the entire traffic pattern changes synchronously. Unfortunately, this is the exact behavior of BSP model programs as well as for the shuffle stages of a MapReduce. So for adaptive routing to be effective it needs to react before the traffic pattern changes. Due to these limitations, adaptive routing is often considered irrelevant for DCNs running big-data applications.

This paper presents the conditions under which a distributed adaptive routing system can cause minimal out-of-order packet delivery for big-data applications, while achieving high throughput.

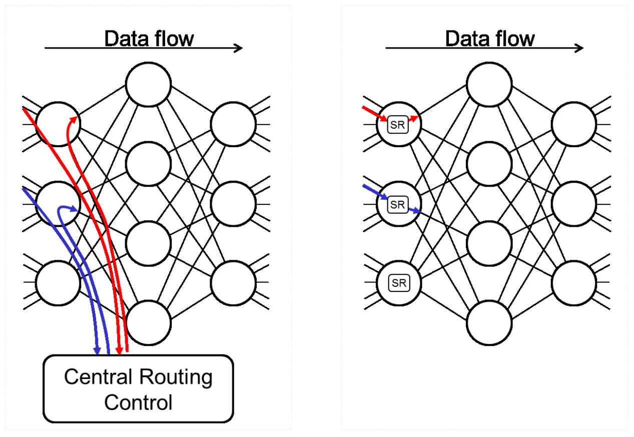
B. Related Work

The network contention caused by a relatively small number of high-bandwidth flows is also a long outstanding problem of static routing in High Performance Computing (HPC) clusters. Scientific applications running on these clusters resemble big-data applications, because most parallel scientific applications are coded according to the BSP model [12], where computation and communication are separated into non-overlapping phases [13]. Under such a model, network contention directly impacts the overall program runtime since the slowest flow dictates the length of the entire communication phase [14] [15]. For these reasons, efforts have been made to provide adaptive-routing, together with heuristics and mechanisms to improve both throughput and latency [16] [17]. Most commercial interconnection networks like Cray BlackWidow [18], IBM BlueGene [19] and the InfiniBand-based Mellanox InfiniScale switch devices [20] provide adaptive routing. The most scalable systems are designed such that each switch knows little or nothing about the traffic or queues of the other elements in the network. Such systems are thus denoted *oblivious adaptive routing*. Mechanisms were also proposed to enhance the adaptive-routing hardware in switches by relying on a complete or partial view of the entire network state [21] [8]. Nevertheless, to maintain scalability, even when complex feedback mechanisms are suggested, the self-routing principle, where each switch makes its own independent decisions, is maintained in most published work.

Adaptive-routing stability was studied in the context of the Internet [22] [23]. In these studies, a centralized adaptive-routing algorithm is employed to optimize the network performance for some figure of merit. The computed routing slowly changes when compared to the traffic message times. The stability of such systems is then defined as the ability to avoid fluctuations in routing assignments when a computed routing is applied. Thus, adaptive-routing stability is different from our definition of adaptive-routing convergence, i.e. the ability of the system to reach a non-blocking routing assignment for any given traffic permutation.

Cell switching techniques are proposed [24] as means to provide load balancing and avoid network congestion. However, as they rely on packet splitting and re-assembly, they are thought of as having higher latency and to require larger buffers on the network edge. For these reasons this paper focuses on packet switching solutions.

A very common DCN topology is the fat-tree, which resembles a folded-Clos network, with the exception that not all routes have to go through the roots of the network. Routing in Clos networks was mostly studied in the context of systems where a centralized control unit allocates virtual circuits to injected flows [25] (see Figure 1(a)). Consequently, previous



(a) (b)

Fig. 1. Centralized Routing versus Self Routing: (a) In a centrally controlled Clos, input switches request an output-port assignment for each arriving flow from the central controller. (b) In a distributed adaptive routing system, a Self Routing (SR) unit within each input switch provides that decision in an autonomous manner.

studies focused on the topological network properties that allow for a central algorithm to provide non-blocking routing assignment for a set of source-destination pairs were defined. These properties define strict sense non-blocking (SNB) as the case where new pairs can be introduced without impact on previously routed flows. Similarly, the conditions for rearangeable non-blocking (RNB) [26] were defined for the case where introducing new pairs may require some changes to previous routing. The central controller ability to rearrange existing flows in order to fit some more flows into the network is similar to the distributed adaptive routing concept of routing reassignment in case of network contention. When packet switching was proposed as an alternative to virtual circuits, research in Clos networks focused on the properties obtained for multi-rate traffic injected into a network of multiple-capacity links [27] [28]. In Clos networks, the term n-rate represents the number of different ratios of the traffic bandwidth to the system link capacity. In our work, we also use the idea of setting the bandwidth of flows as a fraction of the link capacity.

Unfortunately, the centralized controller approach used for Clos routing does not scale with the cluster size, and therefore can hardly apply to our DCNs: To estimate the time available for the central controller for handling a single flow, consider a DCN of 10K nodes each running 10 VMs. An optimistically long flow length of 64KB on 40Gbps link provides 12.8μsec flow lifetime. Further, if we assume communication is only 10% of application runtime, the flows arrival rate is 1/128μsec on average. Under the above conditions the central routing unit has to handle a request rate of ~10^9[req/sec] which allows roughly 2 operations per request on a 2GHz CPU. Even a parallel routing algorithm will have to use more than a single operation for handling a request.

For this reason, a distributed adaptive routing approach was also proposed in Clos networks, and denoted as “self-routing” [29]. In this approach, each switch can make its own routing decisions such that no central control unit is required (see Figure 1(b)). The self-routing study [29] was mainly focused on reducing the non-scalable overhead of the central routing controller. A probability analysis conducted by [30] on some specific self-routing Clos systems also provides an upper-bound on the number of contending flows (with high probability), and thus provides an upper-bound for the expected network queue length and service time. Our work is different as it shows that under some conditions, adaptive-routing can actually converge into a non-blocking routing assignment where no queues build up.

C. Contributions

To the best of our knowledge, *no work in the literature examines the conditions under which adaptive routing converges to a non-blocking forwarding assignment* and in case of convergence, *we also know of no result on its convergence speed.*

By analyzing convergence time, we are able to show that under some traffic conditions the adaptive-routing can converge to a non-blocking routing assignment within a very short time. After this convergence time, there is no out-of-order delivery and no network contention. Indeed, there will be some performance degradation due to re-transmission of the first few packets of the message. However, even for 256KB messages, re-transmission would introduce very small bandwidth degradation for the entire flow.

Even though our analysis is focused on permutation traffic patterns, other traffic patterns that exhibit end-point congestion (allow for more than one flow to be concurrently destined to a single destination) may also benefit from fast distributed-adaptive-routing convergence, once congestion control mechanisms are incorporated to mitigate the excessive bandwidth sent to that destination.

To reach the above conclusion we have developed approximate Markov process models for Clos self-adaptive-routing systems. The importance of these approximated models is the insight they provide about the reason of slow convergence. The models show that for full-link-bandwidth flows, adaptation due to congestion on one link will likely create congestion on another. Thus, it is unlikely that the system converges to a non-blocking steady state. However, for half-link-bandwidth flows, the probability for one routing adaptation to create congestion on some other link is much lower and thus such system converges in few iterations. These models are then compared to a simulation of a simple distributed adaptive routing system. We further define a set of features that are practical to implement and provide converging oblivious-adaptive-routing system. The proposed hardware is then evaluated by simulation. We claim the following contributions:

- We present an approximate Markov chain model for a three-level Clos network to evaluate the convergence rate of the adaptive-routing process.
- We study the convergence time for the case where the bandwidth of each flow equals the link capacity. The convergence time under such conditions for rearrangeably-non-blocking Clos networks is more than exponential with the number of input switches, so it typically does not converge within any practical message size.
- Conversely, for the case where each flow bandwidth equals half the link capacity, the model shows fast convergence with weak dependency on the network size. Under these conditions, adaptive routing causes very little out-of-order packet delivery.
- We propose a set of implementable system features that provide an oblivious-adaptive-routing. A detailed simulation model of InfiniBand hardware, enhanced with these mechanisms, confirms the above results.

The rest of the paper is organized as follows: Section II first provides a description of an oblivious-self-routing system. Then, Section III analyzes that system using a Markov chain model for predicting the convergence time. Next, Section IV discusses implementation guidelines for adaptive routing system, and Section V provides an evaluation of both the model and the proposed implementation. A discussion and conclusions are finally presented in Section VI.

II. A Distributed Adaptive-Routing System Model

In this section, we introduce and define our architecture model and adaptive routing algorithm, before analyzing their performance in the next section. Consider a *1-rate* and uniform symmetrical Clos network. Assume that it has r input (and output) switches, of $n \times m$ ports each, and denote it as CLOS(n, r, m). Further assume that all links have an equal capacity, and that all flows have an equal bandwidth demand, such that this bandwidth equals $1/p$ of the link capacity. For instance, $p = 2$ means that each flow bandwidth requires half of the link capacity.

Assume that the network carries a full-permutation traffic pattern, i.e. each source sends a continuous flow of data to a single destination, and each destination receives data from a single source. For some traffic patterns and routing assignments there are at most p flows going through any network link (this can be guaranteed for some specific topologies [31]). However, when more than p flows are routed through a link, we declare these flows as *bad flows*, and that link as a *bad link*. We consider the routing as a *good routing* if there are no bad links in the system.

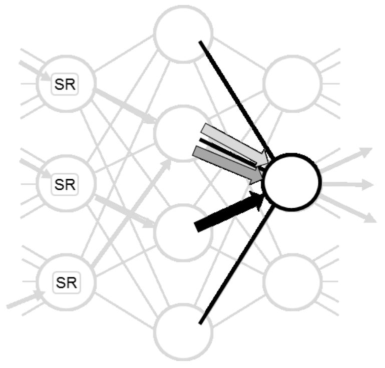
We now want to define the adaptive-routing algorithm. There are many different adaptive-routing systems defined in studies and implemented by hardware, as described in the related work section. Most of these systems are *hard to model* mathematically. Some use complex criteria for selecting output ports [17], some use state history, and some even rely on the distribution of the global network state [8]. Since we seek to learn about the conditions under which convergence is fast enough to support big-data applications, we want to define an adaptive-routing system that is simple enough to be modeled.

Assume that the adaptive-routing system behaves as follows: At $t = 0$, a new full-permutation traffic pattern is applied at the input switches. Each input switch assigns an output port to each of its flows (on Clos and folded Clos topologies this output port defines the complete route to the destination). The output port assignment performed by the input switches is semi-random as a reasonable approach for spreading their traffic with no global knowledge about the flows in other switches. The assignment is termed semi-random since, as input switches do know their own flows, they never assign more than p flows to any of their outputs. This means that bad links are only possible between the middle and output switches where flows from multiple input switches may congest.

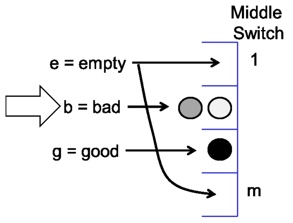
Once the initial routing is defined, the system iterates synchronously through the following phases. Each iteration takes exactly one time slot. First, in the middle of each slot j, i.e. at time $t = j+0.5$, each output switch selects a random bad flow that belongs to its input link with the largest number of flows. It then sends to the input switch at the origin of this bad flow a request to change its routing. The notification process happens before the end of the iteration period $t = j + 1$.

Then, at the start of each time slot $t = j + 1$, when an input switch receives a bad-flow notification, it moves that flow to a new randomly-selected output port. If that new port is already full, the input switch swaps the moved flow with another flow on that output port to avoid congestion. As a result, the swapped flow may cause a new oversubscription on some middle-switch-to-output-switch link. The system keeps adapting routes and iterating through these two steps until no more bad links exist.

In the above model the middle switches do not perform any adaptive routing. All input switches are active at the beginning of each iteration period, and all output switches at the middle of each iteration period. Packets continuously flow through the network during the routing adaptation in order to provide the output switches with the information about the flows routed through their links.



(a)



(b)

Fig. 2. A balls-and-bins representation: (a) A single output switch of the Clos network with 2 flows contending on its second link from the top. (b) A balls-and-bins representation of the m output-switch input links as bins and the n flows as balls. For the case where the bandwidth of each flow equals the link bandwidth ($p = 1$), the state variable e represents the total number of empty bins, b the number of bad bins and g the number of good bins.

III. Analysis

This section presents Markov chain models for the convergence time of the system presented in Section II. Even for that simple system, an accurate model is hard to provide since the system state should represent all the flows on all the links. Since the size of the Markov model grows exponentially with the number of flows and the number of links, we must provide an approximation instead. The *first model* below takes the unrealistic but simplifying assumption that each output switch may be treated as an *independent system*. Due to the interdependency of the output-switch convergence times, as imposed by the input switches, this model is only useful to describe the convergence process of a single output switch.

Then, to better predict the convergence time, we present *two other models*, for full- and half-bandwidth flows. These models track the dependency between the output switches, and focus on the last stages of convergence when that interdependency has its greatest impact. Finally, in the evaluation section, we will use a simulation program that mimics the analyzed system behavior to evaluate these approximations.

A. Balls-and-Bins Model

As shown in Figure 2, we propose a *balls-and-bins model* to represent all the links that feed into the same output switch: each input link is considered as a bin, and each flow as a ball. We start with a random spreading of the n balls into the m bins, and want to obtain the expected time t at which there are at most p balls in each bin. Inspecting the changes to flows routed through the links feeding into a specific output switch, there are two processes that happen concurrently: (a) an *improvement process* and (b) an *induced-move process*.

An *improvement process* describes the sequence of events starting by an output switch request to move one of its worst-link bad flows and ending by that bad flow move to a new input link of that switch. In the balls-and-bins representation of the output switch this would cause the movement of one of the worst bad-bin balls into some new bin (not necessarily an empty one).

With a slight but significant difference, the *induced-move process* describe the sequence of events where an arbitrary flow on some output switch O_i moves due to a bad-flow adaptation request sent by another output switch O_j. Such a sequence happens when the input switch receiving the request swaps the two flows. In the balls-and-bins representation of the output switch O_i this would cause the movement of a good or bad ball into some new bin. Such a move is *induced* by a change of state of another balls-and-bins system.

Figure 3 provides an example for how the improvement in one output switch causes an induced move on the other. The distribution of induced moves on the different output switches follows the random throw of k balls into r bins, where k is the number of output switches that have not reached their steady state.

B. A Single Output-Switch Markov Chain Model for Full-Link Bandwidth per Flow

This sub-section presents an approximate Markov chain model for a single output switch in the case of full-link bandwidth per flow (i.e., $p = 1$). Based on the system symmetry, this model considers each output switch independently.

To model the interactions between output switches, the model assumes that each time an output switch kicks some

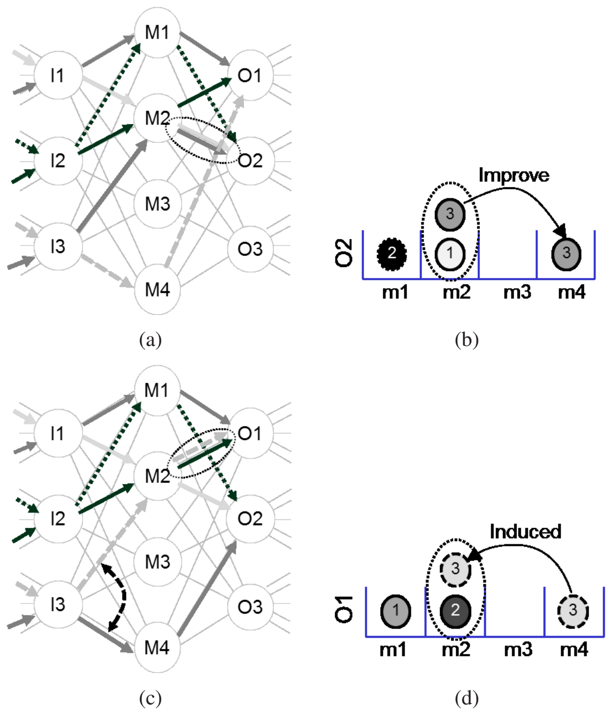


Fig. 3. Improvement and Induced Moves: (a) An example system with congestion on the second input from top of switch $O2$. (b) Output switch $O2$ initiates an *Improvement* process by requesting the move of the flow from input-switch $I3$. The input switch $I3$ randomly selects the new output port connected to middle-switch $M4$ for that flow. Since another flow, to $O1$ was routed through the port to $M4$, $I3$ swaps the two flows (to $O2$ and $O1$ through $M2$ and $M4$). This change actually *improves* the state of output switch $O2$. (c) The routing change done by $I3$ has moved the congestion to the second input of output-switch $O1$. (d) The swap of ports in (b) caused an *Induced* ball move on $O1$.

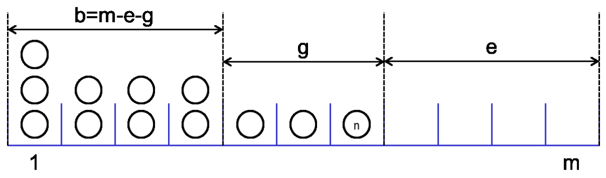


Fig. 4. An approximated model assuming a uniform distribution of bad balls in bad bins: In the shown case, the Improvement process must take one of the balls in the left-most bin, as it is the worst bin. The Induced-Move process may move any ball. In the shown case, if a bad ball is selected by the Induced-Move process from a bad bin with 2 balls (i.e. bin 2, 3 or 4) and then falls on an empty bin, 2 good bins are created and g increases by 2. Else if it is selected from the first bin and then falls on an empty bin, the first bin does not become good and g only increases by 1.

bad ball, an induced move will happen with probability of n/m. As depicted in Figure 4, we define a state variable e that represents the number of empty bins, and another state variable g that counts the number of good bins. So the Markov state for the single output switch can be represented using the pair (e, g). To simplify the analysis, this model makes the following additional approximations. First, induced moves are modeled as evenly distributed over all output switches, such that all the r output-switch systems are identical and can be treated as uncorrelated. Second, in order to avoid the need to have $n-1$ state variables to count the number of bins with 2, 3, ... n balls (which explodes the state space), the model assumes all bad bins are of the same height.

The balls-and-bins is modeled using two concurrent processes that affect the state of the balls:

The improvement process: Each output switch selects one of the balls in one of the bins with the highest number of balls, and randomly places it into a bin. As long as there are any bins with at least 3 balls, it is guaranteed that the number of bad bins cannot decrease by this process (since the selected bin will only lose one ball). The probability for the selected ball to fall on an empty, good or bad bin depends on the number of such bins.

The induced-move process: This process takes a random ball and randomly places it in some bin. The probability for such moves to occur depends on the number of empty links on the input switches. Since there are only n flows spread on m links, and the selection of the flow to be moved is random, the distribution of the moved flows to input switches should follow the distribution of throwing n balls into m bins. In order to avoid the complexity of using this distribution, the model relies on the approximation of the average probability of an induced move, which is n/m.

To predict if the induced move changes the number of good bins by $+1$, -1 or -2, we should have tracked the exact number of bins with 0, 1 and 2 balls. As shown in the example of Figure 4, if the moved ball is from bins with just 2 balls and it falls in an empty bin, g is increased by 2. But if the bad ball is from a bin with more than 2 balls, the number of good bins is only increased by 1. However, to compute the number of bins with 2 balls using a Markov chain, we would need to also track the number of bins with 3 balls; and so forth. If we define these n state variables, *the state space explodes*. As the distribution of the number of balls per bin is expected to be a sharp function, such that the probability drops significantly with the number of balls, we claim that a reasonable approximation is to assume that all the bad bins have the same number of balls. In fact, the output switch policy of re-routing a ball from the bin with the highest number of balls further strengthens this assumption.

Let us consider each state transition induced by the request of some other output switch to improve its state. The ball selected may be a good or bad ball, and we assume by symmetry that it may be moved into any bin with the same probability. The probability for a good ball to be selected is g/n. The probability for a bad ball is the complementary $(n-g)/n$. The possible moves and their respective probabilities are described in Table I. The combined impact of the two processes is obtained by considering each possible pair of the Improve and Induced-move state transitions, and adding their probability product to the Markov state transition matrix.

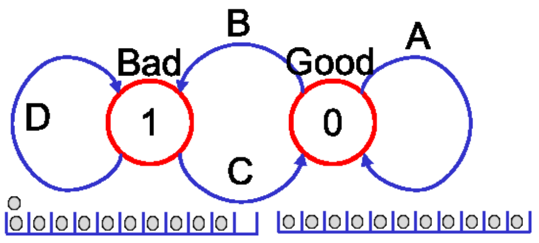
C. *Last-Step Model for Flows of Full Link Bandwidth*

A full model of the entire Markov matrix of all states of all output switches is infeasible due to its size. In order to obtain an approximation for the convergence time, we suggest inspecting the r output switches *just before they reach convergence*. We call this the *Last-Step model*.

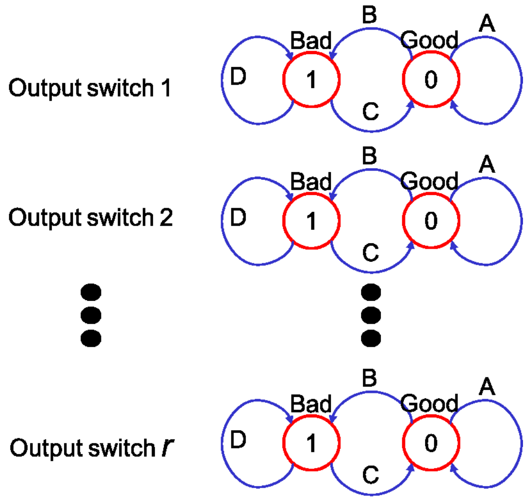
As we suspect that the long convergence times are a result of the induced moves forced by one output switch on another, we focus the model on the last steps of convergence. Only

TABLE I
POSSIBLE STATE CHANGES WITH $p = 1$ AND THEIR PROBABILITY

Process	**b-balls/ b-bins**	**Who moves**	**Move Where**	**New State**	**Probability**
Induced Move	Any case	good ball	empty bin	*e, g*	*g/n*e/m*n/m=ge/m^2*
		good ball	other good bin	*e+1, g-2*	*g/n*(g-1)/m*n/m = g(g-1)/m^2*
		good ball	same good bin	*e, g*	*g/n*1/m*n/m=g/m^2*
		good ball	bad bin	*e+1, g-1*	*g/n*(m-g-e)/m*n/m = g(m-g-e)/m^2*
	> 2.5	bad ball	empty bin	*e-1, g+1*	*(n-g)/n*e/m*n/m=(n-g)e/m^2*
		bad ball	good bin	*e, g-1*	*(n-g)/n*g/m*n/m=(n-g)g/m^2*
		bad ball	bad bin	*e, g*	*(n-g)/n*(m-g-e)/m*n/m=(n-g)(m-g-e)/m^2*
	≤ 2.5	bad ball	empty bin	*e-1, g+2*	*(n-g)/n*e/m*n/m = (n-g)e/m^2*
		bad ball	good bin	*e, g*	*(n-g)/n*g/m*n/m= (n-g)g/m^2*
		bad ball	other bad bin	*e, g+1*	*(n-g)/n*(m-e-g-1)/m*n/m=(n-g)(m-e-g-1)/m^2*
		bad ball	same bad bin	*e, g*	*(n-g)/n*1/m*n/m= (n-g) /m^2*
Improvement	> 2	bad ball	empty bin	*e-1, g+1*	*e/m*
		bad ball	good bin	*e, g-1*	*g/m*
		bad ball	bad bin	*e, g*	*(m-e-g)/m*
	≤ 2	bad ball	empty bin	*e-1, g+2*	*e/m*
		bad ball	good bin	*e, g*	*g/m*
		bad ball	same bad bin	*e, g*	*1/m*
		bad ball	other bad bin	*e, g+1*	*(m-e-g-1)/m*



(a)



(b)

Fig. 5. Approximated Last Step model of a single output-switch: (a) A single bad state implies a lower bound on convergence time as it does not let the bad state degrade further than the last step. (b) The set of r state pairs describes the entire system.

when all the output switches reach together their good state, they stop forcing each other back into bad states. The model only uses a single bad state that is closest to the good state. In that sense, it is an *optimistic* model, as a sequence of bad induced moves is not modeled and the output switch stays close to its good state. Yet, we will later show that it correctly models the *exponential* convergence time of our system.

Figure 5(a) shows the Markov states of a single output switch. There are only two states for an output switch: 0 (good) and 1 (bad). Only a single bad bin is possible one step away from the good state as shown on the balls-and-bins systems drawn below the state graph. The probabilities for transition represent the improvement and induced move processes, but their values depend on the other output switches states. The Markov system contains r approximated output switch sub-systems each with a single state variable which is either 1 (bad) or 0 (good) as shown in Figure 5(b). Since all the combinations of output switch states are possible, the number of system states is the multiplication of all the output switches states. Since each output has only two states the system state can be encoded as a binary variable of r bits, such that bit S_i represents the state of output switch i. The resulting state space has 2^r system states. Consequently, the entire system has a single absorbing state which is when all the bits of the binary representation are 0. We can also assume all output switches start with some bad bins so the initial state value is $2^r - 1$. Before the observing state is reached, the total number of output-switches with some bad-balls, denoted by U, equals the number of 1's in the state binary value. Each one of these U sub-systems will initiate a ball move which may cause an induced move on some other sub-system. We assume the induced moves are equally spread, and thus the probability for an output switch to leave a good state B equals the probability for an induced moves to throw a ball in that output switch (since any ball move will change the state to a bad state):

$$B = \frac{U}{r} \tag{1}$$

The probability to stay in a good state denoted A is:

$$A = 1 - B = 1 - \frac{U}{r} \tag{2}$$

The probability to move from a bad state to a good state is denoted C, and is built from the impact of the two processes C_{imp} and C_{ind} for the probabilities for such move caused by improve or induced process respectively. The improvement process always selects a bad ball and thus improvement depends on the number of empty bins:

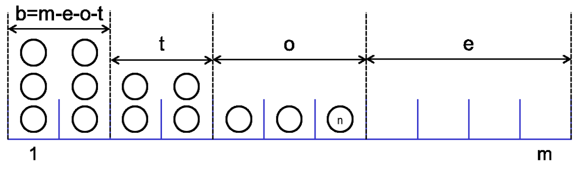


Fig. 6. For $p = 2$ we introduce new state variables: e=the number of empty bins, o= the number of bins with a single ball, t=the number of bins with two balls. b=the number of “bad” bins

$$C_{imp} = (m - n + 1)/m \tag{3}$$

For the induced-move process to cause a move from bad to good state we need to multiply the probability of an induced move by the probability a bad ball will be selected, and by the probability the move will be into an empty bin:

$$C_{ind} = \frac{U}{r}\frac{2}{n}\frac{(m - n + 1)}{m} \tag{4}$$

Combining the two contributions we get:

$$C = C_{imp} + C_{ind} - C_{imp}C_{ind} \tag{5}$$

The probability to stay in bad state D is thus:

$$D = 1 - C \tag{6}$$

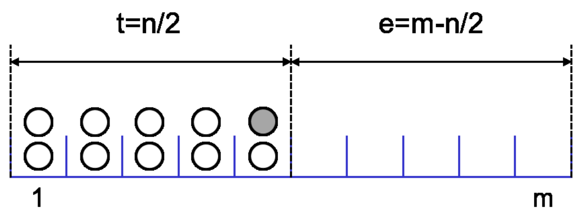
To build the Markov state transitions matrix the present state is represented as a binary variable: $S = S_{r-1}S_{r-2}\dots S_1S_0$ and the next state is represented as $Q = Q_{r-1}Q_{r-2}\dots Q_1Q_0$. Define E as the number of digits j where $S_j = Q_j = 0$, F the number of digits j where $S_j = 0 \neq Q_j$, G the number of digits j where $S_j = Q_j = 1$ and H the number of digits j where $S_j = 1 \neq Q_j$. The probability to move from state S to state Q is thus the multiplication of the probabilities of each one of the sub-systems to change or stay in their previous state:

$$P_{ij} = A^E B^F C^H D^G \tag{7}$$

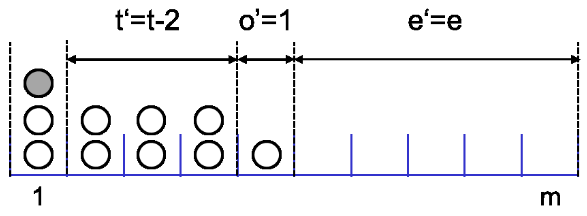
D. Last-Step Model for Flows of Half Link Bandwidth

As we realize by the previous section that the exponential convergence time is a result of the high probability for a bad induced move when flows use the full link bandwidth, we look for ways to reduce that probability. Intuitively, restricting flows to only use half the link bandwidth using $p = 2$ greatly reduces the probability for a bad induced-move. This is because all the cases where two flows are routed through the same link are valid, and only cases with three of more flows are not. In this section we take the same approach of modeling the last-step before convergence in order to obtain an approximation of the convergence time.

Unlike the Last-Step model for $p = 1$ that has just one absorbing state for the output switches, the case of two flows per link has several good states. As illustrated in Figure 6, to distinguish these states we introduce the following variables describing the ball distributions in each output switch: e, the number of empty bins; o, the number of bins with one ball; and t, the number of bins with two balls.



(a)



(b)

Fig. 7. The Markov model for the case of $p = 2$ has many observing states. As we are interested in the slowest convergence for $p = 2$ we focus on the observing state with the highest probability for bad induced move - B: (a) The absorbing state with maximal B, and (b) its neighbor bad state. The moved ball is shadowed.

It can be shown that for a state to be good the total number of balls in good (1 or 2 balls) bins must equal the number of balls n:

$$n = 2t + o \tag{8}$$

$$m = e + t + o \tag{9}$$

Or:

$$t = e - (m - n) \tag{10}$$

$$o = m - 2 \times e + (m - n) \tag{11}$$

As we are interested in the worst number of iterations to convergence (for the $p = 2$ case only) we need to find the absorbing state (o,t) with highest probability B for bad induced move. B, as a function of o and t is the probability to select a ball from a single bin and move it on a 2-balls bin, plus the probability to select a a ball from 2-balls bin and place it on another 2-balls bin:

$$B = \frac{o}{n}\frac{t}{m} + \frac{2t}{n}\frac{(t-1)}{m} = \frac{t}{nm}(o + 2t - 2) = \frac{t}{nm}(n - 2) \tag{12}$$

By calculating t' which zeros out the derivative of Equation 12 versus t we conclude the maximal B is obtained for $t' = n/2$ as shown in Figure 7(a). For that state the maximal B is:

$$B_{max} = \frac{n/2 - 1}{m} = \frac{n}{m}(\frac{1}{2} - \frac{1}{m}) \tag{13}$$

To reach convergence all the sub-systems need to reach the observing state shown in Figure 7(a), from the nearest bad-state shown in Figure 7(b). Similar to the $p = 1$ case, the probability C is then the sum of the probabilities of the *improvement* and the *induced-move* processes to move the subsystem from bad to good state:

$$C_{imp} = \frac{m - t'}{m} = 1 - \frac{n/2 - 2}{m} \tag{14}$$

$$C_{ind} = U\frac{3}{n}\frac{m - t'}{m} \tag{15}$$

The probability for induced moves with $p = 2$ has to take into account the probability for a ball to move without requiring a swap. To that end we could use the balls-and-bins distribution to predict the probability for a bin to have more than two balls. The construction of the Markov state transition matrix follows the same procedure as in Section III-C.

IV. Implementation Guidelines

This section discusses the feature set required for the implementation of an oblivious-adaptive-routing system. To enable an efficient hardware implementation, the proposed mechanisms differ from the model described in Section II in several aspects.

The first implementation detail deals with the implications of folding the Clos network to form a Folded-Clos network, i.e. a Fat-Tree. To create a Fat-Tree a Clos network is folded along the middle-switches axis so the input and output Clos switches are implemented by the same Fat-Tree switches. So, these switches are required to support full-duplex rather than half-duplex links.

To meet the required behavior described in Section II, the switches need to extend their deterministic routing and provide random assignment of output ports for flows. The dynamic routing engine needs to be "sticky" and remember previous assignments, such that routing changes only occur if required. The adaptation is triggered either by receiving an Explicit Adaptation Request (EAR) from an output switch, or when congestion is observed on the previously assigned output port.

Reassignment of a flow output port is only allowed on Clos input switches (Fat-Tree up-going ports of the leaf-level switches). The Fat-Tree switches implement a configurable marking of ports as up-port to limit adaptation to these ports without the need to hard-code port assignments.

In real networks, the adaptation delay, from the moment an EAR is produced until the routing is changed, is not negligible. The main reason is the need to time-multiplex the request with the data packets already being sent on the same port. Such a delay means that if EARs are generated without waiting for the impact of previous EARs, unnecessary adaptations are performed, which reduces the probability to reach a non-blocking routing assignment. To overcome that effect, a timer is used to throttle the number of routing reassignments. Furthermore, introducing "sampling" into the EAR generation increases the probability of larger flows to be rerouted in case of congestion. This increases the probability of resolving the congestion point with less adaptations.

While the analyzed model requires all flows to have the exact same bandwidth, a real system has to deal with a mixture of flows with arbitrary bandwidth. In practice this means that the implementation cannot declare congestion by counting "flows" assuming they are all of the same bandwidth. Instead, it makes more sense to evaluate a transmit-port congestion, in mechanisms similar to those proposed by the IEEE 802.1Qau known as QCN [32].

Algorithm 1 On Queue or De-Queue of Data Flits (numFlits, TP, RP, to DEST)

```
Update number of flits queued for the Transmit Port (TP)
if enough time from port change for this flow and TP is congested then
    if Switch is a middle switch then
        Send EAR though the Receive Port (RP) flits were received on
    else
        if TP is an up-going-port then
            Adapt the output port for DEST
            Possibly swap with another DEST if the new output port is busy
        end if
    end if
end if
```

Algorithm 2 On Receiving an EAR (RP, DEST)

```
if RP is an up-going-port then
    Adapt the output port for DEST
    Possibly swap another DEST if the new output-port is busy
end if
```

Another difference between our analyzed model and a real-life implementation involves the concurrency of bad-flow re-routes. Congestion-based bad-link detection means that the knowledge about bad links is not available at the receiving switch (the output switch in our model), but in the middle switch. In the model used in previous sections, the output switch requires this information in order to choose a single worst bad-flow to be re-routed. Selecting the worst bad-link implies the existence of a protocol for each middle switch to notify the output switches to which it connects about bad links and their severity. To avoid the latency and complexity of such a protocol, the proposed implementation does not enforce a single bad-flow transition per output switch per iteration. Instead, the responsibility for requesting re-routing of bad flows is given to the middle switches that use the same QCN-like monitoring to detect congestion. When congestion is detected, the middle switches send EAR requests to the relevant input switches. These notifications do require a special signaling protocol to be delivered. The algorithms for EAR generation and forwarding, as well as for determining when to adapt to output congestion, are depicted in Algorithms 1 and 2. Algorithm 1 is run by the switches for output ports (ports towards the network output). Algorithm 2 is only run by the input switches.

Finally, it is suggested to add a mechanism to dynamically throttle input flows as a function of the need to perform distributed-adaptive-routing. With such a mechanism the system enjoys the full link bandwidth for permutations which are non-blocking under static routing and applies adaptive routing and flow bandwidth throttling only when congestion occurs. In our evaluation we did not incorporate such a mechanism.

Algorithm 3 Distributed Adaptive Routing Simulation Main Loop

```
Draw a random or worst permutation as $dst[src]$
for all $(src, dst)$ pair do
    $M[src, dst] = mod(src, n)$
end for
$Iterations = 0$
while Any bad link (depends on $p$) do
    Iterations++
    Randomly select one $(src, dst)$ from the worst link for each out-switch
    for all Bad $(src, dst)$ selected, in random order do
        Randomly select a new value for $M[s, d]$
        Move the $(src, dst)$ to the new middle switch
        Optionally swap with previous flow on that link
    end for
end while
Report Iterations
```

V. Evaluation

A. Traffic Matrix

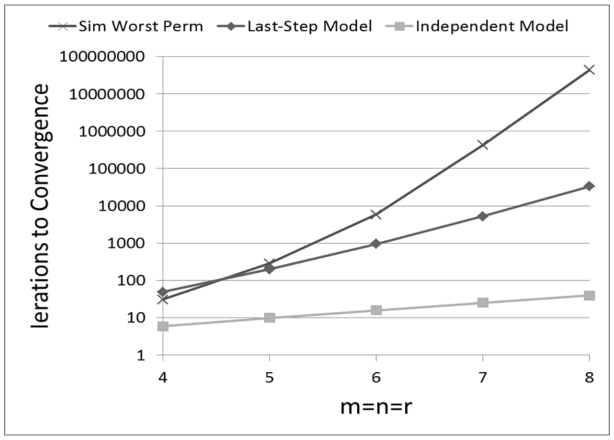
Evaluation of the worst convergence time using simulations relies on the ability to check many traffic-matrix permutations. As the number of possible permutations is extremely large, it is important to focus on the permutations that are presumed to have the worst convergence time, or at least a large one. These so-called *worst permutations* are derived by contrasting them to the set of permutations that are fastest to converge. In the fastest converging permutations, all flows originating on the same input switch are destined to the same output switch. For such permutations it is enough that each input switch spreads its own outputs to avoid bad links and provide good routing. Such assignment is possible if $m \times p \geq n$ (this is also the rearangeable non-blocking condition for *1-rate* Clos with p flows per link). Therefore, intuitively, a permutation where each output switch receives flows from different input switches will be among the hardest to converge, as it will require the most synchronization between the input switches, which do not talk to each other.

B. Analyzed System Simulation Model

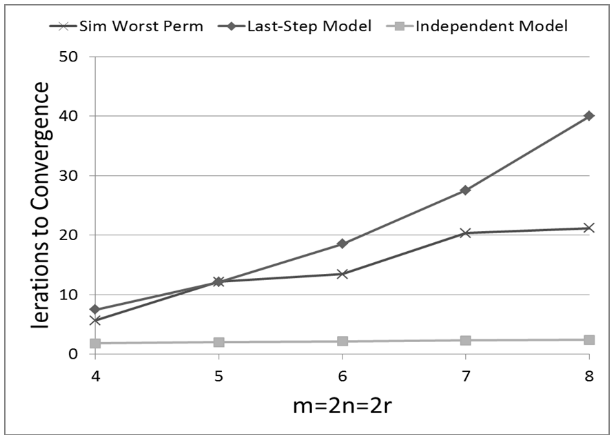
A dedicated simulation program was written to model the system described in Section II. The data structure used is a simple matrix $M[s, d]$, where (s, d) represents the source and destination for each flow, and the value of $M[s, d]$ denotes the middle switch assigned to the (s, d) flow. The simulation algorithm is depicted in Algorithm 3. The program optionally either starts from a fully randomized permutation or from a random one that meets the condition of a so-called worst permutation, as described above. Each point of simulation result is obtained by simulating a batch of 1000 permutations and then continues to simulate new batches until the average number of iterations required to converge changes by less than 1%.

The models of Sections III-B, III-C and III-D were coded in Matlab [1], to form an observable Markov chain matrix following the theory presented in [33].

[1]The network size in the presented Matlab-based evaluation is limited by the exponential state space nature of Markov representations and the capacity of our version of Matlab.



(a)

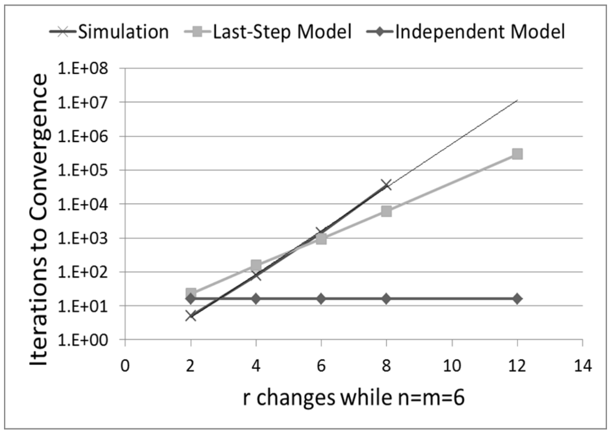


(b)

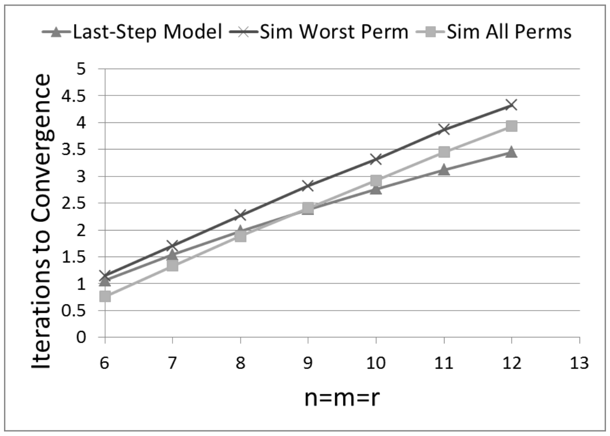
Fig. 8. Expected convergence time for $p = 1$, i.e. at most one flow per link. The plots compare the approximations of the Independent single-output-switch model and the Last-Step model, against simulated results of worst permutations, in the cases where (a) the topology is rearrangeable non-blocking with $m = n = r$. and (b) the topology is strictly non-blocking with $m = 2n = 2r$.

The first comparison made is for the rearrangeable non-blocking topologies, in the case where $p = 1$, i.e. there is at most one flow per link. The number of iterations to reach convergence is provided in Figure 8(a). As expected, the Independent output switch model of Section III-B is vastly optimistic, while the Last-Step model of Section III-C is closer to the simulated results. The Last-Step model is pessimistic for very small n as it assumes all output switches start in a bad state, which is not the case for very small n.

For the strictly non-blocking case in Figure 8(b), it can be seen that the simulation predicts convergence times many orders of magnitude smaller than the rearrangeable non-blocking case, but still mostly above 10. The Independent output-switch model is optimistic, and the Last-Step model is pessimistic, probably due to its assumption that all output switches start at a bad state. In this case of a strictly non-blocking topology, due to the over-provisioning of the network, there are higher chances that some output switches will start in a good state. Also note that the change of slopes on the simulation curve may be attributed to the change between even and odd port numbers, and note that routing in Clos best fits even number of ports.



(a)



(b)

Fig. 9. (a) The dependency on r of the number of iterations to convergence for $p = 1$ and constant $m = n = 6$. (b) Comparison of time to convergence for Clos($n = m = r$) in the case of half-link-rate flows ($p = 2$) using the Last-Step Model and a dedicated simulation model.

To strengthen the point that the systems are dependent, we show the dependency of the convergence time on the number of parallel output switches in Figure 9(a). The convergence is plotted for different values of r and a fixed $n = m = 6$. It can be observed that *the dependency on r is exponential*. The Last-Step model shows a lower slope, which we suspect is a result of the approximation of using a single bad state. Note that the Independent model does not depend on r at all, which exhibits yet another limitation of this model.

The Last-Step Markov model of Section III-D for the case of half-link capacity flows was simulated on a rearrangeable non-blocking topology and the number of iterations to convergence is provided in Figure 9(b). The number of iterations required to reach convergence is *shown to be very small even for large values of n*. As can also be seen, the Markov model is optimistic for larger networks. We attribute this behavior to the approximation used by this model, which defines a single bad state for each output-switch sub-system.

C. Implementable System Simulation Model

A large compute cluster would have been the obvious choice for the evaluation of an implemented adaptive routing system. However, hardware that implements our proposed Explicit Adaptation Request messaging described above was not available to us. Instead, we used a flit-level simulator for InfiniBand that accurately models flow dynamics, network queuing and arbitration, as described below.

TABLE II
SIMULATION RESULTS FOR A 1152-HOSTS CLUSTER

Parameter	SNB Full BW: 40Gbps	RNB Full BW: 40Gbps	RNB Half BW: 19.2Gbps
Avg-of-Avg Throughput	37.7Gbps	23.0Gbps	18.9Gbps
Min-of-Avg Throughput	36.5Gbps	20.7Gbps	18.9Gbps
Avg-of-Avg Network Latency	17.0μsec	32μsec	5.8μsec
Max-of-Max Network Latency	877μsec	656μsec	11.7μsec
Avg-of-Avg Out-of-order / In-order Ratio	**1.87**	**5.2**	**0.0023**
Max-of-Avg Out-of-order / In-order Ratio	3.25	7.4	0.0072
Avg-of-Avg Out-of-order Window	6.9pkts	5.1pkts	2.3pkts
Max-of-Avg Out-of-order Window	8.9pkts	5.7pkts	5.0pkts

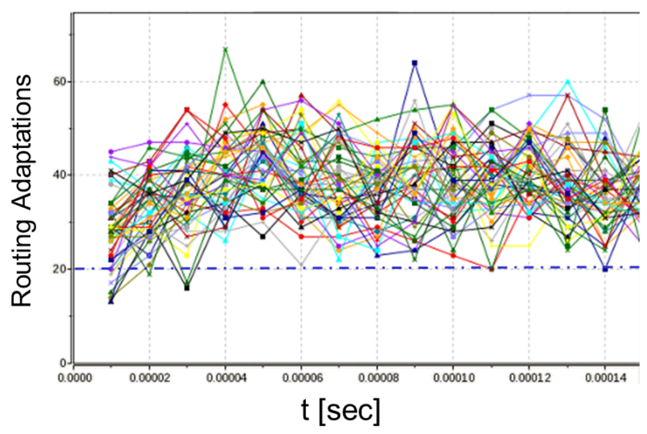
The OMNet++ [34] based InfiniBand flit-level simulation model [35] is commonly used for predicting bandwidth and latency for InfiniBand networks [31] [36] [37]. In order to evaluate the proposed implementation guidelines presented in Section IV, a new packet-forwarding module was added to the switches. This module implements all the algorithms to detect link capacity overflow, provide flow re-route, swap output ports, and introduce a signaling protocol to carry the Explicit Adaptation Requests (EAR) between switches.

The overhead and timing of the EAR protocol is accurately modeled by encapsulating EARs as 8-byte messages similarly to the flow-control packets of InfiniBand, and sending them through the regular packet send queues.

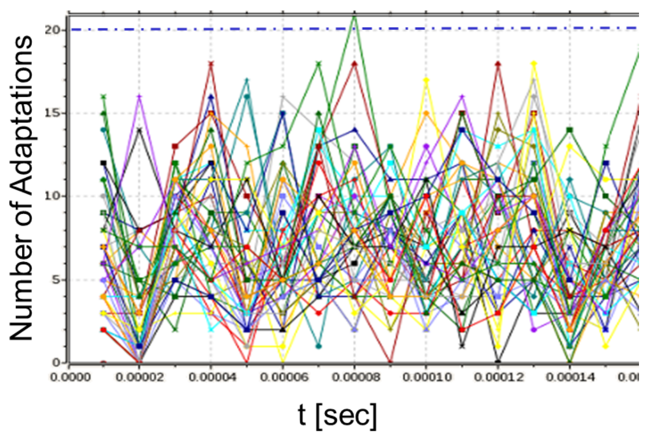
The simulations performed are of two topologies containing 1152 hosts: The rearrangeable non-blocking topology is a folded CLOS(24, 48, 24) equivalent to the XGFT(2; 24,48; 1,24) (eXtended Generalized Fat-Tree) topology. The strictly non-blocking topology has double the number of middle switches, i.e. is a folded CLOS(24, 48, 48), equivalent to an XGFT(2; 24,48; 1,48) fat-tree. The strictly non-blocking topology is simulated to provide a fair comparison to the case of half-rate flows ($p = 2$), as it provides double the links. The model assumes a link capacity of 40Gbps, and hosts may send data at that speed or be throttled to 20Gbps +/- 0.8Gbps.

The traffic pattern applied to the system is a sequence of random permutations. In each permutation each host sends data to a single random destination and receives data from a single random source. The hosts progress through their sequence of destinations in an asynchronous fashion sending 256KB to each destination.

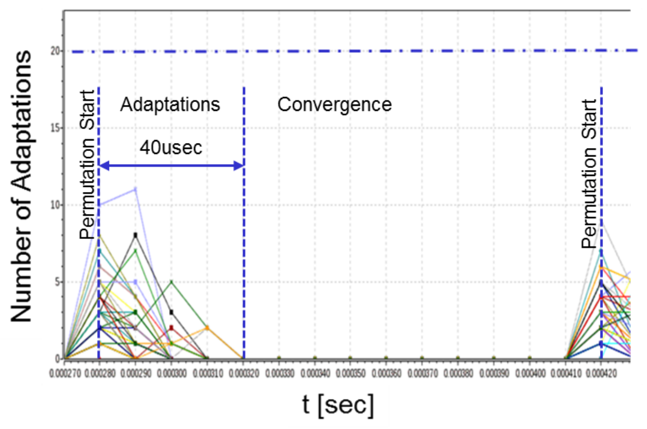
The simulation tracks the number of routing changes performed by each switch in periods of 10μsec, as well as the final throughput at each of the network egress ports. The ratio of packets delivered out-of-order to those provided in-order is also measured. Another measured variable is the out-of-



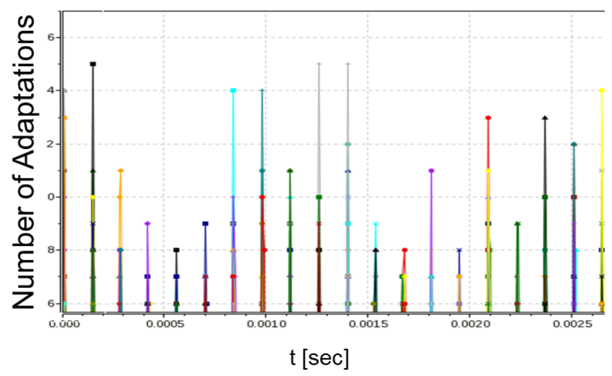
(a) Strictly non-blocking 40Gbps flows



(b) Rearrangeable non-blocking 20.8Gbps flows



(c) Rearrangeable non-blocking 19.2Gbps flows



(d) Many permutations of 19.2Gbps flows

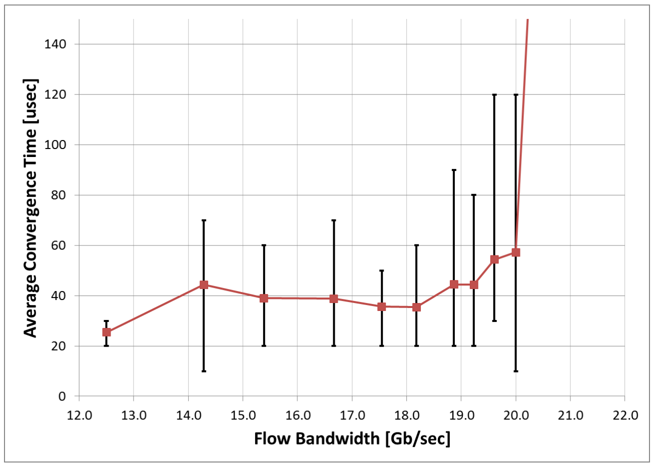
Fig. 10. The number of re-routing events on each input switch of the 1152-node fat tree, accumulated over 10μsec time-periods with 256KB messages, for (a) a strictly non-blocking topology with flows of 40Gbps, (b) a rearrangeable non-blocking topology with 20.8Gbps flows, and (c) a rearrangeable non-blocking toplogy with 19.2Gbps flows. The plots in (a) and (b) focus on a single 150μsec permutation period and show that convergence is not met - adaptations do not stop throughout the entire permutation, while in (c), for the rearrangeable non-blocking 19.2Gbps case, convergence is reached within 40μsec. Finally, (d) shows a longer time period including a set of random permutations of 256KB that are applied to the rearrangeable non-blocking 19.2Gbps case. The convergence time is roughly 1/10 of the permutation time.

order packet window size, defined as the gap in the number of packets, as observed by the receiving host. This variable is a clear indication for the feasibility of implementing a re-order buffer. The bandwidth, latency and out-of-order percent and window size results are presented in Table II. The values are taken as average, max or min value over all receivers of the average or max value measured on each receiving host. For example, the min of average throughput means that each egress port throughput is averaged over time, and the reported number is the minimal value over all the egress ports. To establish a fair comparison we focus on the results of the two cases when only half of the network resources are used: strictly non-blocking 40Gbps (first data column) and rearrangeable non-blocking 19.2Gbps (third data column).

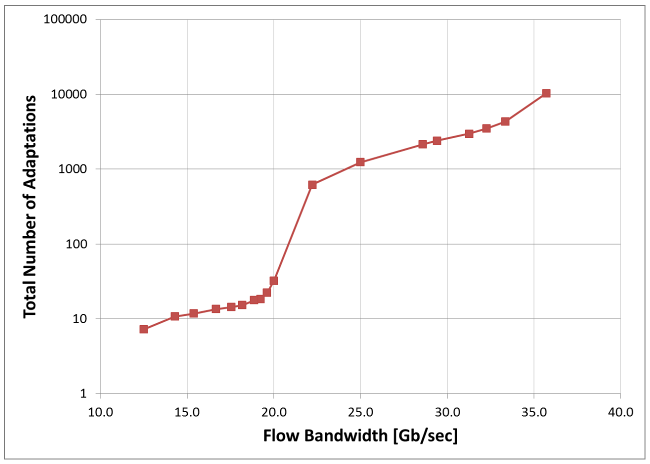
We can see that although the bandwidth provided by the strictly non-blocking case with $p = 1$ is higher, any transport that would require retransmission due to out-of-order delivery would actually fail to work on the strictly non-blocking case with $p = 1$, since *only one out of three packets is provided in order* (ratio of about 2). The latency of the network is also impacted by not reaching a steady state, thus showing a much longer latency.

The routing convergence provided by the $p = 2$ case is most visible when inspecting the number of routing changes per 10μsec. Figure 10 shows on each line the number of adaptations conducted by a specific switch in each 10μsec period. Figures 10(a) and Figure 10(b) shows the accumulative number of adaptations per switch during the first permutation for strictly non-blocking 40Gbps and rearrangeable non-blocking 20.8Gbps respectively. Figure 10(c) show the accumulative number of adaptations per switch during the third permutation and for rearrangeable non-blocking 19.2Gbps. It can be observed that *routing is constantly changing for the cases of the strictly non-blocking 40Gbps and the rearrangeable non-blocking 20.8Gbps, and stabilizes fast for the case of the RNB 19.2Gbps*. A long sequence of 256KB message permutations on rearrangeable non-blocking 19.2Gbps is shown in Figure 10(d). By inspecting the cumulative number of adaptations of all the network switches and measuring the length of the adaptation period after each permutation, it can be observed that *adaptive-routing reaches a non-blocking assignment for all permutations in less than 80μsec*.

To further demonstrate the *singularity around half-*

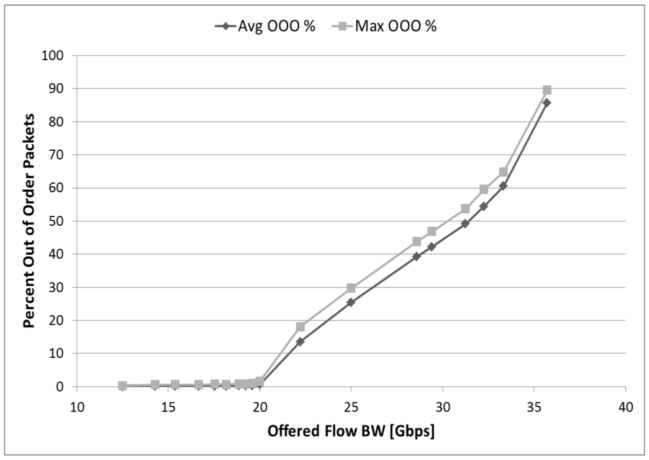


(a)

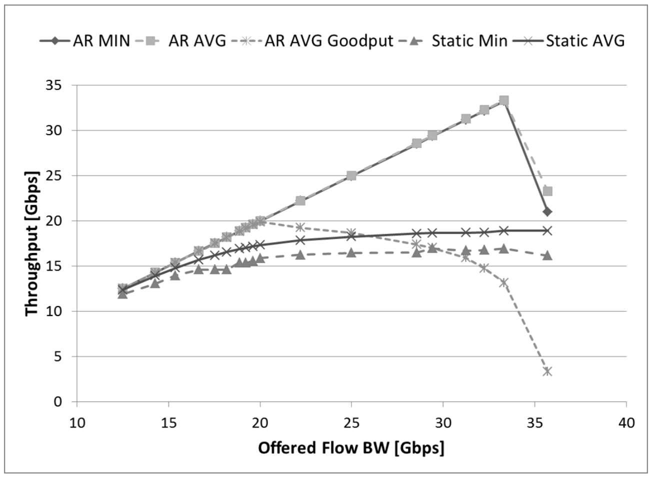


(b)

Fig. 11. Simulated cluster of 1152 nodes, running a a set of random permutations of 256KB for different flow bandwidths. (a) shows the average and range of convergence time from the start of the permutation period. It can be seen that flow bandwidth above half the link bandwidth simply does not converge. (b) depicts the total number of adaptations throughout the entire simulation period on a logarithmic scale.



(a)



(b)

Fig. 12. (a) The average percentage of out-of-order (OOO) packets at the network outputs. A a dramatic increase can be observed above 20Gbps offered flow bandwidth. (b) The obtained average and minimal throughput and goodput over all network outputs, as a function of the offered flow bandwidth. It is clearly seen that while the average bandwidth for static destination routing is saturating at roughly at 17Gbps the adaptive routing system throughput reaches a peak at 28Gbps and its goodput maximizes at half the link bandwidth at 20Gbps. Goodput for AR is calculated as the throughput times the percentage of in-order packets. The minimal goodput (on the worst network output) is not plotted as it overlaps with the average curve.

bandwidth flows, as concluded by our last-step model, we run the set of random permutations on the rearrangeable non-blocking topology with 1152 nodes, for several flow bandwidths, from 1/5- to full-link bandwidth. We measure the convergence time from the beginning of the permutation and the total number of adaptations over all permutations. Both the time for convergence (in a resolutions of 10μsec) and the total number of adaptations are shown in Figure 11. Figure 11(a) shows clearly that convergence does not happen within the 256KB permutation period when the flow bandwidth is above half of the link bandwidth. When flow bandwidth values are smaller, it can be observed that the convergence time is not monotonically decreasing. This can be attributed to the dependency of the EAR generation on the queue build-up time, which depends on the flow bandwidth. For the simulated 32KB per input buffer and 40Gbps links, just filling up the buffer would take about 20μsec for 2 flows of 1/5 link bandwidth.

Figure 11(b) further depicts the total number of adaptations within the entire run. It shows that the number of adaptations grows by half an order of magnitude as the flow bandwidth increases from 1/5 link bandwidth to half link bandwidth, and then by 1.5 orders of magnitude just crossing the half link bandwidth point.

Figure 12(a) shows the percentage of packets that are delivered out-of-order for the same topology and set of permutations described above. It can be observed that while there almost no out-of-order packets for the cases of flow bandwidth $<$ 20Gbps, once the flow bandwidth increases above that value, a significant percentage of packets are delivered out-of-order. For many transports (TCP and InfiniBand Reliable-Connected) such events cause significant reduction of goodput.

Finally, we compare the average and minimal (over all end-ports) output bandwidth for static destination-based routing to the throughput and goodput obtained by the distributed adaptive routing system. We use the same network and apply the same set of random permutations of 256KB size messages. The simulations utilize either static routing (D-Mod-K) or distributed adaptive routing. The bandwidth as a function of the offered flow bandwidth is plotted in Figure 12(b). The graphs show that while the static routing bandwidth saturates at 19Gbps, the distributed adaptive routing supports the offered flow bandwidth up to a throughput of 28Gbps. The comparison of the average and the minimal bandwidth on all network outputs highlights another advantage of the distributed adaptive routing system which provides equal service and does not favor some outputs over the others. The adaptive-routing system provides a significant improvement in goodput over the deterministic routing. Note that for 20Gbps flows the distributed-adaptive-routing provides the minimal goodput of 20Gbps while the worst network outputs only reach 17Gbps when the applied flow bandwidth is ˜30Gbps. Also note that the permutations used in our evaluations are far from being the worst possible permutations. On the simulated network, these adversarial permutations would saturate at $1/24$ of the link bandwidth. These results are also assuming no transport penalty for congestion, i.e. a perfect TCP congestion control or lossless network being used for the static routing case.

VI. Discussion and Conclusions

In this paper we find sufficient conditions allowing distributed oblivious-adaptive-routing to converge to a non-blocking routing assignment within a very short time, thus making it a viable solution for adaptive routing for medium-to-long messages on fat-trees.

Convergence is shown to require flows that do not exceed half of the link capacity, which raises the question of whether it is worth to pay that high price. Note that actually in our proposed Adaptive Routing only the edge links of the network are operated at half the core network link bandwidth. Many of the network links do route more than one flow and thus utilize the full link capacity. To deal with the contention caused by high-volume correlated flows, an alternative approach would be to rely on a centralized traffic engineering engine that can throttle traffic as necessary or perform re-routes. However, such an approach may have scalability issues. As the number of flows correlates to the number of cluster nodes, a central unit is likely to become a bottleneck. Additional approaches [8] to provide adaptive routing based on protocols that convey the system state to each switch are also less scalable due to the state size on every switch, the number of messages to provide state updates, and the synchronous change of traffic which makes previous states irrelevant.

The developed approximate model provides the insight that the origin of the long convergence time is the *interdependency of re-route events on the different output switches*, as imposed by the topology. It was shown that the time it takes to converge to a non-blocking routing is *exponential with the number of input or output switches*. For that reason, the probability of creating bad links on a single output switch has a major impact on the convergence time. For rearangeable-non-blocking CLOS, limiting the traffic flows to half or less of the link bandwidth reduces this probability for creating bad links to less than 0.5, and therefore provides fast convergence.

Finally, we propose a simple system architecture for the signaling needed for adaptation, and simulate it to show how it converges within 20-80μsec on a 1152-host network. The insights provided by this research should help in providing a self-routing solution to long messages in data-center applications of various fields.

We leave an evaluation of larger topologies like 3-level fat-trees or Dragonflies as future work. Finally, we plan to extend our evaluation for cases of mixed flow-bandwidth and duration.

Acknowledgment

We would like to thank Marina Lipshteyn of Mellanox and Israel Cidon, Yossi Kanizo and Erez Kantor from the Technion for their support and insight. This work was partly supported by the Intel ICRI-CI Center, the Israel Ministry of Science and Technology, and the European Research Council Starting Grant No. 210389.

References

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: dynamic flow scheduling for data center networks,” in *Proc. 7th USENIX conference on Networked systems design and implementation*, ser. NSDI’10. Berkeley, CA, USA: USENIX Association, 2010, p. 1919.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” in *Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, ser. EuroSys ’07. New York, NY, USA: ACM, 2007, p. 5972.

[3] T. White, *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., Oct. 2010.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, “Managing data transfers in computer clusters with orchestra,” in *Proc. ACM SIGCOMM 2011 conference*, ser. SIGCOMM ’11. New York, NY, USA: ACM, 2011, p. 98109.

[5] M. Raab and A. Steger, “Balls into bins a simple and tight analysis,” *Randomization and Approximation Techniques in Computer Science*, pp. 159–170, 1998.

[6] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, “Understanding TCP incast throughput collapse in datacenter networks,” in *Proc. 1st ACM workshop on Research on enterprise networking*, ser. WREN ’09. New York, NY, USA: ACM, 2009, p. 7382.

[7] J. C. Martnez, J. Flich, A. Robles, P. Lpez, and J. Duato, “Supporting fully adaptive routing in InfiniBand networks,” in *Proc. 17th International Symposium on Parallel and Distributed Processing*, ser. IPDPS ’03. Washington, DC, USA: IEEE Computer Society, 2003, p. 44.1, ACM ID: 838493.

[8] C. Minkenberg, M. Gusat, and G. Rodriguez, “Adaptive routing in data center bridges,” in *Proc. 2009 17th IEEE Symposium on High Performance Interconnects*. Washington, DC, USA: IEEE Computer Society, 2009, p. 3341, ACM ID: 1634466.

[9] C. Gomez, F. Gilabert, M. Gomez, P. Lopez, and J. Duato, “Deterministic versus adaptive routing in fat-trees,” in *2007 IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, Mar. 2007, pp. 1–8.

[10] M. Koibuchi, J. C. Martinez, J. Flich, A. Robles, P. Lopez, and J. Duato, “Enforcing in-order packet delivery in system area networks with adaptive routing,” *J. Parallel Distrib. Comput.*, vol. 65, no. 10, pp. 1223–1236, Oct. 2005.

[11] W. Wu, P. Demar, and M. Crawford, “Sorting reordered packets with interrupt coalescing,” *Computer Networks*, vol. 53, no. 15, pp. 2646–2662, Oct. 2009.

[12] A. V. Gerbessiotis and L. G. Valiant, “Direct bulk-synchronous parallel algorithms,” *J. Parallel Distrib. Comput.*, vol. 22, no. 2, p. 251267, Aug. 1994.

[13] T. Hoefler, P. Kambadur, R. Graham, G. Shipman, and A. Lumsdaine, “A case for standard non-blocking collective operations,” in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science, F. Cappello, T. Herault, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2007, vol. 4757, pp. 125–134.

[14] T. Hoefler, T. Schneider, and A. Lumsdaine, “Multistage switches are not crossbars: Effects of static routing in high-performance networks,” in *2008 IEEE International Conference on Cluster Computing*. IEEE, Oct. 2008, pp. 116–125.

[15] X. Yuan, “On nonblocking folded-clos networks in computer communication environments,” in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, May 2011, pp. 188–196.

[16] N. Jiang, J. Kim, and W. J. Dally, “Indirect adaptive routing on large scale interconnection networks,” *ACM SIGARCH Computer Architecture News*, vol. 37, p. 220231, Jun. 2009, ACM ID: 1555783.

[17] J. Kim, W. J. Dally, and D. Abts, “Adaptive routing in high-radix clos network.” ACM Press, 2006, p. 92.

[18] S. Scott, D. Abts, J. Kim, and W. J. Dally, “The BlackWidow high-radix clos network,” in *Proc. 33rd annual international symposium on Computer Architecture*, ser. ISCA ’06. Washington, DC, USA: IEEE Computer Society, 2006, p. 1628.

[19] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, “Blue Gene/L torus interconnection network,” *IBM J. Research and Development*, vol. 49, no. 2.3, pp. 265–276, Mar. 2005.

[20] G. BLOCH, D. CRUPNICOFF, M. KAGAN, I. BUKSPAN, I. RABENSTEIN, A. WEBMAN, and A. MARELLI, “High-performance adaptive routing,” U.S. Patent, 2012, publication number: US 2011/0096668 A1 U.S. Classification: 370/237.

[21] M. Gusat, D. Crisan, C. Minkenberg, and C. DeCusatis, “R3C2: reactive route and rate control for CEE,” in *High-Performance Interconnects, Symposium on*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 50–57.

[22] E. J. Anderson and T. E. Anderson, “On the stability of adaptive routing in the presence of congestion control,” in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2. IEEE, Apr. 2003, pp. 948–958 vol.2.

[23] D. Gamarnik, “Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks,” in *Proc. thirty-first annual ACM symposium on Theory of computing*, ser. STOC ’99. New York, NY, USA: ACM, 1999, p. 206214.

[24] A. Smiljanic, “Rate and delay guarantees provided by clos packet switches with load balancing,” *IEEE/ACM Trans. Netw. (TON)*, vol. 16, no. 1, pp. 170–181, 2008.

[25] A. Jajszczyk, “Nonblocking, repackable, and rearrangeable clos networks: fifty years of the theory evolution,” *IEEE Commun. Mag.*, vol. 41, no. 10, pp. 28–33, 2003.

[26] V. E. Benes, *Mathematical theory of connecting networks and telephone traffic*. Academic press New York, 1965, vol. 332.

[27] D. Z. Du, B. Gao, F. K. Hwang, and J. H. Kim, “On multirate rearrangeable clos networks,” *SIAM J. Comput.*, vol. 28, no. 2, p. 463470, 1998.

[28] S. Liew, M.-H. Ng, and C. Chan, “Blocking and nonblocking multirate clos switching networks,” *IEEE/ACM Trans. Netw.*, vol. 6, no. 3, pp. 307–318, 1998.

[29] B. Douglass and A. Oruc, “On self-routing in clos connection networks,” *IEEE Trans. Comput.*, vol. 41, no. 1, pp. 121–124, 1993.

[30] A. Youssef, “Randomized self-routing algorithms for clos networks,” *Computers & Electrical Engineering*, vol. 19, no. 6, pp. 419–429, Nov. 1993.

[31] E. Zahavi, “Fat-trees routing and node ordering providing contention free traffic for MPI global collectives,” *J. Parallel and Distributed Computing*, no. Communication Arch for Scalable Systems, 2008, pending.

[32] N. Finn, “802.1Qau-2010,” Tech. Rep., 2010. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5454061

[33] S. Karlin and H. M. Taylor, *An Introduction to Stochastic Modeling, Third Edition*, 3rd ed. Academic Press, Feb. 1998.

[34] A. Varga, “OMNET++,” 2004. [Online]. Available: http://www.omnetpp.org/

[35] “InfiniBand(TM) Simulation Macro Model Macro http://www.omnetpp.org/omnetpp/doc_details/2070-infiniband.”

[36] E. G. Gran, M. Eimot, S. A. Reinemo, T. Skeie, O. Lysne, L. P. Huse, and G. Shainer, “First experiences with congestion control in InfiniBand hardware,” in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, Apr. 2010, pp. 1–12.

[37] E. G. Gran, E. Zahavi, S.-A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, “On the relation between congestion control, switch arbitration and fairness.” IEEE, May 2011, pp. 342–351.

Eitan Zahavi is a Ph.D. candidate at the Technion Electrical Engineering department. He is also a senior principal engineer in Mellanox Technologies. Eitan earned his B.Sc. and M.Sc. in Electrical Engineering from the Technion, Israel in 1987 and 2012 respectively. During twenty years in the industry worked in Intel as a Circuit designer, EDA developer and architect. Co-founder of Mellanox at 1999 and since then leads the company Design Automation group and architects InfiniBand networks with focus on their management aspects. His recent research interests include High Performance Computing interconnects and Data Center Networks. Serves as a co-chair of the InfiniBand trade association Management Working Group.

Isaac Keslassy (M’02, SM’11) received his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 2000 and 2004, respectively. He is currently an associate professor in the Electrical Engineering department of the Technion, Israel. His recent research interests include the design and analysis of high-performance routers and multi-core architectures. The recipient of the European Research Council Starting Grant, the Alon Fellowship, the Mani Teaching Award and the Yanai Teaching Award, he is an associate editor for the IEEE/ACM Transactions on Networking.

Avinoam Kolodny received his doctorate in microelectronics from Technion - Israel Institute of Technology in 1980. He joined Intel Corporation, where he was engaged in research and development in the areas of device physics, VLSI circuits, electronic design automation, and organizational development. He has been a member of the Faculty of Electrical Engineering at the Technion since 2000. His current research is focused primarily on interconnects in VLSI systems, at both physical and architectural levels.