Tapping into the Router's Unutilized Processing Power

Marat Radan, Isaac Keslassy
Technion
{radan@tx, isaac@ee}.technion.ac.il

***Abstract*—The growing demand for network programmability has led to the introduction of complex packet processing features that are increasingly hard to provide at full line rates.**

In this paper, we introduce a novel load-balancing approach that provides more processing power to congested linecards *by tapping into the processing power of underutilized linecards*. Using different switch-fabric models, we introduce algorithms that aim at minimizing the total average delay and maximizing the capacity region. Our simulations with real-life traces then confirm that our algorithms outperform current algorithms as well as simple alternative load-balancing algorithms. Finally, we discuss the implementation issues involved in this new way of sharing the router processing power.

I. Introduction

A. Motivation

The emerging and overwhelming trend towards a *software-based approach to networks* (through network function virtualization, software-defined networking, and expanded use of virtual machines) increasingly lets the network managers configure the packet processing functions needed in their network. For instance, one network manager may want to alter video packets by inserting ads or changing the streaming rate, while another manager may want to filter packets based on a specific key within the packet payload.

This growing demand for *network programmability* has already forced several vendors to integrate full 7-layer processing within switches and routers. For example, EZChip has recently introduced NPS (Network Processor for Smart networks), a 400Gbps C-programmable NPU (Network Processing Unit) [1]. Cisco also promotes its DevNet developer effort for ISR (Integrated Services Router) [2], and already put Akamai's software onto its ISR-AX branch routers.

The processing complexity of the programable features defined by the network managers is hard to predict, and can also widely vary from packet to packet. These features can easily go beyond standard heavy packet processing tasks such as payload encryption [3], intrusion detection [4], and worm signature generation [5], [6], which already have a complexity that can be two orders of magnitude higher than packet forwarding [7]. Technology improvements can help in part, in particular through the increase in CMOS technology density and the growth of parallel architectures [8]. Still, these new features clearly make it hard for vendors to provide a guarantee about the packet line rates at which their processors will process all packets.

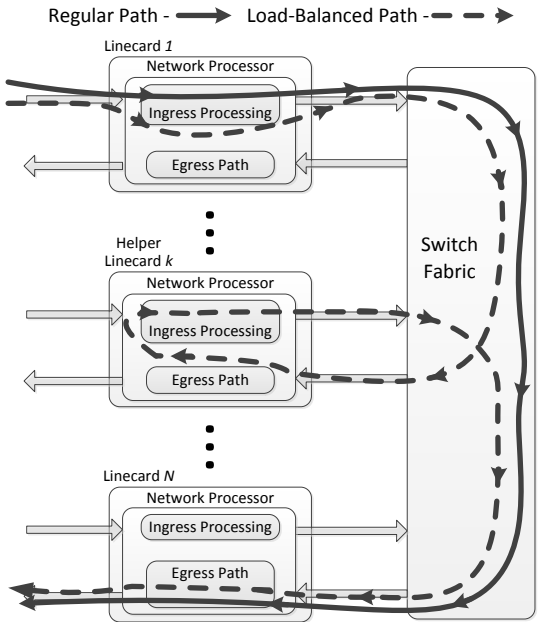


Fig. 1: Generic switch fabric with N linecards. The load-balanced path allows each packet to be processed at a linecard different from its source linecard, enabling the source linecard to tap into the router's unutilized processing power.

The goal of this paper is to help vendors introduce a new architectural technique that would enable routers to better use their processing resources to deal with these emerging programmable features. We do so by suggesting *a novel multiplexing approach that better utilizes the unused processing power of the router.*

Specifically, Figure 1 illustrates a router with N linecards, each containing an input and an output. Assume that linecard 1 experiences heavy congestion because its incoming packets require too much processing. Then its queues will fill up, and eventually it will drop packets. Meanwhile, the other router linecards could be in a near-idle state, with their processing power left untapped. This additional processing power would be invaluable to the congested linecard.

The main idea of this paper is to tap into the unused processing power of the idle router linecards. We do so by load-balancing the packets among the linecards. As illustrated in the figure, suppose linecard 1 is heavily congested while linecard k is idle. Then a packet arriving at linecard 1 could be sent by linecard 1 to linecard k, where it would be processed, and later switched to its output destination. Using

this technique, linecards with congested ingress paths can send workload to other linecards through the switch fabric. By doing so, the available processing power for the traffic arriving to a single linecard would potentially expand up to the entire router's processing power, depending on the other arrival rates and on the switch fabric capacity.

Of course, many challenges exist to implement this idea. For instance, the load-balancing mechanism may require an update mechanism to collect and distribute the information on the congestion of each linecard, together with an optimization algorithm that would take this information and decide when to load-balance packets. There are also additional considerations such as potential packet reordering and a more complex buffer management. We discuss these issues later in the paper.

B. Contributions

The main contribution of this paper is a novel technique for load-balancing packet processing between router linecards to achieve better processing performance.

Our goal is to expand the capacity region of the router by load-balancing and diverting traffic away from congested linecards. However, load-balancing more and more packets may also lead to a congestion of the switch fabric. Therefore, we need to trade off between reducing the *processing load* at the linecards, and reducing the *switching load* at the switch fabric. To help us develop some intuition about the impact of the switch fabric architecture on this trade-off, we present three increasingly-complex queueing models of switch-fabric architectures:

At first, we neglect the impact of the switch fabric architecture, by assuming that the switch fabric has infinite capacity. We demonstrate that to minimize the total average delay when linecards have equal service rates, the load-balancing algorithm should assign *the same processing load* to all linecards.

Second, we model a shared-memory switch fabric with finite switching bandwidth. We model the entire switch fabric as a single queue with a finite service rate, representing the finite rate of memory accesses to the shared memory. Then, we prove the Karush-Kuhn-Tucker (KKT) conditions for optimal total average delay in the router. We also present an efficient algorithm to achieve such an optimal solution, and bound its complexity. We further show that *our load-balancing scheme significantly increases the capacity region of the router*.

Third, we provide the most advanced algorithm, given an output-queued switch fabric model. In this model, each switch-fabric output has its own queue, which packets can enter when the linecard is either their helper (middle) linecard or their egress (destination) linecard. We build on previous results to determine the conditions for minimal delay, and quantify the complexity used to achieve the optimal solution.

In the experimental section, we compare our algorithms to existing load-balancing algorithms using more realistic settings, including OC-192 Internet backbone link traces [9], and an input-queued switch fabric based on VOQs (virtual output queues) with an iSLIP switch-fabric scheduling algorithm [10]. The congestion values of the different linecards are updated at fixed intervals, and packet processing times are based on [7]. We confirm that our algorithms significantly extend the capacity region. In addition, they achieve a lower delay than existing algorithms and than a simple Valiant-based load-balancing algorithm.

Finally, we discuss the different implementation concerns, e.g. buffer management, packet reordering and algorithm overheads. Our suggested schemes are shown to be feasible and able to accommodate various features using only limited modifications, if any, to the existing implementations.

Due to space limits, we outline the complex proofs and algorithms, and fully present them in an online tech. report [11].

C. Related Work

There is a large body of literature on improving the performance of network processors. Most recent works have focused on harnessing the benefits of multi-core architectures for network processing [12]–[18]. However, these approaches are local to the processors, and therefore orthogonal to ours. They can be combined for improved performance.

Other efforts have attempted to optimize the switch fabric architecture, e.g. using Valiant-based load-balanced routers [19]–[21]. These efforts have focused on providing a guaranteed switching rate in the switch fabric, while our goal is to optimize the processing capacity of the linecards. Still, in this paper, we also introduce Valiant-like load-balancing algorithms, and show that they often prove to be sub-optimal.

Building on these load-balanced routers, [22] has presented the GreenRouter, an energy-efficient load-balanced router architecture. Instead of load-balancing traffic to all middle linecards, the GreenRouter only load-balances traffic to a subset of the middle linecards, enabling it to shut down idle processing power at other linecards to save power. Therefore, the GreenRouter always load-balances traffic, while we only need to do so in case of congestion. Also, it tries to concentrate traffic into as few processing units as possible, while our goal is to spread it across all the linecards to increase the capacity region and reduce delay. The architecture is also partly modelled by the sub-optimal Valiant-based algorithm that we introduce in the paper, when all linecards are used. Of course, if needed, the energy-efficient goal could be incorporated into our load-balancing scheme as well.

II. Model and Problem Formulation

A. Linecard Model

We start by introducing notations and formally defining the problem. We consider a router with N linecards connected to a switch fabric. Each linecard is associated with an independent exogenous Poisson arrival process. We denote by γ_i the arrival rate at linecard i, and by $\gamma_{i,j}$ the arrival rate at linecard i of traffic destined to egress j.

To load-balance processing tasks, we assume that each linecard i may choose to redirect a fixed portion $p_{i,j}$ of its incoming traffic to be processed at linecard j. Thus, each incoming packet is independently redirected to linecard j with

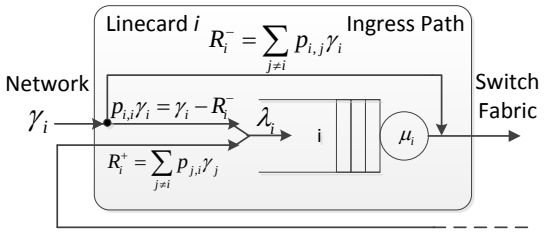


Fig. 2: Notation used to describe the traffic within a linecard.

probability $p_{i,j}$, with $\sum_{j=1}^{N} p_{i,j} = 1$. For instance, if linecard i is not redirecting traffic, then $p_{i,i} = 1$.

As illustrated in Figure 2, we denote by R_i^- the redirected traffic away from linecard i, such that $R_i^- = \sum_{j \neq i} p_{i,j}\gamma_i$. Similarly, we denote by R_i^+ the redirected traffic into linecard i, such that $R_i^+ = \sum_{j \neq i}^{N} p_{j,i}\gamma_j$. Note that a packet can be redirected at most once, immediately upon its arrival.

In addition, we denote by λ_i the total effective arrival rate into the processing queue of linecard i, i.e.

$$\lambda_i = \sum_{j=1}^{N} p_{j,i}\gamma_j = \gamma_i + R_i^+ - R_i^-. \quad (1)$$

Finally, we assume that the linecard processing time of each packet is exponentially distributed with parameter μ_i.

Example 1. *Assume $N = 2$ linecards, with equal service rate $\mu_1 = \mu_2 = 1$. Further assume that the first linecard is congested with an arrival rate of $\gamma_1 = 1.2$ and the second linecard is idle, i.e. $\gamma_2 = 0$. Then the first linecard could redirect half of its incoming traffic to the second linecard, namely $p_{1,2} = 0.5$. Therefore $R_1^- = R_2^+ = 0.6$, and the effective arrival rate into each linecard queue is $\lambda_1 = \lambda_2 = 0.6$.*

B. Switch-Fabric Model

We now propose three different models for the switch-fabric architecture and the resulting switch-fabric congestion.

In the *first model*, we assume that the switch fabric has an infinite switching capacity, such that packets pass through it without delay. Therefore, the load-balancing algorithm will not take switch congestion into consideration. This model provides us with some intuition on the pure load-balancing approach.

As illustrated in Figure 3, the *second model* represents a shared-memory switch fabric that relies on a single shared queue. We model the service time at this shared switch-fabric queue as exponentially distributed with parameter μ^{SF}, representing the switching capacity of the switch fabric. The total arrival rate to this queue is $\lambda^{\text{SF}} = \sum_{j=1}^{N} \gamma_j + \sum_{i=1}^{N} R_i^+$, i.e. the sum of the total incoming traffic rate into the router and the additional rate of redirected packets.

Figure 4 shows the *third model*, which represents an output-queued switch fabric as a set of N queues, each queue being associated with a different switch-fabric output. We denote by μ_i^{SF} the service rate for output queue i. Its arrival rate is

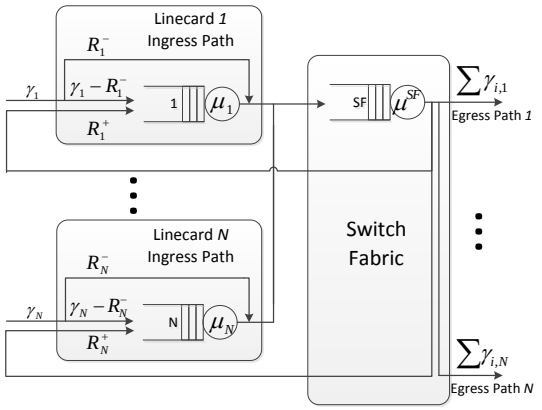


Fig. 3: The single-queue switch-fabric model of a shared-memory router.

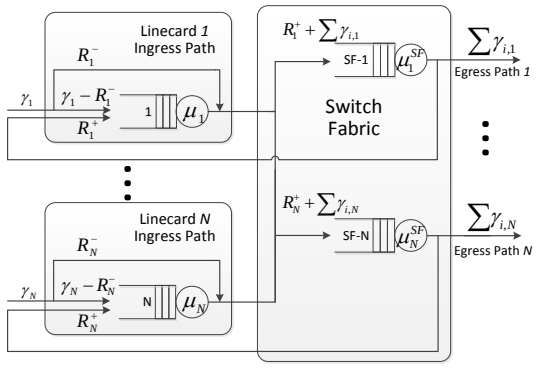


Fig. 4: The N-queues switch fabric model of an output-queued router. The N output queues represent the switch-fabric delay that depends only on the packet destination.

$\lambda_i^{\text{SF}} = \sum_{j=1}^{N} \gamma_{j,i} + R_i^+$, where $\sum_{j=1}^{N} \gamma_{j,i}$ is the total arrival rate destined towards linecard i.

In all three switch-fabric models, the stationary distribution of the queue sizes can be modeled as following a *product-form distribution*. This can be seen by directly applying either Kelly's results on general customer routes in open queueing networks (Corollary 3.4 of [23]), or by applying the BCMP theorem, with each flow being a chain, and each redirected flow being denoted as a new class after its first passage through the switch fabric (Fig. 3 of [24]).

C. Delay Optimization Problem

Given a router with linecard arrival rates $\{\gamma_{i,j}\}$, our goal is to find the fixed redirection probabilities $p_{i,j}$ *that minimize the average packet delay*. Whenever defined, we assume that we are in the steady state. We also assume that all the buffers have infinite size and FIFO policy.

Note that by minimizing the average delay, we also *maximize the capacity region*, the group of feasible arrival vectors with finite expected delay, because outside the feasible region the steady-state average delay is infinite by definition.

III. Infinite-Capacity Switch-Fabric Model

To gain some intuition on the delay optimization problem, we start by studying the first model where the switch-fabric capacity is infinite.

Theorem 1. *In the infinite-capacity switch-fabric model with feasible incoming rates (i.e. $\sum_{i=1}^{N} \gamma_i < N \cdot \mu$), the load-balancing algorithm is optimal for the average delay iff*

$$\forall i,j \in [1,N],\ \frac{\mu_i}{(\mu_i-\lambda_i)^2} = \frac{\mu_j}{(\mu_j-\lambda_j)^2} \tag{2}$$

Proof Outline: We minimize the average delay in the system subject to the condition that the incoming rates are feasible. We obtain a convex optimization function on a convex region, and find the Karush-Kuhn-Tucker (KKT) conditions to be necessary and sufficient. ∎

Moreover, following intuition, if we assume that all the processing rates are equal, i.e. $\forall i \in [1,N] : \mu_i = \mu$, then all the linecards need to have the same incoming traffic rate: $\forall i \in [1,N],\ \lambda_i = \lambda_{\text{avg}} = \frac{\sum_{i=1}^{N} \gamma_i}{N}$.

A possible algorithm to achieve these conditions is quite simple. Linecards with arrival rates above λ_{avg} load-balance their excess arrival rates to helper linecards. To do so, they simply pick each helper linecard proportionally to its capacity to help, i.e. to the amount of traffic that it would need to reach λ_{avg}. Formally, each linecard i sends $R_i^- = \max(0, \gamma_i - \lambda_{\text{avg}})$ and receives $R_i^+ = \max(0, \lambda_{\text{avg}} - \gamma_i)$, using load-balancing probability

$$p_{i,j\neq i} = \frac{R_i^-}{\gamma_i} \cdot \frac{R_j^+}{\sum_{j=1}^{N} R_j^+}.$$

Example 2. *Suppose we have an infinite-capacity switch with arrival rates $\underline{\gamma} = (1,2,2,3.5,4.5,7,8)$, and equal service rates $\mu_i = \mu = 5$. Then any algorithm that achieves $\lambda_i = 4$ for all i is optimal. In particular, the above algorithm yields $\underline{R^-} = (0,0,0,0,0.5,3,4)$ and $\underline{R^+} = (3,2,2,0.5,0,0,0)$.*

IV. Single-Queue Switch-Fabric Model

A. Minimizing the Average Delay

We now want to start taking into account the congestion at the switch fabric when deciding whether to load-balance traffic to reduce processing congestion. As illustrated in Figure 3, we model a shared-memory switch fabric using a single shared queue. Therefore, using the product-form distribution, the average total delay $\overline{D}(R^+,R^-)$ of a packet through the router can be expressed as:

$$\overline{D} = \frac{1}{\sum_{i=1}^{N}\gamma_i}\left[\left(\sum_{i=1}^{N} \overbrace{\frac{\lambda_i}{\mu_i-\lambda_i}}^{\text{Linecard Delay}}\right) + \left(\frac{\overbrace{\sum_{i=1}^{N}\left(\gamma_i+R_i^+\right)}^{\text{Switch Fabric Delay}}}{\mu^{\text{SF}} - \sum_{i=1}^{N}\left(\gamma_i+R_i^+\right)}\right)\right]$$

i.e. as the sum of the average delay through the linecards and through the switch fabric, averaged over all the traffic. The linecard delay is simply the delay through an M/M/1 queue of arrival rate λ_i (from Eq. (1)) and service rate μ_i. Likewise, the switch-fabric delay is simply the delay through an M/M/1 queue of arrival rate $\sum_i \left(\gamma_i + R_i^+\right)$ and service rate μ^{SF}.

As a result, the optimization problem is given by:

$$\begin{aligned}
&\text{minimize } \overline{D}(R^+,R^-)\\
&\text{subject to } \sum_{i=1}^{N} R_i^+ - \sum_{i=1}^{N} R_i^- = 0\\
&\sum_{i=1}^{N}\gamma_i + \sum_{i=1}^{N} R_i^+ - \mu^{\text{SF}} \le 0\\
&\gamma_i + R_i^+ - R_i^- - \mu_i \le 0, i \in [1,N]\\
&-\gamma_i - R_i^+ + R_i^- \le 0, -R_i^+ \le 0, -R_i^- \le 0, i \in [1,N]
\end{aligned}$$

where the first condition expresses flow conservation, the next two conditions signify that the queues are stable, and the last three conditions keep the flows non-negative. Note that formally, we further substitute $\sum_{i=1}^{N} R_i^+$ by $\frac{1}{2}\left(\sum_{i=1}^{N} R_i^+ + \sum_{i=1}^{N} R_i^-\right)$ for symmetry in the expressions of the derivative value, and also replace λ_i in the expression of $\overline{D}(R^+,R^-)$ by the expression in Eq. (1).

We find that in our load-balancing scheme, *an arrival vector $\underline{\gamma}$ is feasible when both the processing and switching loads are feasible*, i.e. (a) the combined service rate of the linecards is greater than the total arrival rate, and (b) the switch-fabric service rate is greater than the total arrival rate combined with the second pass of redirected traffic. Formally,

$$\begin{cases} \sum_{k=1}^{N}\gamma_k < \sum_{k=1}^{N}\mu_k \\ \sum_{k=1}^{N}\min(\gamma_k,\mu_k) + 2\cdot\sum_{k=1}^{N}\max(\gamma_k-\mu_k,0) < \mu^{\text{SF}} \end{cases}$$

We obtain the following result for an optimal load-balancing:

Theorem 2. *A solution to the average delay minimization problem for the single-queue switch fabric model is optimal if and only if the arrival rate vector is feasible and there exists a constant τ_0 such that for each linecard j, if $R_j^+ > 0$:*

$$\frac{\mu_j}{\left(\mu_j-\gamma_j-R_j^++R_j^-\right)^2} = -\tau_0 \sum_{i=1}^{N}\gamma_i - \frac{1}{2}\frac{\mu^{SF}}{\left(\mu^{SF} - \sum_{i=1}^{N}\gamma_i - \frac{1}{2}\left(\sum_{i=1}^{N} R_i^+ + \sum_{i=1}^{N} R_i^-\right)\right)^2} \tag{3}$$

If $R_j^- > 0$:

$$\frac{\mu_j}{\left(\mu_j-\gamma_j-R_j^++R_j^-\right)^2} = -\tau_0 \sum_{i=1}^{N}\gamma_i + \frac{1}{2}\frac{\mu^{SF}}{\left(\mu^{SF} - \sum_{i=1}^{N}\gamma_i - \frac{1}{2}\left(\sum_{i=1}^{N} R_i^+ + \sum_{i=1}^{N} R_i^-\right)\right)^2} \tag{4}$$

Proof Outline: Once again, the KKT conditions are the first-order necessary conditions, which in our case of convex function and region are also sufficient conditions for a solution to be optimal. ∎

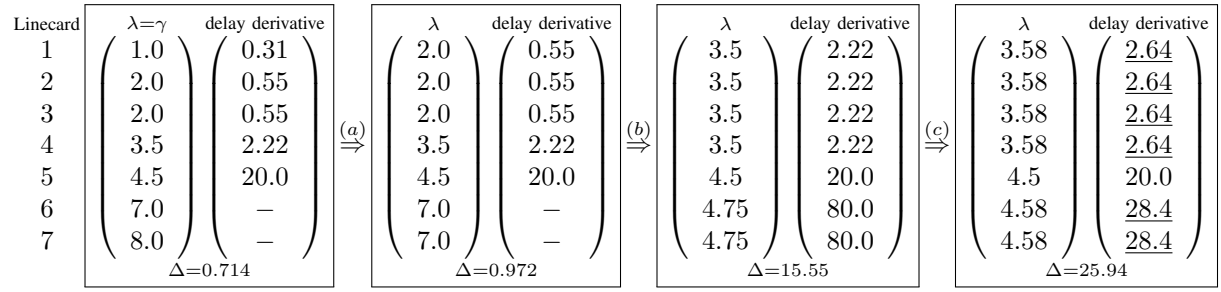


Fig. 5: Successive steps of the algorithm in the single-queue switch-fabric model, based on Example 3. At first, each linecard receives some traffic, and assumes there is no load-balancing. It also computes its resulting processing delay derivative. Then, in step (a), linecard 7 (with an infinite derivative and the highest amount of traffic) load-balances one unit of flow to linecard 1 (which has the lowest derivative). Next, in step (b), saturated linecards 6 and 7 send 4.5 units of flow to linecards 1, 2 and 3. Finally, in step (c), all derivatives are finite. Linecards 6 and 7 send 0.34 additional units of flow to linecards 1, 2 3 and 4. The amount in the final step (c) is found using a binary search, other steps use a simple calculation on the derivatives. The algorithm stops, since the difference between the heavy and light derivatives is Δ, the switch fabric delay derivative value, i.e. *reducing the processing congestion will already deteriorate too much the switch congestion.* Also, for this reason, linecard 5 does not participate in the load-balancing.

Intuitively, we define the (weighted) delay derivative of a linecard i as $\frac{\mu_i}{(\mu_i-\lambda_i)^2}$. Then Theorem 2 states that *the delay derivative of all linecards that receive traffic must equal the same value in Eq. (3)*. Similarly, the delay derivative value of redirecting linecards with excess traffic must equal the same value in Eq. (4). The difference between Eq. (3) and Eq. (4) is the delay derivative of the switch fabric. This is expected because the switch fabric acts as a penalty for redirecting traffic. Also, as expected, $R_j^+ > 0$ and $R_j^- > 0$ cannot both be true for a specific linecard j.

Moreover, an interesting result is that *it is possible for a linecard not to participate in the load-balancing*. Specifically, if for a linecard j, $\frac{\mu_j}{(\mu_j-\gamma_j)^2}$ is between the two values in Eq. (3) and Eq. (4), then $R_j^+ = 0$ and $R_j^- = 0$. Intuitively, the linecard is not congested enough to send traffic and congest the switch fabric, but also not idle enough to gain from receiving traffic that would further congest the switch fabric.

The algorithm needed to achieve the KKT optimality conditions is relatively simple. It orders all linecards by their delay derivative, and *progressively sends more flow from the linecard(s) with the highest delay derivative to the linecard(s) with the lowest*, until it achieves the conditions specified in Theorem 2. Let's provide an intuitive example to show how it works, as further illustrated in Figure 5.

Example 3. *Consider the same arrival rates as in Example 2, together with a finite-capacity switch fabric of switching rate equal to the sum of the linecard processing rates, i.e.* $\mu^{SF} = 5 \cdot 7$.

As shown in Figure 5, the algorithm progressively decides to load-balance more and more flows. We can see how at the end of the algorithm, the resulting arrival rates into the linecard queues are not all equal to λ_{avg}*, as they were in Example 2. It is because the switch fabric is limiting the amount of redirected traffic. This illustrates the tradeoff between switch-fabric congestion and linecard processing congestion. In fact, linecard* 5 *ends up not participating in the load-balancing.*

The following result shows that the algorithm complexity is relatively low.

Theorem 3. *The complexity of reaching the optimal solution within an error bound of* ε *is* $\mathrm{O}(N + \log_2(\frac{1}{\varepsilon} \cdot \sum_{i\in[1,N]} \mu_i))$.

Proof Outline: At each step, at least one linecard joins either the maximal or minimal delay derivative group, and there are at most $N-2$ such steps. The algorithm halts at the iteration that causes the distance between the derivative value of the two groups to fall below the switch-fabric derivative value. Then we run a root-finding algorithm to find the optimal redirected flow size, with a logarithmic complexity on the maximal possible size of redirected flow divided by the desired accuracy. ■

B. Capacity Region.

The capacity region of each linecard without load-balancing is bounded by its own processing capability, regardless of the traffic in other linecards. Using the load-balancing scheme, we can expand the capacity region based upon the *total available processing power* of the router.

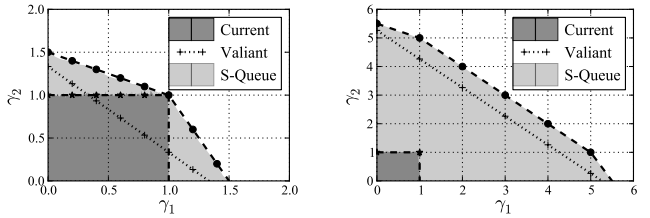
We define the capacity region Γ as the set of all non-negative arrival rate vectors $\{\gamma_i\}$ that are feasible (as in [25], [26]). We want to compare the capacity region of our algorithm against that of a basic *Current* scheme without load-balancing, and of a *Valiant-based* load-balancing scheme that we now introduce (based on [27]).

Definition 1 (Valiant-based load-balancing)**.** *Let the Valiant-based algorithm be defined as an oblivious load-balancing scheme where each linecard* i *automatically load-balances a uniform fraction* $\frac{1}{N}$ *of its incoming traffic to be processed at each linecard* j *(and processes locally* $\frac{1}{N}$ *as well), regardless of the current load in the different linecards.*

Theorem 4. *The capacity region using the different algorithms is as follows:*

i) Current*:* $\forall i : \gamma_i < \mu_i$*, and* $\sum_i \gamma_i < \mu^{SF}$.

ii) Valiant*:* $\forall i : \frac{\sum_i \gamma_i}{N} < \mu_i$*, and* $\frac{2N-1}{N} \cdot \sum_i \gamma_i < \mu^{SF}$.



(a) Capacity region with $N = 2$ linecards. (b) Capacity region with $N = 10$, only 2 linecards have incoming traffic.

Fig. 6: Single-Queue switch fabric model with $\mu^{\text{SF}} = N$. The colored fills represent the simulation results for the current scheme without load-balancing (dark grey) and our Single-Queue delay-optimal scheme (light grey). The plotted borders represent the theoretical values for the two schemes and for the Valiant-based scheme. We can see how our scheme achieves a larger capacity region in both cases, since the Current scheme does not do load-balancing, and the Valiant scheme performs oblivious and sometimes harmful load-balancing.

iii) Single Queue *(ours):* $\sum_i \gamma_i < \sum_i \mu_i$, *and*

$$\overbrace{\sum_i \min(\gamma_i, \mu_i)}^{\textit{processed locally}} + 2 \cdot \overbrace{\sum_i \max(\gamma_i - \mu_i, 0)}^{\textit{redirected traffic}} < \mu^{SF}.$$

Proof: (i) The result for the current implementation is straightforward, since it cannot redirect traffic.
(ii) This derives from the fact that Valiant load-balancing uniformly redirects $\frac{N-1}{N}$ of the arrival rate of each linecard to all other linecards.
(iii) Finally, in our scheme, the first equation derives from the processing multiplexing, and the second equation states that the switch-fabric arrival rate is smaller than its service rate. ■

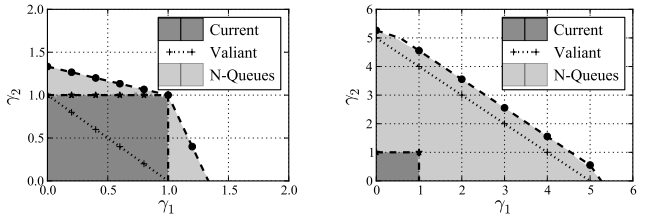
Figure 6 illustrates the theorem results. It plots the capacity region when either $N = 2$ or $N = 10$, using a uniform service rate $\mu_i = 1$, $\mu^{\text{SF}} = N$, and assuming that only two linecards have arrivals. We can see how when one of the linecards is idle, the feasible processing rate of the other linecard can significantly increase using our load-balancing scheme. The two other schemes appear sub-optimal, since the Current scheme does not do load-balancing, and the Valiant scheme relies on an oblivious and sometimes harmful load-balancing. The performance gain with respect to Current is especially large when there is dissymmetry between linecard rates, and against Valiant when there is symmetry and load-balancing becomes harmful.

V. N-Queues Switch Fabric Model

We now consider a more accurate switch-fabric model, where the congestion in the switch fabric depends on the destination. As illustrated in Fig. 4, we model an output-queued switch fabric by assigning a queue at each output. This helps us capture output hot-spot scenarios.

A. Minimizing the Average Delay

As in the previous sections, we want to minimize the average delay of packets in the router, which equals the sum of the delays through all queues, weighted by the flows going through these queues, and normalized by the total incoming exogenous flow. We obtain a similar result on the average-delay minimization problem.



(a) $N = 2$. (b) $N = 10$, only $\gamma_1, \gamma_2 \neq 0$.

Fig. 7: Capacity region using the N-Queues SF model, $\mu_i^{\text{SF}} = 1$. The plotted borders are the theoretical values and the colored fill is the simulation result.

Theorem 5. *A solution to the average-delay minimization problem for the N-queues switch fabric model is optimal iff (i) the flows are feasible and (ii) there exists a constant τ_0 such that for each linecard j, if $R_j^+ > 0$:*

$$\frac{\mu_j}{(\mu_j - \gamma_j - R_j^+ + R_j^-)^2} + \frac{\mu_j^{SF}}{\left(\mu_j^{SF} - \sum_{i=1}^{N} \gamma_{i,j} - R_j^+\right)^2} = -\tau_0 \sum_{i=1}^{N} \gamma_i;$$

and if $R_j^- > 0$: $\frac{\mu_j}{(\mu_j - \gamma_j - R_j^+ + R_j^-)^2} = -\tau_0 \sum_{i=1}^{N} \gamma_i.$

Proof Outline: The proof is very similar to the one in the previous section. Again, the KKT conditions are shown to be necessary and sufficient. ■

From the optimization conditions, it is easy to deduce an algorithm that works quite similarly to the algorithm in the previous section. The algorithm solves the optimization problem by performing a binary search on the possible values of R_k^+, where k is the linecard with the smallest combined (linecard and switch fabric) delay derivative. At each step of the binary search, all the linecards with a lesser (respectively, greater) combined derivative are considered to belong to the minimal (maximal) derivative group, and have traffic redirected to them (away from them) until their combined derivative value equals that of linecard k.

Theorem 6. *The complexity of reaching the optimal solution within an error bound of ε is* $\mathrm{O}(N \cdot \log_2(\frac{\mu}{\varepsilon})^2)$.

Proof Outline: We apply the above algorithm, and continuously compare the total redirected and received traffic. We continue the binary search accordingly until we reach the desired accuracy. ■

B. Capacity Region.

Similarly to the single-queue model, we again plot the capacity region. We assume the same settings, and also assume that the packet destinations are uniformly distributed and that the output queue services rates are equal to $\mu_i^{\text{SF}} = 1$. As illustrated in Figure 7, the results are largely similar, although

the two load-balancing algorithms now slightly suffer from the increased switch fabric congestion, since the service rate of the switch fabric is now split between N queues.

VI. Simulation Results

In this section we compare the performance of our suggested algorithms to the existing algorithms. We first simulate the N-*Queues* switch-fabric queuing architecture using synthetic traces. Afterwards, we simulate an iSLIP input-queued switch-fabric scheduling algorithm with traces from high-speed Internet backbone links.

To clarify, we compare all algorithms *on the same architecture*, but each algorithm determines its load-balancing policy based on its own switch-fabric model. Thus, the algorithm model is not necessarily identical to the simulated architecture.

A. N-Queues Switch Fabric Algorithm

Figures 8a, 8b and 8c simulate all algorithms on the output-queued switch-fabric architectural model that appears in Figure 4. The figures plot the performance of five algorithms: (i) the baseline Current algorithm without load-balancing, (ii) the Valiant-based algorithm, (iii) our load-balancing algorithm that assumes an infinite switch-fabric service rate, (iv) our single-queue switch-fabric algorithm, and (v) our N-queues switch-fabric algorithm, which is expected to outperform since its model is closest to the architecture. The service rates are $\forall i \in [1, N]$, $\mu_i = \mu_i^{\text{SF}} = 1$. The destination linecard is chosen uniformly.

Figure 8a confirms the capacity region in Figure 7b, where two linecards have positive incoming arrival rates and the other linecards are idle. As expected, the Current algorithm cannot accommodate an arrival rate that is larger than the service rate of a single linecard. Furthermore, as in Figure 7b, the capacity region for the Valiant-based load-balancing is capped at $\gamma_{1,2} = 2.5$, and for our algorithm at $\gamma_{1,2} = 2.8$.

Figure 8b shows the delays when five out of ten linecards have an equal arrival rate and the rest are idle. The Valiant-based load-balancing performs significantly worse due to the fact that now $\frac{4}{N} = 40\%$ of all the redirections it performs are redundant and towards active linecards, whereas with two active linecards only 10% of the redirections were redundant. An interesting result is at the low arrival rates, where the Valiant-based algorithm, and to a lesser degree our Infinite algorithm, perform poorly and have a larger average delay than without load-balancing. The reason is that the switch fabric service rate is split between $N = 10$ queues, therefore each packet experiences a larger delay when passing through the switch fabric and these algorithms have many redundant redirections at low arrival rates. On the other hand, our optimal algorithm decides at these low arrival rates not to redirect.

In Figure 8c the arrival rates are assumed to be distributed linearly at equal intervals. For instance, when the x-axis is at 1.8, the arrival rates to the linecards are $(0.0, 0.2, 0.4, \ldots, 1.8)$. The result appears similar.

Incidentally, note that while our *Infinite* algorithm performs slightly worse than our more accurate algorithms, it may be interesting when complexity is an issue.

B. Real Traces

We now simulate a real input-queued switch fabric with VOQs and an implementation of the iSLIP [10] scheduling algorithm. Upon arrival at the switch fabric, each packet is divided into evenly-sized cells and sent to a queue based on its source and destination linecards. At each scheduling step, the iSLIP scheduling algorithm matches sources to destinations, and the switch fabric transmits cells based on these pairings. The destination linecard must receive all of the packet's cells before the packet can continue.

For this simulation we rely on real traces from [9], which were recorded at two different monitors on high-speed Internet backbone links. One of the traces has an average OC192 line utilization of 24%, and the other 3.2%.

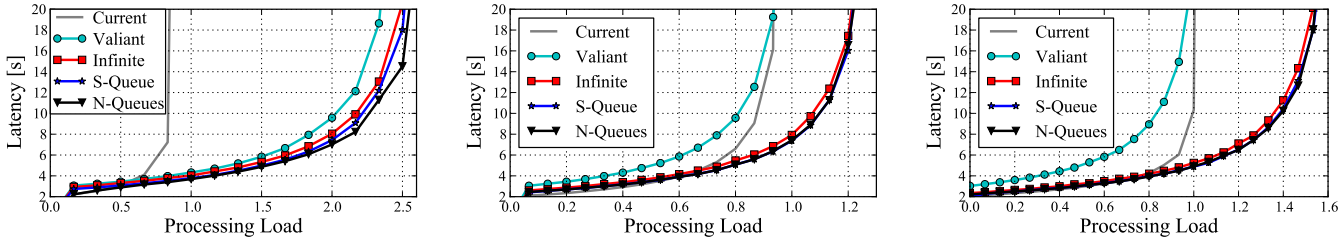
Note that in these simulations, all the arrival rates into the linecards are positive, while in reality many router linecards are simply disconnected. Thus, *in real-life, we would expect load-balancing schemes to be even more beneficial*.

The switch fabric rate is set to be equivalent to OC192 per matching. The processing time is calculated based on the packet size using the number of instructions and memory-accesses estimate from [7] for header and payload processing applications. Memory access times are assumed to be constant at 4ns.

The algorithms in previous sections require several adjustments to accommodate the more realistic features of this simulation. The average arrival rates at each linecard are no longer known in advance, and no longer constant. Instead, the algorithm performs periodic updates. At each interval the algorithm measures the arrival rates to all linecards, and based on previous intervals it predicts the arrival rates in the next interval and calculates the required load-balancing policy based on this prediction. In this section the updates were made at thousand-packet intervals.

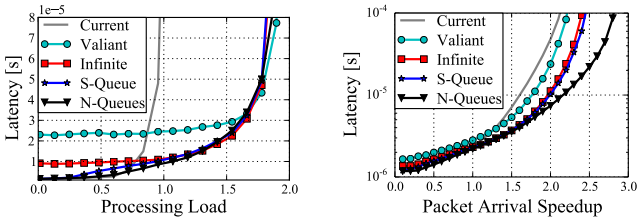
Figure 9a shows a comparison between the current implementation, Valiant-based load-balancing, and our algorithms. The packet arrival times were divided by two to simulate a more congested network with 48% and 6.4% OC192 line utilization. There are N linecards in the system, five of which are connected to the relatively highly utilized line rates and the remaining linecards are connected to the lower arrival rate links. The x-axis shows *the average processing load of the most congested line card*. The change in the x-axis only affects the service rate of the line cards while the switch fabric rate and the arrival rates stay constant.

The Valiant-based delay is high, even at low loads. This is due to the significant additional congestion at the switch fabric. This delay is somewhat constant up to very high loads, where the delay begins to be mostly influenced by the congested linecards. Our algorithm does not redirect traffic at low loads and maintains a delay similar to the delay without load-balancing. However, for a small range around a processing load of 0.6, it has a slightly higher delay, because it redirects traffic earlier than it should due to inaccurate arrival rate estimation. Significantly, our algorithm almost doubles the capacity region of the most congested linecard in the system



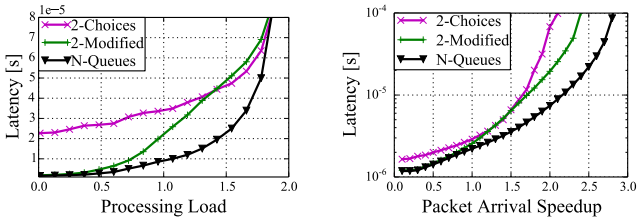
(a) Two linecards out of ten have incoming traffic. (b) Five linecards out of ten have incoming traffic. (c) Arrival rates are distributed between 0 and x-axis.

Fig. 8: N-Queues switch fabric model, with $N = 10$ and a Poisson arrival model.



(a) Constant arrival rate, variable packet processing load. (b) Variable arrival rate, constant packet processing load.

Fig. 9: Simulation with OC-192 Internet backbone link traces [9], iSLIP switch fabric scheduling algorithm [10] and packet processing times based on [7], comparing our three algorithms with the Current and Valiant algorithms.



(a) Constant arrival rate, variable packet processing load. (b) Variable arrival rate, constant packet processing load.

Fig. 10: Simulation with same settings as Figure 9, comparing the performance of our best algorithm with queue-length-based algorithms.

when compared to the Current algorithm. Of course, load-balancing algorithms with less accurate models perform with slightly higher delays.

Figure 9b shows the same comparison, but now *the service rates of the linecards are constant while the arrival rates are variable*. The N-queues algorithm performs significantly better than the other algorithms in this scenario because it takes into consideration the different loads at the switch fabric queues. As the arrival rate increases, the switch fabric delay grows along with the processing delay in the linecards.

VII. Discussion

A. Queue-Length-Based Algorithms

Our algorithms balance the traffic load *based on the arrival rates to the linecards*. A different approach would be to load-balance *based on the queue lengths*. For instance, if the switch-fabric capacity were infinite, as in our first switch architecture, and the queue lengths were known at all times, then a simple redirect to the shortest queue approach would be *optimal* [28]. However, such global and constantly updated knowledge would of course be costly to maintain across all the linecards, and therefore updates regarding the current queue lengths would be sent periodically, similarly to the updates of the arrival rates.

Based on the *power of two choices* [29], we introduce two simple algorithms. First, in the *2-Choices* algorithm, each packet compares the queue sizes at two random linecards and is redirected to the shortest one. Second, in the *2-Modified* algorithm, each incoming packet only compares the queue sizes of its ingress linecard and of another random linecard, and chooses the linecard with the smallest queue.

Figures 10a and 10b compare the different algorithms. Figure 10a uses a constant arrival rate and variable service rates at the linecards, while Figure 10b assumes a variable arrival rate and constant service rates at the linecards. The 2-Modified algorithm performs better than 2-Choices at low processing loads, since it needs less redundant redirects. In any case, our N-Queues algorithm seems to outperform both.

B. Implementation

We initially had several significant *implementation concerns* regarding our algorithms, in particular regarding (a) *buffer management* and (b) *reordering*. However, these concerns were alleviated following discussions with industry vendors.

First, our algorithms may cause problems if redirecting traffic requires to first *reserve the buffer* at the helper linecard. But this is not significantly different from reserving the buffer at the output, and therefore seems reasonable. The algorithms could also *bypass the buffer* at the input linecard when redirecting, unlike what is currently done. But this is simply about updating the implementation, and does not appear to cause any fundamental issues.

A second potential concern is that our algorithms may cause *reordering*, thus damaging the performance of TCP flows. But there are many papers and vendor implementations that address this concern [22], [30], [31]. For instance, it is possible to change the granularity of the load-balancing from a per-packet decision to a per-flow decision. In fact, we reran our simulations based on OC192 traces, with over 10^7 different flows appearing at each second in each linecard, and found

that this change did not influence the performance (within an approximation of 10^{-5}). This is expected because the granularity of the load-balancing with this number of flows is fine enough for any practical use. An alternative technique would be to maintain reordering buffers at the egress paths, although this would of course require additional memory [31].

More minor concerns are about the *algorithm overhead*. Our algorithms need to periodically collect information about the various linecards and in addition incur computational complexity and memory overheads. However, in the worst case every linecard transmits an update to every other linecard. Assuming $N = 64$ linecards, 1,000 updates per second and 32 bits representing the flow size, yields a total overhead of 131Mbps, which is negligible in routers with total rates of over 100 Gbps. In addition, our algorithms incur other overheads, including complexity and memory overheads. The complexity overhead of our most complex algorithm is roughly of the same order of magnitude as running a typical payload processing application on several packets based on [7]. This computation is done every update interval, i.e. every 10,000 packets over the whole switch in most of our simulations, and therefore the amortized per-packet additional computation is less than 0.1%. Also, the required memory per linecard is at most $2N \cdot 32 = 3,200$ bits to represent the redirection probabilities and congestion values per linecard. Therefore, we find that these overheads appear quite reasonable.

VIII. Conclusion

In this paper, we have introduced a novel technique for tapping into the processing power of underutilized linecards. Given several switch-fabric models, we introduced load-balancing algorithms that were shown to be delay-optimal and throughput-optimal. We also illustrated their strong performance in simulations using realistic switch architectures.

Of course, there remain many interesting directions for future work. In particular, we are interested in exploring the possibility for router vendors of adding helper linecards with no incoming traffic, with the unique goal of assisting existing linecards to process incoming traffic. In addition, this idea may be merged with the recent *NFV (Network Function Virtualization) proposals*: such helper linecards may in fact contain multi-core processors that would also be devoted to additional programmable tasks, thus avoiding the need to send these tasks to remote servers.

IX. Acknowledgment

The authors would like to thank Shay Vargaftik, Aran Bergman, Rami Atar and Amir Rosen for their helpful comments. This work was partly supported by the Mel Berlin Fellowship, the Gordon Fund for Systems Engineering, the Neptune Magnet Consortium, the Israel Ministry of Science and Technology, the Intel ICRI-CI Center, the Hasso Plattner Institute Research School, the Technion Funds for Security Research, and the Erteschik and Greenberg Research Funds.

References

[1] EZchip. (2013) EZchip Technologies — NPS Family. [Online]. Available: http://www.ezchip.com/p_nps_family.htm

[2] Cisco. (2013) Cisco 4451-x integrated services router data sheet. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/routers/ps10906/ps12522/ps12626/data_sheet_c78-728190.html

[3] R. Atkinson and S. Kent, "RFC 2406 – IP encapsulating security payload (ESP)," 1998.

[4] M. Roesch, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, 1999, pp. 229–238.

[5] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection." in *USENIX security symposium*, 2004.

[6] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting." in *OSDI*, vol. 4, 2004, pp. 4–4.

[7] R. Ramaswamy, N. Weng, and T. Wolf, "Analysis of network processing workloads," *Journal of Systems Architecture*, vol. 55, no. 10-12, 2009.

[8] F. Abel *et al.*, "Design issues in next-generation merchant switch fabrics," *IEEE/ACM Trans. Netw.*, vol. 15, no. 6, pp. 1603–1615, 2007.

[9] The CAIDA UCSD anonymized Internet traces 2011 - 20110217-130100-130500. [Online]. Available: http://www.caida.org/data/passive/passive_2011_dataset.xml

[10] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, 1999.

[11] M. Radan and I. Keslassy, "Tapping into the router's unutilized processing power," Technion, Israel, Technical Report TR15-01, 2015. [Online]. Available: http://webee.technion.ac.il/~isaac/papers.html

[12] R. Ennals, R. Sharp, and A. Mycroft, "Task partitioning for multi-core network processors," in *Compiler Construction*, 2005.

[13] Y. Qi *et al.*, "Towards high-performance flow-level packet processing on multi-core network processors," in *ACM/IEEE ANCS*, 2007.

[14] R. Sommer, V. Paxson, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," *Concurrency and Computation*, vol. 21, no. 10, pp. 1255–1279, 2009.

[15] I. Keslassy, K. Kogan, G. Scalosub, and M. Segal, "Providing performance guarantees in multipass network processors," *IEEE/ACM Trans. Netw.*, vol. 20, no. 6, pp. 1895–1909, Dec. 2012.

[16] Y. Afek *et al.*, "MCA2: multi-core architecture for mitigating complexity attacks," *ACM/IEEE ANCS*, pp. 235–246, 2012.

[17] O. Rottenstreich, I. Keslassy, Y. Revah, and A. Kadosh, "Minimizing delay in shared pipelines," *IEEE Hot Interconnects*, pp. 9–16, 2013.

[18] A. Shpiner, I. Keslassy, and R. Cohen, "Scaling multi-core network processors without the reordering bottleneck," *IEEE HPSR*, pp. 146–153, Jul. 2014.

[19] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load balanced Birkhoff–von Neumann switches," *Computer Communications*, 2002.

[20] I. Keslassy, "The load-balanced router," Ph.D. dissertation, Stanford University, 2004.

[21] B. Lin and I. Keslassy, "The concurrent matching switch architecture," *IEEE/ACM Trans. Netw.*, vol. 18, no. 4, pp. 1330–1343, 2010.

[22] Y. Kai, Y. Wang, and B. Liu, "GreenRouter: Reducing power by innovating router's architecture," *IEEE CAL*, 2013.

[23] F. P. Kelly, *Reversibility and stochastic networks*. Cambridge University Press, 2011.

[24] S. Balsamo, "Product form queueing networks," *Performance Evaluation*, pp. 377–401, 2000.

[25] M. J. Neely, "Delay-based network utility maximization," *IEEE/ACM Trans. Netw.*, vol. 21, no. 1, pp. 41–54, 2013.

[26] ——, "Stability and capacity regions for discrete time queueing networks," *arXiv preprint arXiv:1003.3396*, 2010.

[27] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *ACM Symp. on Theory of Computing*, 1981.

[28] W. Winston, "Optimality of the shortest line discipline," *Journal of Applied Probability*, pp. 181–189, 1977.

[29] M. Mitzenmacher, "How useful is old information?" *IEEE TPDS*, vol. 11, no. 1, pp. 6–20, 2000.

[30] C. Hu, Y. Tang, X. Chen, and B. Liu, "Per-flow queueing by dynamic queue sharing," in *IEEE Infocom*, 2007, pp. 1613–1621.

[31] O. Rottenstreich *et al.*, "The switch reordering contagion: Preventing a few late packets from ruining the whole party," *IEEE Trans. Comput.*, vol. 63, no. 5, 2014.