How to waste $2/3$ of the throughput of a switch: a tight characterization of load balancing algorithms that do not split.

Saad Mneimneh
Visiting Professor, Computer Science, Hunter College of CUNY,
695 Park Avenue, New York, NY 10021
Email: saad@alum.mit.edu, Tel: (214)707-1075, Fax: (212)772-5219

Dedicated to the memory of a thought in September 2002

Abstract— **Load balancing has recently acquired increased interest among researchers in the switching community. The premise is to replace the single switch, running at a speed proportional to the number n of input and output ports (and to the line rate), by a load balanced switch consisting of k switches, each running at $1/k$ the speed. Indeed, with such an architecture, and for some classes of traffic patterns, simply spreading the traffic among the k switches achieves 100% throughput (thus the term load balancing). However, reordering among packets of the same flow may occur, which becomes a major concern. One way of avoiding this problem is to use a load balancing algorithm that does not split, i.e. that routes all packets of a given flow through exactly one of the k switches, without re-routing existing flows. We shall call such a load balancing algorithm a non-splitting algorithm. While non-splitting algorithms definitely preserve the order of packets, it is intuitively well understood that these algorithms waste throughput. The focus of this paper is to exactly characterize this waste. We prove an asymptotic tight upper bound of $1/3$ on the throughput of any non-splitting algorithm when $n >> k$.**

Keywords: Load balanced switch, load balancing algorithm, non-splitting algorithm, throughput, upper bound.

I. Introduction

Load balancing has recently acquired an increased interest among researchers in the switching community [14]. The premise is to replace the single switch, running at a speed proportional to the number n of input and output ports (and to the line rate), by a load balanced switch consisting of k switches, each running at $1/k$ the speed. A load balanced switch is shown in Figure 1.

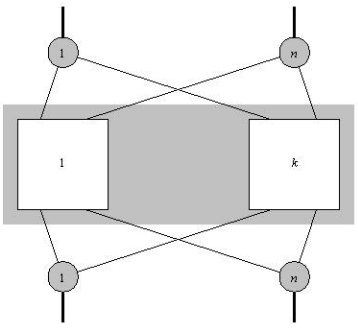

Fig. 1. Load balanced switch with unit capacity links and a line rate of k

The load balanced switch appeared repeatedly in literature over the past few years, e.g. [11], [7], [2], and [10]. In particular, the work in [10] provides a comprehensive overview of the load balanced switch and a number of useful contributions. It is the work of [2] that first identified and emphasized the load balancing aspect of such an architecture: Each input spreads its traffic among the k switches using a load balancing algorithm. Each output then assembles its traffic from the k switches. Indeed, for some classes of traffic pattern, it is possible to find simple load balancing algorithms that achieve 100% throughput [2].

However, reordering among packets of the same flow may occur, which becomes a major concern. Several solutions have been proposed to cope with reordering. Most of these solutions use buffers at the input ports to limit the amount of reordering in the k switches, and therefore, use buffers at the output ports also to deliver packets in order, e.g. [7], [3], [10], and [12]. Variations on this scheme, such as [4] and [9], exist.

Yet another way of making use of input buffers to avoid reordering is a load balancing algorithm that does not split, i.e. that routes all the packets of a given flow through exactly one of the k switches, and does not re-route existing flows (hence packets competing for a given switch wait at the input). Such algorithms (usually based on hashing schemes) are common in load balancing systems, e.g. [1], [13], [5], [8], [11]. We shall call them non-splitting algorithms. While non-splitting algorithms definitely preserve the order of packets, it is intuitively well understood that these algorithms waste throughput: they may result in many remaining small capacities scattered throughout the switches, and yet none of them is enough by itself to route a single flow. Although these algorithms may not be the ideal choice (for the obvious reason), the focus of this paper is to exactly characterize how much waste of throughput these algorithms can exhibit. We prove an asymptotic tight upper bound of $1/3$ on the throughput of any non-splitting algorithm when $n >> k$. Therefore, we show that $2/3$ of the throughput is wasted as a cost of preserving order in this particular way.

The paper is organized as follows. Section II defines non-splitting algorithms and the notion of throughput to be used. Section III shows that is it possible to achieve a throughput greater than $1/3$ with a simple non-splitting algorithm, pro-

1-4244-1206-4/07/$25.00 ©2007 IEEE

vided that the rates of flows are known. Section IV shows that a non-splitting algorithm cannot achieve a throughput greater than $1/3$ asymptotically when $n >> k$. We finally conclude in Section V.

II. Non-splitting load balancing algorithms

A non-splitting algorithm is an *online* load balancing algorithm that routes all packets of a given flow through exactly one of the k switches. Moreover, it does not re-route existing flows (otherwise the online property becomes void algorithmically, and more importantly, reordering may occur). The choice of an online algorithm is motivated by the fact that in general a load balancing algorithm has no knowledge of future flows.

Since a non-splitting algorithm routes flows instead of individual packets, we assume that each flow has a rate. We assume further that this rate is known to the algorithm when the first packet of the flow arrives [1]. This assumption is needed for any non-splitting algorithm to make reasonable routing decisions in the absence of any knowledge about the flows, and it only strengthens the negative result of Section IV. Therefore, we may assume without loss of generality (WLOG) that:

- every flow f has a known rate r_f
- once the first packet of a flow arrives, the flow is routed through one of the k switches
- each link has a capacity of 1

We define $F_{i\rightarrow l}$ be the set of flows originating at input i and routed through switch l. Similarly, we define $F_{j\leftarrow l}$ to be the set of flows destined to output j and routed through the switch l.

Definition 1 (Throughput): Let

$$\alpha = \max(\max_{i,l} \sum_{f\in F_{i\rightarrow l}} r_f, \max_{j,l} \sum_{f\in F_{j\leftarrow l}} r_f)$$

Then the throughput of a non-splitting load balancing algorithm is $z = \min(1/\alpha, 1)$.

Note that $\sum_{f\in F_{i\rightarrow l}} r_f$ is the total rate of flows using the link between input i and switch l. Similarly, $\sum_{f\in F_{j\leftarrow l}} r_f$ is the total rate of flows using the link between switch l and output j. As a result, α represents the largest load on any link. For instance, if $z = 1$, then $1/\alpha \geq 1$. This means that $\alpha \leq 1$ and no link capacity is exceeded. If $z < 1$, then $z = 1/\alpha$ and $\alpha > 1$. Consequently, at least one flow must have its rate multiplied by z (or less) to avoid the excessive load. The choice of such flow(s) may be determined at a level higher than the switch (e.g. transport protocol). For instance, TCP fairness guarantees that all flows sharing a bottleneck will have their rates multiplied by z (theoretically). Therefore, the above definition provides a reasonable notion of throughput. Besides, we would like to avoid definitions that make it possible to achieve a throughput arbitrarily close to 1 but completely starve a flow (which does not usually happen when splitting).

[1]While this assumption may be restrictive, we use the existence of such an algorithm to prove that the bound of Section IV is tight.

Define $F_{i\rightarrow} = \bigcup_{l=1}^{k} F_{i\rightarrow l}$ and $F_{j\leftarrow} = \bigcup_{l=1}^{k} F_{j\leftarrow l}$. The condition $\sum_{f\in F_{i\rightarrow}} r_f \leq k$ and $\sum_{f\in F_{j\leftarrow}} r_f \leq k$ guarantees that, in principle, each switch can run at $1/k$ the speed (the total rate at the ports is at most nk and the total capacity of each switch is n). Without this condition, throughput may be arbitrarily close to zero, even if $k = 1$ (simply flood the switch with flows). Therefore, we would (uninterestingly) claim that the throughput of non-splitting algorithms is trivially $z = 0$. Moreover, consider one flow f with rate $r_f = k$. This flow has to be routed through one of the k switches, which makes $\alpha \geq k$. Therefore, the throughput $z \leq 1/k$. If k is large enough, z is arbitrarily close to 0, which again makes the setting unrealistic. The condition that for each flow f, $r_f \leq 1$, avoids this triviality, and guarantees that, in principle, each flow can be routed without being split (since the link capacity is 1). We bundle the above conditions into one condition that we call the no overload condition, and we shall assume thereafter that it is enforced.

Definition 2 (No overload condition): Given a flow f_0 from input i to output j that is not yet routed, $r_{f_0} \leq 1$. Moreover, $r_{f_0} + \sum_{f\in F_{i\rightarrow}} r_f \leq k$ and $r_{f_0} + \sum_{f\in F_{j\leftarrow}} r_f \leq k$.

We now present some examples to gain intuition on how a non-splitting algorithm can waste throughput. Consider the example where $k = 2$ and $n = 1$. We identify the input and the output by i and j, respectively. Consider three flows from i to j with the following rates $1/2$, $1/2$, and 1. Obviously, the no overload condition cannot be violated. Now say that the non-splitting algorithm routes the first flow of rate $1/2$ through switch 1, then routes the second flow of rate $1/2$ through switch 2. The third flow can now be routed through either switch 1 or switch 2, resulting in $\alpha = 3/2$ and a throughput $z = 2/3$.

One would definitely think that a non-splitting algorithm with *some intelligence* will pack the first two flows into a single switch. Hence routing the first two flows through switch 1, and the third flow through switch 2, resulting in $\alpha = 1$ and a throughput $z = 1$. However, as the following example shows, such an *intelligence* does not always help.

Here's a more drastic example with k switches and $n = 1$. Consider $k + 1$ flows from input i to output j each with rate $\frac{k}{k+1}$. Again, the no overload condition cannot be violated. Since we have k switches only, at least two flows must use the same switch, resulting in $\alpha \geq \frac{2k}{k+1}$. Therefore, if k is large, α is arbitrarily close to 2, and hence the throughput z is arbitrarily close to $1/2$. No *intelligence* in the non-splitting algorithm can avoid this situation, even if the non-splitting algorithm is *offline*, i.e. even if it has the knowledge of all the flows in advance [2].

With such an example in mind, one may wonder whether it is possible to force a non-splitting algorithm not to achieve

[2]It was shown in [6], that it is possible to achieve a throughput $z \geq 1/2$ with an offline non-splitting algorithm. Therefore, the above example provides a tight characterization of offline non-splitting algorithms. For these algorithms, one may also ask about the possibility of obtaining the maximum throughput, given the knowledge of the flows. Unfortunately, this problem is NP-hard (see [11] for a variety of problems and approximations regarding offline non-splitting algorithms for a load balanced switch).

any throughput at all (i.e. making z arbitrarily close to 0). The following section shows that this is not true, by proving the existence of a non-splitting algorithm that achieves a throughput $z > 1/3$.

III. A GREEDY NON-SPLITTING LOAD BALANCING ALGORITHM

We describe a non-splitting load balancing algorithm and call it the *greedy* non-splitting algorithm. This algorithms achieves a throughput $z > 1/3$. We claim no particular importance of the algorithm itself, but rather, we use it to show that the $1/3$ upper bound of Section IV is tight, i.e. no better upper bound can be proved.

Greedy non-splitting

$F_{i\to l} \leftarrow \phi$ for all $1 \le i \le n$ and $1 \le l \le k$
$F_{j\leftarrow l} \leftarrow \phi$ for all $1 \le j \le n$ and $1 \le l \le k$
repeat
given a flow f_0 from input i to output j,
find a switch $l \in \{1 \dots k\}$ such that

$$r_{f_0} + \sum_{f \in F_{i\to l}} r_f \le 3 - 2/k$$

$$r_{f_0} + \sum_{f \in F_{j\leftarrow l}} r_f \le 3 - 2/k$$

route flow f_0 through switch l
$F_{i\to l} \leftarrow F_{i\to l} \cup \{f_0\}$
$F_{j\leftarrow l} \leftarrow F_{j\leftarrow l} \cup \{f_0\}$
until no more flows

The name *greedy* comes from the fact that the non-splitting algorithm finds any switch l that satisfies the criteria shown above and routes the flow f_0 through it. Note that if no such switch exists, the greedy algorithm is not well defined!

Lemma 1: Given a flow f_0 from input i to output j, a switch l exists such that

$$r_{f_0} + \sum_{f \in F_{i\to l}} r_f \le 3 - 2/k$$

$$r_{f_0} + \sum_{f \in F_{j\leftarrow l}} r_f \le 3 - 2/k$$

Proof: Assume the opposite for the sake of contradiction. Then for all $1 \le l \le k$ either $r_{f_0} + \sum_{f\in F_{i\to l}} r_f > 3 - 2/k$ or $r_{f_0} + \sum_{f\in F_{j\leftarrow l}} r_f > 3 - 2/k$. If $k = 1$, then this is a contradiction to the no overload condition. Therefore, assume $k \ge 2$.

$$k r_{f_0} + \sum_{l=1}^{k} \Big(\sum_{f \in F_{i\to l}} r_f + \sum_{f \in F_{j\leftarrow l}} r_f \Big) > 3k - 2$$

$$k r_{f_0} + \sum_{f \in F_{i\to}} r_f + \sum_{f \in F_{j\leftarrow}} r_f > 3k - 2$$

$$(r_{f_0} + \sum_{f \in F_{i\to}} r_f) + (r_{f_0} + \sum_{f \in F_{j\leftarrow}} r_f) > 3k - 2 - (k-2) r_{f_0} \ge 2k$$

The last inequality holds because $r_{f_0} \le 1$ (no overload condition) and $k \ge 2$. Therefore, either $r_{f_0} + \sum_{f\in F_{i\to}} r_f > k$ or $r_{f_0} + \sum_{f\in F_{j\leftarrow}} r_f > k$, which contradicts the no overload condition.

■

Now since the greedy algorithm routes a new flow f_0 through switch l only when $r_{f_0} + \sum_{f\in F_{i\to l}} r_f \le 3 - 2/k$ and $r_{f_0} + \sum_{f\in F_{j\leftarrow l}} r_f \le 3 - 2/k$, then $\sum_{f\in F_{i\to l}} r_f \le 3 - 2/k$ and $\sum_{f\in F_{j\leftarrow l}} r_f \le 3 - 2/k$ after every routing decision for all $1 \le i, j \le n$ and $1 \le l \le k$. Therefore, $\alpha \le 3 - 2/k$ [3] [4] and hence the throughput $z \ge \frac{1}{3-2/k} > 1/3$.

Theorem 1: There exists a non-splitting load balancing algorithm that achieves a throughput $z > 1/3$.

IV. AN UPPER BOUND ON THROUGHPUT

In this section, we show an asymptotic upper bound of $1/3$ on the throughput of any non-splitting load balancing algorithm. By virtue of the previous section, this bound is tight, i.e. no better upper bound can be established. To show the desired upper bound on throughput, we show that $\alpha \ge 3 - \delta$, where $\lim_{k\to\infty} \delta = 0$, for any non-splitting algorithm. The basic idea behind our proof is the following: assume all flows have rate $\frac{k}{2\lceil k/2\rceil + 1}$ [5] and consider any fixed set of $\lfloor k/2 \rfloor$ switches. Assume also that we manage to force the algorithm to route $2\lfloor k/2 \rfloor$ flows for each output in a set of $2\lceil k/2 \rceil + 1$ outputs through those $\lfloor k/2 \rfloor$ switches. This scenario is illustrated in Figure 2.

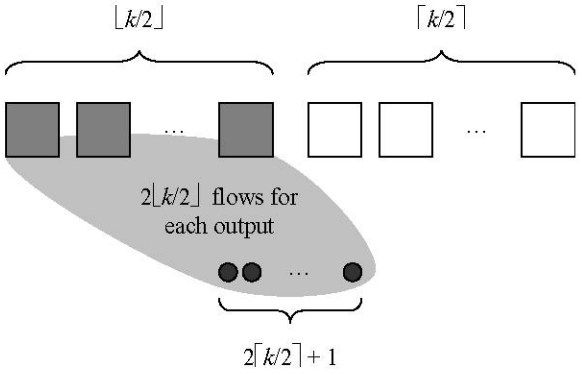

Fig. 2. Proof idea

We can also assume WLOG that the algorithm successfully avoids so far the event of routing three flows for a given port

[3] As a consequence of this result, full throughput ($z = 1$) can be achieved without splitting if link capacities are $3 - 2/k$ instead of 1 (this is at most 3 and only 2 when $k = 2$).

[4] We note the similarity of this result to the result of [15] which states that $3k - 2$ switches are necessary for a multirate Clos network to be wide-sense non-blocking. Since a multirate Clos network is a special case of the load balanced switch, $3k - 2$ switches (instead of k switches but with the no overload condition remaining the same) are also necessary for a non-splitting algorithm to achieve a throughput $z = 1$ (i.e. $\alpha \le 1$). This is an intuitive indication that $\alpha = 3 - 2/k$ (which could be interpreted as the number of switches needed divided by the number of switches available) must be close to optimal, as shown in Section IV.

[5] Although this rate approaches 1 (the internal line speed) at the limit, it is only a $1/k$ fraction of the external line speed.

(input or output) through the same switch. When such an event occurs, $\alpha \geq \frac{3k}{2\lceil k/2 \rceil + 1} = 3 - \delta$, where $\lim_{k \to \infty} \delta = 0$, and we are done. Therefore, each of the gray switches of Figure 2 is routing exactly two flows for each of the $2\lceil k/2 \rceil + 1$ outputs. The proof idea is based on the fact that the scenario of Figure 2 inevitably leads to the event mentioned above.

Consider an input with $2\lceil k/2 \rceil + 1$ flows, one for each of the above outputs. Referring to Figure 2 above, and based on the pigeon hole principle, we have one of two cases (or both):

- at least three of these flows are routed through the same white switch
- at least one of these flows, say for some output j, is routed through one of the gray switches. But since every gray switch is already routing two flows for output j, this results in three flows for output j routed through the same gray switch.

Note that the no overload condition is not violated because every port has at most $2\lceil k/2 \rceil + 1$ flows with a rate of $\frac{k}{2\lceil k/2 \rceil + 1}$ for each.

It remains to show that an algorithm that successfully avoids routing three flows for a given port through the same switch can be forced to reach the scenario of Figure 2. In doing so, we assume that n is large enough (but finite) compared to k.

Theorem 2 (Non-splitting load balancing algorithms): The throughput of a non-splitting load balancing algorithm is $z \leq 1/3 + \epsilon$, where $\lim_{k \to \infty} \epsilon = 0$ (when $n >> k$).

Proof: We show that $\alpha \geq 3 - \delta$, where $\lim_{k \to \infty} \delta = 0$, by constructing the scenario of Figure 2. Hence $z \leq 1/\alpha \leq 1/3 + \epsilon$, where $\lim_{k \to \infty} \epsilon = 0$. Let $n \geq (2\lceil k/2 \rceil + 1)^{(2\lfloor k/2 \rfloor + 1)}$. Let I_0 be the set of all input ports, and similarly O_0 be the set of all output ports. Pick $(2\lceil k/2 \rceil + 1)^{2\lfloor k/2 \rfloor}$ inputs in I_0 and let I_1 be the set of remaining inputs of I_0. For each input in $I_0 - I_1$, pick a distinct set of $2\lceil k/2 \rceil + 1$ outputs in O_0 and send one flow for each. Therefore, for each input $i \in I_0 - I_1$, there must be an output j such that the flow from i to j is routed through one of the first $\lfloor k/2 \rfloor$ switches (otherwise, input i will have three flows routed through the same switch). For each input $i \in I_0 - I_1$ fix one such output. Call the set of all such outputs O_1. Note that $|I_1| \geq (2\lceil k/2 \rceil + 1)^{2\lfloor k/2 \rfloor}$ [6] and $|O_1| = (2\lceil k/2 \rceil + 1)^{2\lfloor k/2 \rfloor}$. More importantly, every output in O_1 has one flow routed through the first $\lfloor k/2 \rfloor$ switches. Now consider I_1 and O_1 and repeat the same process, i.e. pick $(2\lceil k/2 \rceil + 1)^{(2\lfloor k/2 \rfloor - 1)}$ inputs in I_1 (and let I_2 be the set of remaining inputs of I_1) to obtain O_2 in a similar manner as above. Again, $|I_2| \geq (2\lceil k/2 \rceil + 1)^{(2\lfloor k/2 \rfloor - 1)}$ and $|O_2| = (2\lceil k/2 \rceil + 1)^{(2\lfloor k/2 \rfloor - 1)}$ and every output in O_2 has two flows routed through the first $\lfloor k/2 \rfloor$ switches. This can be repeated $2\lfloor k/2 \rfloor$ times to obtain $I_{2\lfloor k/2 \rfloor}$ and $O_{2\lfloor k/2 \rfloor}$ with $|I_{2\lfloor k/2 \rfloor}| \geq 2\lceil k/2 \rceil + 1$ and $|O_{2\lfloor k/2 \rfloor}| = 2\lceil k/2 \rceil + 1$ and every output in $O_{2\lfloor k/2 \rfloor}$ has $2\lfloor k/2 \rfloor$ flows routed through the first $\lfloor k/2 \rfloor$ switches. This is the scenario of Figure 2 with enough inputs in $I_{2\lfloor k/2 \rfloor}$ (only one is needed) to carry out the inevitable event described earlier. In this construction, we do not violate the no overload condition since every input port has at most $2\lceil k/2 \rceil + 1$ flows and every output port has at most $2\lfloor k/2 \rfloor$ flows and every flow has a rate of $\frac{k}{2\lceil k/2 \rceil + 1}$. ■

[6] This is because for any two numbers $a \geq 2$ and b, $a^{b+1} - a^b = a^b(a-1) \geq a^b$.

Figure 3 illustrates the construction described in the proof (scenario of Figure 2) for $k = 2$.

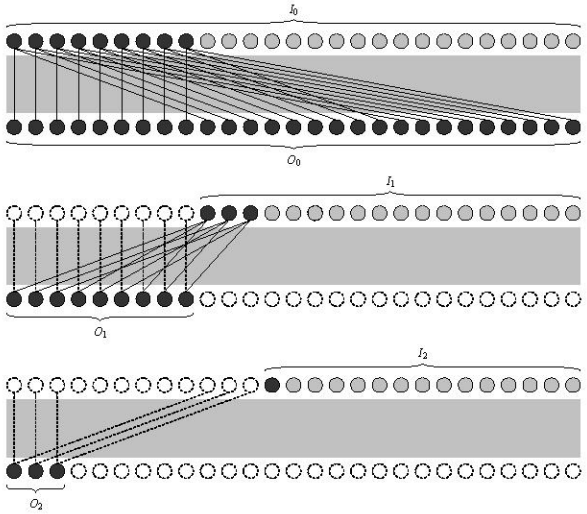

Fig. 3. Construction for $k = 2$. Assuming the algorithm avoids routing 3 flows for a given port through the same switch, at least one flow for each of $(2\lceil k/2 \rceil + 1)^{2\lfloor k/2 \rfloor} = 3^2 = 9$ inputs of I_0 is routed through the first switch. This results in O_1 with 9 outputs each having 1 flow routed through the first switch. Then at least one flow for each of $(2\lceil k/2 \rceil + 1)^{(2\lfloor k/2 \rfloor - 1)} = 3^1 = 3$ inputs of I_1 is routed through the first switch. This results in O_2 with 3 outputs each having 2 flows routed through the first switch. Now sending a flow from an input in I_2 to each of the 3 outputs in O_2 will result in at least one port (either that input or one of the 3 outputs) having 3 flows routed through the first switch.

V. Conclusion

For a load balanced switch with n ports and k switches (see Figure 1), we show that a load balancing algorithm that does not split, i.e. that routes all the packets of a given flow through exactly one of the k switches, and does not re-route existing flows, cannot achieve a throughput greater than $1/3$ asymptotically, when $n >> k$. Moreover, this upper bound on throughput is tight, i.e. there exists a non-splitting load balancing algorithm that achieves a throughput $z > 1/3$. Therefore, we establish a tight characterization of non-splitting load balancing algorithms: they waste $2/3$ of the throughput of a switch.

References

[1] Z. Cao, Z. Wang, E. Zegura, *Performance of hashing based schemes for interlet load balancing.* Proceedings of IEEE Infocom 2000, pp. 332-341, March 2000.

[2] C.-S. Chang, D.-S. Lee, T.-S. Jou, *Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering.* Computer Communications, Vol. 25, No. 6, pp. 611-622, 2002.

[3] C.-S. Chang, D.-S. Lee, C.-M. Lien, *Load balanced Birkhoff-von Neumann switches, part II: multi-stage buffering.* Computer Communications, Vol. 25, No. 6, pp. 623-634, 2002.

[4] C.-S. Chang, D.-S. Lee, C.-Y. Yue, *Providing guaranteed rate services in the load balanced Birkhoff-von Neumann switches.* Proceedings of IEEE Infocom 2003, San Francisco, CA, USA.

[5] G. Dittmann, A. Herkersdorf, *Network processor load balancing for high speed links.* Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS 2002, pp. 727-735, San Diego, CA, USA, July 2002.

[6] D. Z. Du, B. Gao, F. K. Hwang, J. H. Kim, *On multirate rearrangeable Clos networks.* Siam Journal of Computing 28(2), pp. 463-70, 1998.

[7] S. Iyer, N. McKeown, *Making parallel packet switches practical.* Proceedings of IEEE Infocom 2001, pp. 1680-87, March 2001.

[8] L. Kencl, J. Y. Le Boudec, *Adaptive load sharing for network processors.* Proceedings of IEEE Infocom 2002, pp. 545-554, New York, NY, USA, June 2002.

[9] I. Keslassy, N. McKeown *Maintaining packet order in two-stage switches.* Proceedings of IEEE Infocom, 2002, New York, NY, June 2002.

[10] I. Keslassy, *The load balanced router.* Ph.D. Thesis, Stanford University, 2004.

[11] S. Mneimneh, K.-Y. Siu, *Scheduling unsplittable flows using parallel switches.* Proceedings of IEEE ICC 2002, New York, NY, USA, April 2002.

[12] S. Mneimneh, *Load balancing in a switch without buffers.* Proceedings of IEEE Workshop on High Speed Switching and Routing, Poland, June 2006.

[13] R. Russo, B. Metzler, P. Droz, L. Kencl, *Scalable and adaptive load balancing on IBM PowerNP.* Technical Report No. RZ-3431, IBM Research Report, July 2002.

[14] A. Baldini, D. Bergamasco, D. Blumenthal, D. Chiou, S.-T. Chuang, V. De Feo, P. Donner, S. Fraser, Y. Ganjali, P. Giaccone, I. Keslassy, M. Kodialam, F. Mattus, N. McKeown, D.-S. Lee, S. Mneimneh, M. Neely, A. Smiljanic, B. Towles, R. Zhang-Shen, M. Zirngibl, *Stanford Workshop on Load Balancing.* Stanford Universiry, CA, May 19 2004. http://yuba.stanford.edu/lb/ .

[15] K.-H. Tsai, D.-W. Wang, F. Hwang, *Lower bounds for wide-sense nonblocking Clos network.* Theoretical Computer Science 261, pp. 323-28, 2001.