A Simple Re-Sequencing Load-Balanced Switch Based on Analytical Packet Reordering Bounds

Sen Yang
Georgia Institute of Technology
Email: sen.yang@gatech.edu

Bill Lin
University of California, San Diego
Email: billlin@eng.ucsd.edu

Paul Tune
University of Adelaide
Email: paul.tune@adelaide.edu.au

Jun (Jim) Xu
Georgia Institute of Technology
Email: jx@cc.gatech.edu

***Abstract*—Chang et al. proposed the load-balanced switch in their seminal work [1], which has received wide attention due to its inherent scalability properties in both size and speed. These scalability properties continue to be of significant interest due to the relentless exponential growth in Internet traffic. The main drawback of the load-balanced switch is that packets can depart out-of-order from the switch, which can significantly degrade network performance by negatively interacting with TCP congestion control. Hence, a large body of subsequent work has proposed a variety of modifications for ensuring packet ordering, but all the proposed approaches tend to increase packet delay significantly in comparison to the basic load-balanced switch. In this paper, we show that the amount of packet reordering that can occur with the load-balanced switch is actually quite limited, which means that packet reordering can simply be rectified by employing reordering buffers at the switch outputs. In particular, we formally bound the worst-case amount of time that a packet has to wait in these output reordering buffers before it is guaranteed to be ready for in-order departure with high probability, and we prove that this bound is *linear* with respect to the switch size. This linear bound is significant because previous approaches can add quadratic or cubic delays to the load-balanced switch. In addition, we use a hash-grouping method that further reduces resequencing delays significantly. Although simple and intuitive, our experimental results show that our output packet reordering approach substantially outperforms existing load-balanced switch architectures.**

I. Introduction

Network traffic across the Internet as well as inside data centers continues to grow exponentially. This relentless traffic growth is fueled by an increasing adoption of video streaming and cloud computing, and a proliferation of network-connected devices with increasing networking capabilities. To keep up with the ever increasing traffic demands with reliable service, network operators need high-performance switch architectures that can scale well in both switch size (in terms of the number of switch ports) and link speed, provide throughput guarantees, achieve low latency, and maintain packet ordering. Unfortunately, conventional switch architectures have not been able to keep up with these challenges.

A promising class of highly scalable switch architectures, first introduced by Chang et al. [1], [2], and later further developed by others (e.g. [3], [4], [5], [6]), is the *load-balanced switch* (LBS). As shown in Fig. 1, a generic LBS relies on two switching stages for forwarding packets. The first switching stage connects the input ports to the center stage of intermediate ports, and the second switching stage connects the center stage of intermediate ports to the final stage of output ports. Both switching stages execute a deterministic connection pattern such that each input is connected to each output of a switching stage $1/N$-th of the time.

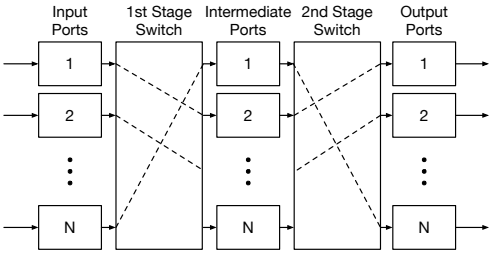

Fig. 1. Generic load-balanced switch.

This architecture can be implemented using two identical $N \times N$ crossbar switching stages where each switching stage goes through a predetermined periodic sequence of cyclic-shift connection patterns such that each input is connected to each output of a switching stage exactly once every N cycles. Alternatively, as shown in [5], the deterministic connection pattern can also be efficiently implemented using optics where all inputs are connected to all outputs of a switching stage in parallel at a rate $1/N$-th of the line rate. LBSes are much more scalable, in terms of both switch size and link speed, than conventional switch architectures. This is because, in LBS architectures, the connection pattern during every switching cycle at each switching stage is fixed and requires zero computation, and each port can forward packets in a fully distributed manner based only on local information.

A. The Packet Reordering Problem

Although the basic LBS originally proposed in [1] is highly scalable, it has the critical problem that packets can depart out-of-order from the switch. In the basic LBS, consecutive packets at an input port are spread to all N intermediate ports upon arrival. These packets, going through different intermediate ports, may encounter different queuing delays. Thus, some of these packets may arrive at their output ports out-of-order. This is detrimental to Internet traffic since the widely used TCP transport protocol falsely regards out-of-order packets as indications of congestion and packet loss. The outcome

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

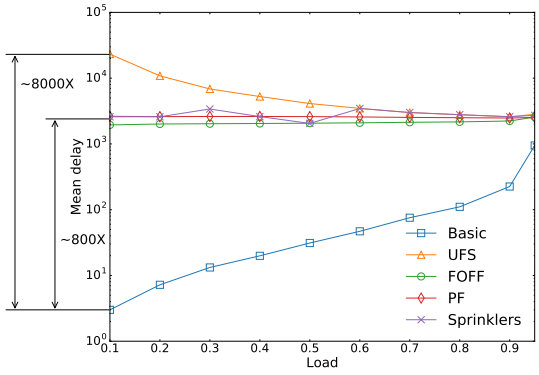

Fig. 2. Poor average delays under moderate loads. A uniform traffic pattern was used here. The switch size $N = 64$. Loads are normalized to 1. At a load of 0.1, the average delay of the proposed schemes are about 800 times higher than the basic LBS (labeled "Basic").

is the retransmission of packets, often multiple times, further exacerbating the problem. Therefore, a number of researchers have subsequently explored this packet ordering problem.

Most existing approaches to the packet ordering problem are based on some form of complete or partial aggregation of packets into frames or stripes. Uniform Frame Spreading (UFS) [5], Full-Order Frames First (FOFF) [5], Padded Frames (PF) [4], and Sprinklers [3] are representative examples of such approaches. However, these methods pay a significant price for ensuring packet ordering in that they perform significantly worse than the basic LBS [1].

Fig. 2 compares these aggregation-based methods with the basic LBS (labeled as "Basic") that does not guarantee packet ordering. The results are for a uniform traffic pattern, one in which the output port destination for every arriving packet is chosen uniformly at random. As can be seen, these approaches all have packet delays can be significantly higher than the basic LBS, especially under moderate loads.

B. Our Approach

In this work, we propose a natural solution, named the RS-LBS (Re-Sequencing Load-Balanced Switch), that incurs significantly lower packet delays compared to the aforementioned packet-aggregation-based solutions. Our solution is based on the following observation of the queuing dynamics in the basic LBS: although two "back-to-back" packet arrivals A (earlier) and B (later) belonging to the same *switch flow* (i.e., they share the same input and output) may reach the output port out of order as explained above, the amount of reordering (i.e., how much earlier B arrives at the output port before A) is upper-bounded with high probability by a fairly small value. The reason for this is that under the round-robin load dispatching mechanism of the basic LBS, the two VOQs at two different intermediate ports that A and B traverse through respectively, have almost independent and stochastically identical queuing processes. Hence the delays A and B experience at their respective intermediate ports are almost i.i.d. (independent and identically distributed) random variables, so with high probability the latter cannot be much larger than the former.

Our basic RS-LBS scheme is for an output port to hold each out-of-order packet, for a period of time not exceeding this (small) upper-bound, for re-sequencing. While packets that are severely out-of-order, i.e., those causing this upper-bound to be exceeded, may remain out-of-order after the re-sequencing, they represent a tiny percentage of the network traffic and may negatively impact the performance of an even tinier percentage of TCP flows. We prove that this upper bound is *linear* with respect to the switch size, i.e., $O(N)$, under any traffic arrival process for which the switch is stable. Hence with a worst-case delay of at most $O(N)$ due to this re-sequencing, we can ensure that an overwhelming majority of packets exit the switch in-order; in fact, since the output port holds a packet for only as long as is necessary, the average delay of packets is much smaller than this worst-case delay, as will be shown in Section V. In contrast, all other present solutions to the packet reordering problem incur $O(N^2)$ or $O(N^3)$ delays.

Our solution can be further refined to reduce the average delay of packets by relaxing the packet ordering semantics as follows. The semantics discussed above, which we refer to as *per-switch-flow ordering*, aims to ensure that packets arriving to the same input port departs from the corresponding output port in the same order. The per-switch-flow ordering is unnecessarily conservative because if two out-of-order packets within the same switch flow belong to *two different TCP flows*, then having them depart the switch in a different order would not negatively impact either TCP flow. The most relaxed yet still harmless semantics would be to only ensure the sequencing of packets within a TCP flow, which we refer to as *per-TCP-flow ordering*. With per-TCP-flow ordering, the average packet delay (due to re-sequencing) can be reduced significantly, as an out-of-order packet no longer has to be held waiting for packets from other TCP flows. This relaxed semantics, however, is computationally expensive to implement, since to do so would require each output port to perform per-TCP-flow queuing and re-sequencing. We propose the use of a slightly less relaxed semantics called *per-hashed-group ordering*. With per-hashed-group ordering, incoming packets at an input port with the same departure output port (i.e., the same switch flow) are hashed to one of K counters and are numbered by the corresponding hashed counter. Per-hashed-group ordering simply requires two packets belonging to the same *hashed group* to depart in order in accordance to their assigned sequence numbers[1]. This slightly less relaxed semantics results in almost the same level of reduction to the average delay as per-TCP-flow ordering, but yet it has an implementation complexity comparable to that of per-switch-flow ordering.

C. Contributions of the Paper

This paper makes three major contributions:

- First, we show that the amount of packet reordering is, with high probability, quite limited. More specifically, we formally derive a linear bound on the amount of time that

[1]In contrast, with per-switch-flow ordering, incoming packets are numbered in accordance to their arrivals to the same switch flow.

a packet has to wait at the output port before it is, with high probability, ready for in-order departure.

- Second, using flow statistics measured from real-world Internet traces, we show by means of simulations that our solution, with the standard per-switch-flow ordering semantics, leads to very low packet delays, enabling significantly better performance compared to existing LBS solutions.
- Third, we apply a more relaxed yet still harmless packet ordering semantics, called per-hashed-group ordering, that can further reduce the average packet delay significantly, yet has a low implementation complexity comparable to that of the standard and more conservative per-switch-flow ordering semantics.

The rest of the paper is organized as follows. In Section II, we analyze the amount of packet reordering under in a basic LBS. In Sections III and IV, we present the basic and enhanced (with hash-grouping) RS-LBS schemes. In Section V, we compare the average delay performance of our re-sequencing schemes with existing LBS architectures. In Section VI, we provide a brief review of related work, before concluding the paper in Section VII.

II. Analysis of Packet Reordering Probability

As explained earlier, the efficacy of our natural solution hinges upon the premise that only a tiny percentage of packet pairs are severely reordered when they reach their output ports. A packet pair A and B within the same switch flow, A arriving earlier at the input port than B, are considered severely reordered, if B arrives earlier than A by at least a lateness threshold θ. Clearly, the larger is this lateness threshold θ, the smaller is the proportion of severely reordered packet pairs.

In this section, we show, through a careful analysis, that even with a fairly small lateness threshold of $O(N)$, the proportion of severely reordered packet pairs can go down to a tiny number. Note this $O(N)$ result holds for any traffic arrival process for which the switch is stable. However, in analyzing the constant factor (e.g., "23" in the following example) in this $O(N)$, we assume the arrival process is Poisson. For example, we will show that when the switch is 90% loaded under Poisson traffic and the lateness threshold is set to $23N$, no more than 1 out of 1,000 "back-to-back" packet (arrival) pairs in the same switch flow will be severely reordered (There is a quotation mark around the word "back-to-back" here, since between this pair of packet arrivals there can be other packet arrivals belonging to other switch flows to this input port). This property allows our natural solution of re-sequencing packets at the output ports to remove all but a tiny proportion of out-of-order packets while introducing at most $O(N)$ re-sequencing delay to each packet.

A. Problem formulation

Consider two "back-to-back" packets A and B *in the same swich flow* where A arrives before B. Let m_1 and m_2 be the intermediate ports that A and B transit through respectively, and j be their common destination output port. Then A and B both transmit through the j-th VOQ (for buffering all packets destined for output port j) at the corresponding intermediate ports. If these two packets happen to transit through the same intermediate port, i.e., $m_1 = m_2$, then they clearly will not be reordered because they go through the same intermediate VOQ. Hence we assume $m_1 \neq m_2$ when we analyze the reordering probability.

Let L_1 and L_2 be random variables denoting the queuing delays that A and B experience, at the j-th VOQs of the intermediate ports m_1 and m_2, respectively. Define a *cycle* as N *time slots*. Even if A and B depart from the input port simultaneously, B can arrive at the output port at most $L_1 - L_2$ cycles before A. Based on this observation, given any lateness threshold $\theta > 0$ (in the unit of cycles), we aim to derive $P(L_1 - L_2 > \theta)$, the probability that this pair of "back-to-back" packets is severely reordered (with respect to θ).

Note this measure of packet reordering, namely *"back-to-back" packet reordering probability*, is different than the conventional measure readers may have in mind. In particular, this measure does not account for all packet reordering cases that may impact TCP performance, which would be accounted for in the conventional measure. For example, consider 5 "consecutive" packets A, B, C, D, and E in the same switch flow. This measure accounts only for the reorderings between 4 pairs of "back-to-back" packets: (A, B), (B, C), (C, D) and (D, E), but not for other pairs such as (A, D) and (C, E).

We emphasize, however, that our measure is only used for the theoretical analysis, including the $O(N)$ packet waiting time proof, and for deciding on the right parameters used in our packet buffering and re-sequencing schemes. We will use the conventional measure of packet reordering when evaluating the efficacy of our schemes. We further emphasize that our $O(N)$ proof (established below) remains valid according to the conventional measure, because our measure turns out to be more conservative than the conventional measure for the following two reasons.

First, in estimating the reordering probability of a pair of "back-to-back" packets, we assume these two packets depart from the input port, to their respective intermediate ports, simultaneously. However, in reality, two "back-to-back" packet arrivals (say A and B) within the same switch flow can be separated by τ packet arrivals, where τ can be tens, hundreds or more packets from other switch flows. These τ intervening packets from other switch flows would cause a gap of at least τ time slots (or τ/N cycles) between the departure times of A and B from the input port. The reordering probability $P(L_1 - L_2 > \theta + \tau/N)$ can be much smaller than $P(L_1 - L_2 > \theta)$, when τ is tens, hundreds or more. Second, the gap between the departure times of other pairs of packets within the same switch flow, say A and C, is typically much wider than this τ, the gap between the departure times of "back-to-back" pairs, hence the reordering probability between any such pair is generally negligible compared to that between a "back-to-back" pair.

As mentioned earlier, each input port distributes incoming packets, regardless of their destination output ports (i.e., the

switch flows they belong to), uniformly to all N intermediate ports in a round-robin fashion. It has been shown in [1] that, under very mild assumptions on the traffic arrival process to the switch (i.e., to its input ports) such as weakly mixing, for any j, m_1, and m_2, the packet arrival process to the j-th VOQ of the intermediate port m_1 and that to the j-th VOQ of the intermediate port m_2 can be viewed as i.i.d. stochastic random processes. Since these two intermediate VOQs also have the same deterministic service process, more specifically switching one packet to output port j every N time slots, the queuing processes of VOQs are i.i.d. Hence the queuing delays L_1 and L_2 are i.i.d. random variables.

Let the packet arrival process to each of the N^2 switch flows be independent of each other and be stationary. Let $\lambda_{ij} \geq 0$ be the average arrival rate of packets arriving at input port i destined for output port j. Define $\lambda_j \equiv \sum_{i=1}^{N} \lambda_{ij}$ to be the total average arrival rate for output port j. Let the service rate of each input or intermediate port be 1. For this arrival process (to the switch) to be admissible, we must have

$$\sum_{j=1}^{N} \lambda_{ij} < 1, \qquad i = 1, 2, \ldots, N,$$
$$\sum_{i=1}^{N} \lambda_{ij} < 1, \qquad j = 1, 2, \ldots, N.$$

B. The $O(N)$ Proof

We again emphasize that Poisson assumption is not needed for deriving this $O(N)$ proof. Define $\pi_n \equiv P(L_1 = n)$ for $n = 0, 1, 2, \ldots$. Recall L_1 is the queueing delay packet A experiences at the j-th VOQ of the intermediate port m_1. Since the arrival process to this VOQ is stationary, as stated before, and the service process is deterministic (and hence stationary), the distribution of L_1 is precisely the stationary queueing delay distribution of this VOQ. Since the term N does not appear in either the arrival rate λ_j or the service rate 1 (i.e., 1 packet every cycle or N time slots), it should not appear in the distribution of L_1 (i.e., any of the π_n terms). Hence each π_n term is a function of only λ and n. Since L_1 and L_2 are i.i.d., by the convolution formula, we have,

$$\begin{aligned} P(L_1 - L_2 \geq \theta) &= \sum_{k=\theta}^{\infty} P(L_1 - L_2 = k) \\ &= \sum_{k=\theta}^{\infty} \sum_{i=0}^{\infty} P(L_2 = i) P(L_1 = i + k) \\ &= \sum_{k=\theta}^{\infty} \sum_{i=0}^{\infty} \pi_i \pi_{i+k}. \end{aligned}$$

Note that the term N does not appear in the above formula. This implies that $P(L_1 - L_2 \geq \theta)$ is a function of only θ and λ_j. Hence given a load factor λ_j and a target $P(L_1 - L_2 \geq \theta)$ value (say 10^{-3}), the θ value that allows us to reach (i.e., go under) this target $P(L_1 - L_2 \geq \theta)$ value is a constant (with respect to N). Note that under any admissible traffic arrival process to the switch $P(L_1 < \infty) = 1$, so any nonnegative target $P(L_1 - L_2 \geq \theta)$ value can be reached, no matter how small it is. Hence the output needs to hold packet B for up to θ cycles, or θN time slots, to achieve this target $P(L_1 - L_2 \geq \theta)$ value. *This proves that at most $O(N)$ amount of buffering (and waiting) is needed at each output port to re-sequence the vast majority of packets.*

C. Analysis of the Constant Factor θ

In analyzing the constant factor θ in this $O(N)$ result, we assume that arrival processes to all N^2 switch flows are Poisson. With this Poisson assumption, the queuing process of the j-th intermediate VOQ at intermediate port m_1 can be viewed as M/D/1 (i.e., Poisson arrival[2] and deterministic service time). Then the distribution of L_1 is equal to the stationary queueing delay distribution of this M/D/1 queue. The same can be said about the queuing process of the j-th intermediate VOQ at intermediate m_2, and about L_2.

1) The distribution of L_1 and its numerical computation: Recall that the (normalized) arrival rate to this M/D/1 queue (i.e., the j-th VOQ at intermediate port m_1) is λ_j and its service rate is 1 (i.e., 1 packet every cycle or N time slots). For the ease and clarity of the presentation, we drop the subscript j from λ_j and denote the total load to the output port j simply as λ. By the Pollaczek-Khinchin formula [7], we have

$$\pi_0 = 1 - \lambda \tag{1}$$
$$\pi_1 = (1 - \lambda)(e^{\lambda} - 1) \tag{2}$$
$$\pi_n = (1 - \lambda) \sum_{k=1}^{n-1} e^{k\lambda} (-1)^{n-k} \left[\frac{(k\lambda)^{n-k}}{(n-k)!} + \frac{(k\lambda)^{n-k-1}}{(n-k-1)!} \right] + (1 - \lambda) e^{n\lambda}. \tag{3}$$

Unfortunately, computing (3) leads to numerical instability, since it is a summation of alternating sign large absolute values. Our tests show that numerical computation via Matlab can accurately compute only up to π_{10} when $\lambda = 0.1$ (10% loaded) and up to π_{20} when $\lambda = 0.9$ (90% loaded). We overcome the problem by using following upper bound of π_n that is numerically stable [8]:

$$\pi_n \leq -\frac{\alpha_0}{\zeta_0^{n+1}} + \frac{M_1(r)}{r^n}, \quad n = 0, 1, 2, \ldots \tag{4}$$

where ζ_0, α_0, r and $M_1(r)$ are parameters that arise in analyzing this distribution via complex analysis.

Fig. 3 presents the numerical results of this upper bound for load factors $\lambda = 0.1, 0.5, 0.9$, with the parameters listed in Table I. We see from the figure that the upper bound given by (4) is quite tight when n is large. Thus whenever n is large enough so that π_n cannot be computed accurately using equation (3), it can be tightly bounded using equation (4).

[2]Due to the discretization of time into slots by the switch, this arrival process (to an intermediate VOQ) is i.i.d. Bernoulli, not Poisson, even when the arrival process to the input port is assumed to be Poisson. However, when N is large, which is what LBS is designed for, the Bernoulli process is stochastically very close to the Poisson process.

TABLE I
PARAMETERS FOR THE UPPER BOUND IN EQUATION (4) (TAKEN FROM [8] WITH A CORRECTION).

λ	ζ_0	α_0	r	$M_1(r)$
0.1	37.1	−444.8	84.0	384.0
0.5	3.5	−5.8	16.0	60.6
0.9	1.2	−0.26	8.0	0.95

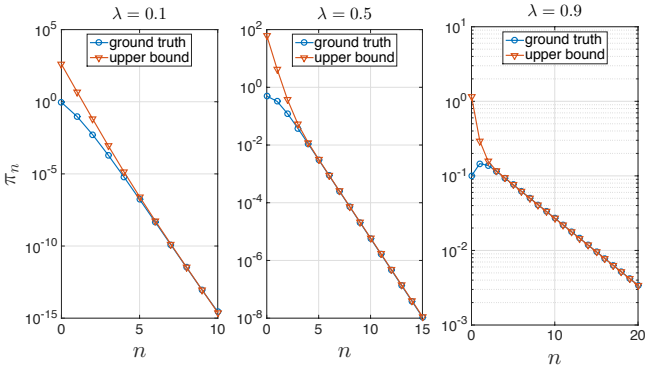

Fig. 3. Comparing the true value of π_n (denoted *ground truth*) with its upper bound under $\lambda = 0.1, 0.5, 0.9$.

2) Piecing Everything Together: Combining equations (1), (2), (3) and (4), we obtain the following upper bound of $P(L_1 - L_2 \geq \theta)$, which is also a close approximation of it. Here d is the aforementioned threshold such that, when $n > d$, an accurate numerical computation of π_n using equation (3) becomes impossible. Recall that, as explained before, when $n > d$, the π_n terms in equation (3) can each be tightly bounded using equation (4), and their sums result in the right-hand-side terms in inequalities (5) and (6). We omit its proof due to lack of space.

Theorem 1: When $\theta > d$, we have

$$P(L_1 - L_2 \geq \theta) \leq C_\zeta^{(d)} \frac{1}{\zeta_0^\theta} + C_r^{(d)} \frac{1}{r^\theta} \tag{5}$$

When $0 \leq \theta \leq d$, we have

$$P(L_1 - L_2 \geq \theta) \leq \sum_{\ell=\theta}^{d} \sum_{i=0}^{d-\ell} \pi_i \pi_{i+\ell} + C_\zeta^{(d)} \frac{1}{\zeta_0^{d+1}} + C_r^{(d)} \frac{1}{r^{d+1}} + \widetilde{C}_\zeta^{(\theta,d)} + \widetilde{C}_r^{(\theta,d)}, \tag{6}$$

where

$$C_\zeta^{(d)} = -\sum_{i=0}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{i+1}} + \frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})}$$

$$C_r^{(d)} = \sum_{i=0}^{d} \frac{\pi_i M_1(r)}{r^i} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} + \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})}$$

$$\widetilde{C}_\zeta^{(\theta,d)} = \left(\frac{\alpha_0^2}{\zeta_0^{2d+4}(1-\zeta_0^{-2})} - \frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} \right) \sum_{\ell=\theta}^{d} \frac{1}{\zeta_0^\ell} - \sum_{\ell=\theta}^{d} \sum_{i=d-\ell+1}^{d} \frac{\pi_i \alpha_0}{\zeta_0^{\ell+i+1}}$$

$$\widetilde{C}_r^{(\theta,d)} = -\left(\frac{\alpha_0 M_1(r)}{\zeta_0^{d+2} r^{d+1}(1-\zeta_0^{-1} r^{-1})} - \frac{M_1^2(r)}{r^{2d+2}(1-r^{-2})} \right) \sum_{\ell=\theta}^{d} \frac{1}{r^\ell} + \sum_{\ell=\theta}^{d} \sum_{i=d-\ell+1}^{d} \frac{\pi_i M_1(r)}{r^{\ell+i}}.$$

3) Numerical Results of $P(L_1 - L_2 \geq \theta)$: Let $\overline{P}(\theta)$ denote the tight upper bound of $P(L_1 - L_2 \geq \theta)$ given in the right hand sides of (5) and (6). $\overline{P}(\theta)$ for $\lambda = 0.1, 0.5, 0.9$ is plotted in Figure 4. From their definitions given in [8], we can infer that $r \geq \zeta_0 > 1$, $\alpha_0 < 0$ and $M_1(r) > 0$, and thus $\overline{P}(\theta)$ is a decreasing function of θ. This monotonicity can also be seen from Figure 4.

Given the monotonicity of $\overline{P}(\theta)$, we can define L_ϵ as follows. For any $\epsilon > 0$, let L_ϵ be the smallest nonnegative number such that $\overline{P}(\theta) \leq \epsilon$ if and only if $\theta \geq L_\epsilon$. Since the service rate of an intermediate port VOQ is $1/N$, as explained earlier, with probability no more than ϵ, packet B arrives at output j at least $L_\epsilon N$ time slots earlier than A. So if B waits up to $L_\epsilon N$ time slots at the output port, the probability that B departs from the output port earlier than A should be no more than ϵ. In other words, if we maintain a re-sequencing buffer of size $\lceil L_\epsilon \rceil N$ at each output port, we can expect the packet reordering probability to be no more than ϵ. For example, for $\epsilon = 10^{-3}$ and $\lambda = 0.9$, we have $\lceil L_\epsilon \rceil = 23$. In other words, if each output port maintains a re-sequencing buffer of size $23N$, the switch can keep the packet reordering probability below 10^{-3} when the traffic load is no more than 90%.

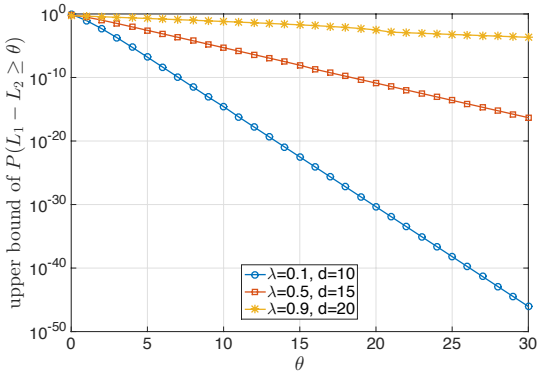

Fig. 4. $\overline{P}(\theta)$, the upper bound of $P(L_1 - L_2 \geq \theta)$.

III. THE BASIC RS-LBS SCHEME

In the previous section, we proved that a "back-to-back" packet pair A and B in the same switch flow is, with high probability, re-ordered by at most $O(N)$ time slots. Hence, the core idea of our basic RS-LBS scheme is, for each output port j, to hold out-of-order packets for at most $O(N)$ time slots and re-sequence them to the extent possible under this $O(N)$ maximum holding time constraint. With this idea, in theory at most $P(L_1 - L_2 \geq \theta)$ fraction of packets will remain re-ordered after this buffering and re-sequencing.

A. Re-sequencing Policies

The key idea of the basic RS-LBS scheme is, however, insufficiently detailed to define our scheme. The following packet re-sequencing policy at each output port fills the gap:

- Whenever a packet has waited for θN (equal to buffer size) time slots at an output port buffer, we mark this packet (and all packets before it) as expired.
- We mark a packet as in-order, if and when all packets have departed from the output port that are in the same switch flow as, and arrived earlier at the input port than, the packet.
- A packet can depart from the output port whenever it is in-order or expired. If there are multiple expired packets in the buffer, they will be served in the right order (according to the order they arrive at the input port as indicated by an internal sequence number field assigned by the input port).

Since the policy guarantees that no packet waits more than θN time slots in the buffer, the re-sequencing delay is no larger than θN. A drawback of this policy, however, is that it does not fully utilize the buffer space, because it may force an expired packet to depart from the output port *unnecessarily* (i.e., even when the buffer is not full). An alternative policy that fully utilizes the buffer is as follows:

- Each out-of-order packet shall keep waiting in the buffer until it becomes in-order or the buffer is full.
- When the buffer is full and there are no in-order packets in the buffer, we mark the packet that arrived at the switch (to any input port) the earliest as "expired".
- A packet can depart from the output port whenever it is in-order or expired.

The second policy essentially keeps a packet in the buffer for as long as possible in hopes of getting it back in-order. It is more aggressive than the first policy in reducing the proportion of reordered packets, so it should result in a smaller number of packet reorderings than the first policy. However, conceivably there is a tradeoff here: the second policy may result in a larger average re-sequencing delay than the first policy. To make an informed decision as to which policy to adopt, we compared the reordering proportions and re-sequencing delays of these two policies via simulations. Here is a summary of the observations:

- The re-sequencing delay of first policy is slightly smaller than that of the second when the buffer size is small ($W = 2N$ or so), but are almost the same when the buffer size is large ($W \geq 5N$).
- The re-ordering probability of the second policy is much smaller than that of the first one.

We conclude from these observations that the second policy achieves a much better tradeoff between packet reordering probability and re-sequencing delay. In addition, as will be shown in Section V-B, the re-sequencing delay will be even less an issue in the enhanced (by hash-grouping) RS-LBS scheme. Hence, we adopt the second policy in both the basic and the enhanced RS-LBS schemes.

B. Implementation Issues

In this section, we explain some implementation issues regarding the aforementioned second packet re-sequencing policy that we adopt for our RS-LBS schemes. At any input port, an internal (switch-wide) header is added to each incoming packet that contains a timestamp, corresponding to the arrival time (slot) of the packet, and the identifier of the input port. Information contained in this header is used by the output ports for implementing the packet re-sequencing policy. At an output port, the packet re-sequencing operation dictated by this policy can be implemented using a heap of $O(N)$ nodes, and hence has a computational complexity of $O(\log N)$ per packet. This complexity can be further decreased to $O(1)$ using *pipelined* implementations of heaps or similar sorting data structures [9], [10], [11].

C. Stability of the Basic RS-LBS Scheme

Since our scheme holds at most $O(N)$ additional (waiting) packets in each output port buffer, compared to the basic LBS, our scheme is stable for any traffic arrival process for which the basic LBS is stable. Note that, for all practical purposes, we may consider the basic LBS to be stable under all aforementioned admissible traffic arrival processes, although in theory, some very mild conditions, such as weakly mixing, need to be imposed on the arrival processes to preclude the pathological cases in which the destination output ports of packets arriving at an input port are perfectly synchronized with their "sequence numbers modulo N" that create a severe load-imbalance across the intermediate ports. We refer readers to [1] for further details on the mild conditions and the pathological cases they aim to preclude.

IV. Improving the performance by hash-grouping

As explained in the introduction, the only type of packet reordering that negatively impacts TCP performance is the reordering between two packets that belong to the same TCP flow. However, our basic RS-LBS scheme is oblivious to the flow identifier of a packet, so a packet already in-order within its own TCP flow may well be out-of-order within its switch flow, and be kept waiting in the output port buffer. Clearly, this wait time unnecessarily increases the average re-sequencing delay of a packet. A natural solution to this problem is to make each output port aware of the (TCP) flow identifier information in the packets. This solution, however, requires the output port to perform per-TCP-flow buffering and queuing, which is prohibitively expensive computationally.

We use a *hash-grouping* technique that was first introduced in [12], which significantly enhances the performance of the basic RS-LBS scheme, but yet has almost the same computational complexity and implementation cost as the basic scheme. The key idea is that, at each input port, the packets inside each switch flow are demultiplexed into K virtual groups via hashing. In other words, the (TCP) flow identifier of each packet is hashed to produce the index of the group to which the packet belongs. Note that since all packets of a

TCP flow have the same flow identifier, they will be assigned to the same virtual group via hashing.

The packet re-sequencing policy needs only to be slightly revised to take advantage of hash-grouping. A packet in a hashed group is considered in-order if it is in-order within its own group, even though it is not in-order within its switch flow. With a sufficiently large number of groups, each group contains only a small number of active TCP flows at any given time. In this case, a large fraction of unnecessary wait time is avoided, and the resulting re-sequencing delay becomes so small that it is almost the same as if the output ports were to enforce correct packet ordering only within each TCP flow. We will demonstrate the efficacy of this technique in Section V-B.

V. Evaluation

In this section, we compare the performance of our proposed approach with other existing load-balanced switching algorithms, including the basic LBS [1], Uniform Frame Spreading (UFS) [5], Full-Ordered Frame First (FOFF) [5], Padded Frames (PF) [4], and the recently proposed Sprinklers scheme [3]. Although the basic LBS (labeled "Basic" in subsequent figures) does not guarantee packet ordering, it is included in our comparisons because it provides a lower bound on the average delay that any LBS-based scheme can achieve. UFS, FOFF, PF, and Sprinklers are all known to provide fairly good performance while guaranteeing packet ordering.

Throughout this section, we assume $N = 64$. The normalized traffic load λ injected into the switch ranges from 0.1 to 0.95, while the output buffer size W varies from 0 to ∞ (i.e., no restriction on the buffer size). We also vary the traffic patterns in our evaluation. Our simulation covers 4 types of traffic matrices: uniform, quasi-diagonal, log-diagonal and diagonal. The load matrices are listed in the order of how skewed the traffic is to an output port: from uniform being the least skewed, to diagonal being the most skewed.

In order to simulate the effects of distributing TCP/UDP application flows into hashed groups, we generate application flows over the simulation period using flow statistics measured from a real-world Internet traffic trace. Specifically, we assume that the arrival of new application flows to an input port follows a Poisson process, and the rate and duration of each new application flow, viewed as a random vector, follows the corresponding joint empirical distribution measured from the traffic trace under consideration. The rate of this Poisson process is set according to the intended traffic rate λ of the input port in a simulation run, and the measured empirical average size (number of packets) of an applications flow.

The trace used was donated privately to the authors and was collected by University of North Carolina (UNC) on a 1 Gbps access link connecting the campus to the rest of the Internet on April 24, 2003. It contains $198,944,706$ packet headers and around 13.5 million flows. In our simulation study, incoming traffic to each input port is generated according to the empirical flow statistics measured from this trace.

In the remainder of this section, we will first look at the performance of the basic RS-LBS scheme, followed by the performance of the enhanced RS-LBS scheme (with hash-grouping).

A. Performance of Basic RS-LBS

The simulation results of the packet reordering probability and mean delay of our approach are shown in Fig. 5 and Fig. 6 respectively. As explained earlier, for evaluating empirical performance we use the conventional measure of packet reordering: a packet A is considered re-ordered if and only if it departs from the output port earlier than another packet in the same TCP flow that arrives at the input port earlier than A. The infinite output buffer size case ("RS-LBS ($W = \infty$)") is when all out-of-order packets wait at the output port until they are in-order. This guarantees a zero packet reordering probability, and its delay can be taken as an upper bound for the re-sequencing delay introduced by our scheme.

Fig. 5 presents the packet reordering probability of our RS-LBS scheme. The packet reordering probabilities are in line with our analysis in Section II. For instance, when the traffic load is $\lambda = 0.9$, for all 4 traffic patterns, the packet reordering probability is around 10^{-3} when the buffer size $W = 10N$ and is around 10^{-4} when $W = 23N$. Interestingly, even with $W = 2$, the packet reordering probability is drastically reduced for light loads compared to the basic LBS.

Fig. 6 presents the average delay. Compared to the other LBS-based schemes, the delay of our RS-LBS scheme looks fairly good for uniform and quasi-diagonal traffic, but this is not the case for log-diagonal and diagonal traffic. We explain this observation as follows:

- The frame-based load-balancing algorithms, such as UFS, FOFF and PF, have very good delay performance for diagonal or log-diagonal traffic since aggregating a full frame when the traffic pattern is "concentrated" is relatively effortless.
- In contrast, the load-balanced stage spreads traffic from the input ports uniformly across the intermediate ports, so a "concentrated" traffic pattern provides no benefit to the basic LBS and our scheme. The queue length distribution at the intermediate ports stay roughly the same no matter what the traffic pattern is (recall that no explicit assumption was made on the traffic matrix for the derivation of the bound on the packet reordering probability in Section II-C2), as does the re-sequencing delay.

In summary, the queuing delay of UFS, FOFF and PF is small for diagonal or log-diagonal traffic while the re-sequencing delay of RS-LBS remains unchanged, as concentrated traffic benefits the frame-based schemes. However, as we shall see below, the introduction of the "hash-grouping" technique (from Section IV) overcomes this weakness.

B. Performance of Enhanced (with Hash-Grouping) RS-LBS

As mentioned earlier, our natural solution can be enhanced by relaxing the packet reordering semantics and implementing the hash-grouping technique. In our simulations, the number of virtual groups for each switch flow is set to $K = 1,000$.

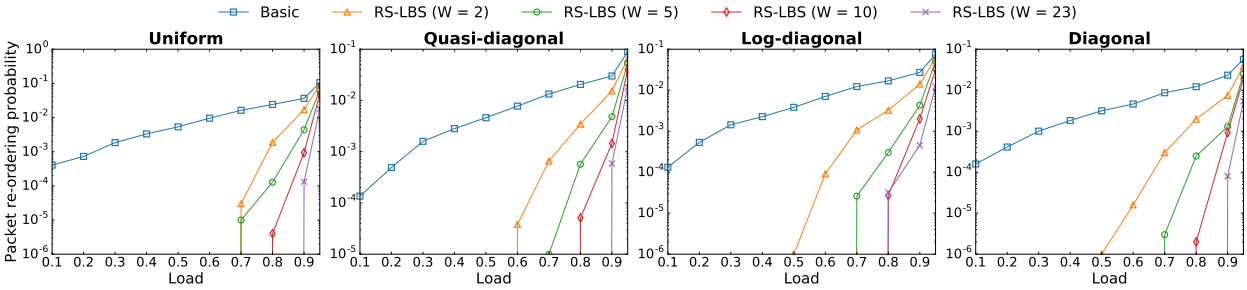

Fig. 5. Packet reordering probability of the RS-LBS scheme, with various buffer sizes W.

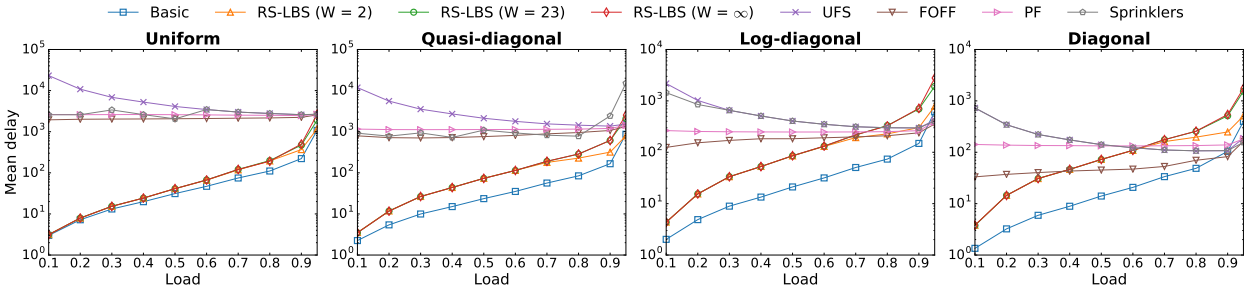

Fig. 6. Average delay of the RS-LBS scheme, with various buffer sizes W, compared to UFS, FOFF, PF, and Sprinklers.

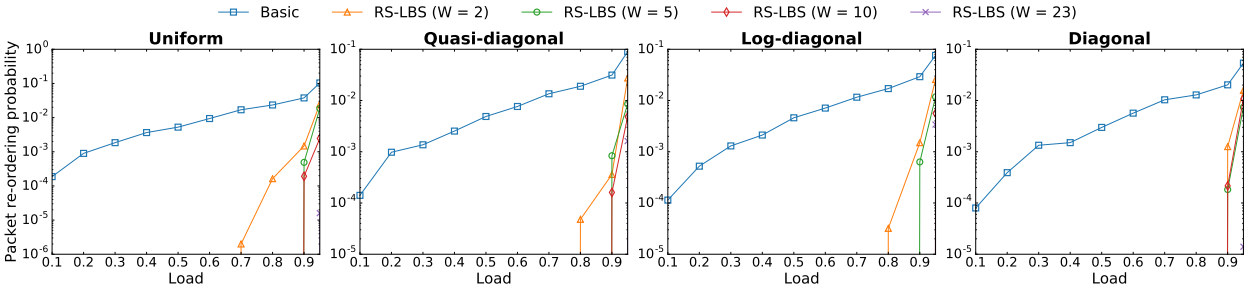

Fig. 7. Packet reordering probability of the RS-LBS scheme with hash-grouping ($K = 1000$ groups).

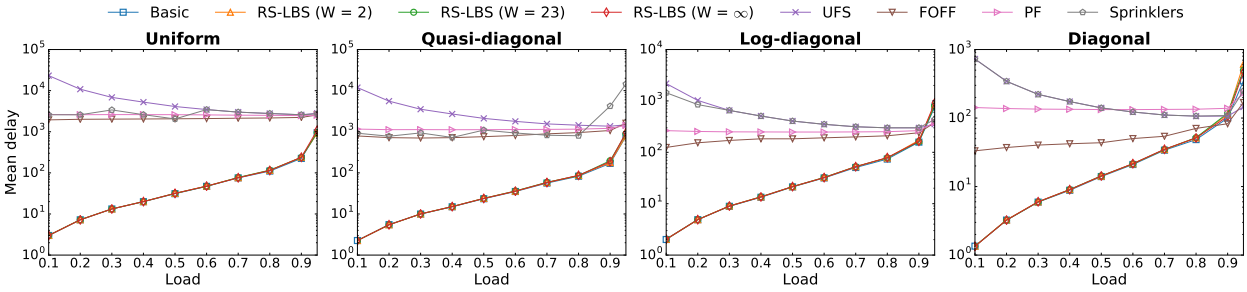

Fig. 8. Average delay of the RS-LBS scheme with hash-grouping ($K = 1000$ groups), compared to UFS, FOFF, PF, and Sprinklers.

Fig. 7 presents the packet reordering probability. We see from the figure that the packet reordering probability is decreased by a factor of approximately 10 times compared to the original scheme. For instance, when the traffic load is $\lambda = 0.9$, the packet reordering probability is around 10^{-4} for a buffer size $W = 10N$ (in contrast to 10^{-3} in Fig. 6). When $W = 23N$, the packet reorder probability is so miniscule for $\lambda = 0.9$ that we did not observe any packet reordering in our simulations.

Fig. 8 presents the average delay, demonstrating a significant improvement over the original approach. It can be seen from Fig. 8 that the average delay is now *almost the same* to the average delay of the basic LBS, no matter how large buffer size we use. In other words, by using the hash-grouping

technique, our scheme achieves an amazing tradeoff: it is expected to bound the packet reordering probability to an arbitrarily small value with a linear, i.e., $O(N)$, buffer size and a negligible average re-sequencing delay. We also see that the average delay of the scheme with hash-grouping on the log-diagonal and diagonal traffic matrices is much lower compared to the basic RS-LBS scheme. Overall, it is clear that hash-grouping significantly improves our original scheme with a minor incremental memory cost.

VI. Related Work

Here, we briefly review existing solutions to the packet reordering problem in the LBS. As already discussed, UFS [5] prevents reordering by requiring that each input port VOQ first accumulates a full-frame of N packets before they are uniformly spread across the N intermediate ports. The main drawback of UFS is that it suffers from $O(N^3)$ delay in the worst-case. FOFF [5] also uniformly spreads full-frames when available. When no full-frame is available, FOFF will serve incomplete frames in a round-robin manner, but it suffers from $O(N^2)$ delay in the worst-case for packet reordering at the output. PF [4] is another method that avoids the need to accumulate full-frames. When no full-frame is available, PF will pad the longest incomplete frame with fake packets to create a full-frame, which is then uniformly spread across the N intermediate ports, just like UFS. However, its worst-case delay is still $O(N^3)$. Recently, an approach called Sprinklers [3] was proposed based on the idea of variable-size striping. Sprinklers uses the arrival rate of packets to a input port VOQ to determine a variable stripe size L rather than always requiring a full-frame. It then only requires the accumulation of L packets in a VOQ before uniformly spreading them across a randomly chosen contiguous block of L intermediate ports, where VOQs with slower arrival rates are given smaller stripe sizes. Finally, packet ordering can be guaranteed via another approach called a Concurrent Matching Switch (CMS) [6], which enforces packet ordering throughout the switch by using a fully distributed LBS approach. While all of these existing approaches can ensure packet ordering, they all add significant complexity to the implementation of the LBS, as well as significant increases in packet delays.

Besides existing works on LBS architectures, recently Dixit et al. [13] studied empirically the effects of random load-balancing on packet ordering in symmetric data center networks such as multi-rooted tree topologies. Their empirical results confirm our observations that the amount of packet reordering that can occur is quite limited. We believe that our analysis framework can be applied to formally bound the amount of packet reordering in that data center setting as well, and we plan to provide analytical bounds for symmetric data center networks in future work.

VII. Conclusion

In this paper, we showed that the amount of packet reordering that can occur in the LBS is actually quite limited. This means that packet ordering can be ensured simply by employing reordering buffers at the switch outputs. In particular, we formally bound the worst-case amount of time that a packet has to wait in these output reordering buffers before it is guaranteed to be ready for in-order departure with high probability, and we prove that this bound is linear with respect to the switch size. This linear bound is significant because previous approaches can add quadratic or cubic delays to the load-balanced switch. Further, we presented hash-grouping strategies at the switch outputs that can further reduce the average packet waiting times at the output reordering buffers. We showed experimentally that our packet reordering approach significantly outperforms existing load-balanced switch architectures. Finally, we believe that our analytical framework and main theoretical results are applicable to other load-balancing applications such as routing over symmetric data center fabrics.

Acknowledgement: This project is supported in part by an NSF INSPIRE grant (1248117), a collaborative NSF grant that includes awards CNS-1423182 and CNS-1422286, and NSF grant CNS-1302197.

References

[1] C.-S. Chang, D.-S. Lee, and Y.-S. Jou, "Load balanced birkhoff–von neumann switches, part i: one-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 611–622, 2002.

[2] C.-S. Chang, D.-S. Lee, and C.-M. Lien, "Load balanced birkhoff–von neumann switches, part ii: multi-stage buffering," *Computer Communications*, vol. 25, no. 6, pp. 623–634, 2002.

[3] W. Ding, J. Xu, J. G. Dai, Y. Song, and B. Lin, "Sprinklers: A randomized variable-size striping approach to reordering-free load-balanced switching," in *ACM CoNext, the 10th International Conference on Emerging Networking EXperiments and Technologies*. ACM, 2014.

[4] J. J. Jaramillo, F. Milan, and R. Srikant, "Padded frames: a novel algorithm for stable scheduling in load-balanced switches," *Networking, IEEE/ACM Transactions on Networking*, vol. 16, no. 5, pp. 1212–1225, 2008.

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

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

[7] J. F. Hayes and T. V. G. Babu, *Modeling and analysis of telecommunications networks*. John Wiley & Sons, 2004.

[8] K. Nakagawa, "On the series expansion for the stationary probabilities of an m/d/1 queue," *Journal of the Operations Research Society of Japan*, vol. 48, no. 2, pp. 111–122, 2005.

[9] R. Bhagwan and B. Lin, "Fast and scalable priority queue architecture for high-speed network switches," in *IEEE INFOCOM*. IEEE, 2000.

[10] A. Ioannou and M. G. Katevenis, "Pipelined heap (priority queue) management for advanced scheduling in high-speed networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 450–461, 2007.

[11] H. Wang and B. Lin, "Per-flow queue management with succinct priority indexing structures for high speed packet scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1380–1389, 2013.

[12] O. Rottenstreich, P. Li, I. Horev, I. Keslassy, and S. Kalyanaraman, "The switch reordering contagion: Preventing a few late packets from ruining the whole party," *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1262–1276, 2014.

[13] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 2130–2138.