This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



# *Promenade*: Proportionally Fair Multipath Rate Control in Datacenter Networks with Random Network Coding

Li Chen, Yuan Feng, Baochun Li, *Fellow, IEEE,* and Bo Li, *Fellow, IEEE*

**Abstract**—In today's datacenter topologies, there exist multiple equal-cost paths between each pair of communicating virtual machines. Yet, splitting flows and routing them along multiple paths may lead to packet reordering, which may affect the performance of TCP. In this paper, we propose *Promenade*, a new protocol that uses random network coding to mitigate the negative effects of packet reordering, while at the same time achieving weighted proportional fairness in bandwidth allocation across different tenants. To achieve weighted proportional fairness when allocating bandwidth to tenants, the problem of rate control is formulated as a convex optimization problem, and *Promenade* uses its distributed solution as a theoretical foundation to design its bandwidth allocation protocol. With our real-world implementation of *Promenade* in the Mininet testbed, we are able to show that *Promenade* is able to achieve weighted proportional fairness in its rate control when individual flows are split into multiple paths.

**Index Terms**—Datacenter networks, Multipath rate control, Network coding, Fairness

## 1 Introduction

In order to provide satisfactory performance to a rapidly increasing number of applications running in the cloud, modern datacenter networks typically adopt multi-rooted tree topologies with full bi-section bandwidth, such as the fat-tree architecture [1]. As full bi-section bandwidth is only provided when all possible paths between a pair of physical servers are fully utilized, equal-cost multipath (ECMP) [2] has been used in today's datacenter networks. Although called multipath, each flow between a pair of servers is actually routed through *one of* the paths using hashing. Although it is possible for two flows between the same server pair to take different paths, recent research shows that ECMP is not sufficient in achieving high network utilization in datacenter networks, as it is possible that two heavy hitter flows are scheduled onto the same path, leading to hot-spots in the network [3].

In order to take advantage of multiple paths to transmit data from a sender to a receiver, one can resort to random packet spraying [4], which spreads the packets from a single TCP flow across multiple paths, by randomly choosing one of the eligible output ports that a packet should be forwarded to at each switch. Unfortunately, different queue lengths and link loads on different paths will result in packets arriving out of order, which will be incorrectly interpreted as packet losses by TCP's fast retransmit mechanism, negatively affecting TCP performance due to unnecessary reductions of the congestion window size [5].

Multipath TCP (MPTCP) has recently been proposed to strip data across different TCP sub-flows with a finer granularity to achieve better load balancing [6]. In MPTCP, each TCP flow is further split into multiple sub-flows, each with its own congestion window, and ECMP may be used to send them along multiple paths. Even though packet reordering across different sub-flows will no longer affect the congestion window sizes in the sub-flows, packet reordering cannot be avoided at the time of merging the sub-flows at a receiver. It has been shown that hundreds of packets may need to be processed with a reordering algorithm at the receiver, before they are delivered to the application layer [7], leading to additional delays due to the reordering process.

To enjoy the benefit of increased throughput by splitting individual flows while mitigating the adverse effects of packet reordering, in this paper, we advocate the use of random network coding [8], [9] in datacenter networks, implemented as a shim layer between the applications and the network interface. With random network coding, packets belonging to the same flow are split into multiple sub-flows, and then forwarded along different paths in a coded fashion. Particularly, each packet is encoded with a vector of randomly chosen coefficients, which will be embedded in the packet to be transmitted to the receiver. In principle, once a receiver has collected a sufficient number of linearly independent packets, it is able to recover the original packets in order, which indicates


- *Li Chen is with the Department of Computer Science, University of Louisiana at Lafayette, USA. E-mail: li.chen@louisiana.edu*
- *Baochun Li is with the Department of Electrical and Computer Engineering, University of Toronto, Canada. E-mail: bli@ece.toronto.edu*
- *Bo Li is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China. E-mail: bli@cse.ust.hk*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



that the order of receiving the coded packets no longer matters. We believe that with a carefully designed rate control mechanism, the use of random network coding leads to a simpler way of transmitting packets along multiple paths in datacenter networks.

A salient advantage of using random network coding is that it allows us the opportunity to redesign the rate control algorithm from scratch, which corresponds to solving a bandwidth allocation problem from a global perspective, rather than using local information only. As this opportunity arises, we believe that a new rate control algorithm should be designed with the objective that different tenants sharing the same datacenter can fairly share the network bandwidth. Since the network bandwidth is typically shared in a best-effort fashion, we wish to provide *weighted proportional fairness* across multiple tenants, with a weight assigned to each tenant. We formulate the problem of computing flow rates along multiple paths in the datacenter network as a convex optimization problem, with weighted proportional fairness across different tenants as the objective. Our optimization problem is amenable to a distributed solution based on primal-dual updates.

The original highlight of this paper is the design of *Promenade*, a new protocol designed to utilize the advantage of random network coding, and built upon the theoretical foundation of our problem formulation and its distributed solution. *Promenade* is implemented in a shim layer at a sender, and is topology-agnostic. By controlling the rate of transmitting coded packets over a sliding coding window, *Promenade* incorporates both random network coding and a new rate control algorithm into its protocol. To prevent tenants from bypassing the rate control protocol in *Promenade*, the shim layer runs in the virtualization network stack, where it is well isolated from tenant code. Given a network weight for each tenant, *Promenade* first computes the weight per communicating VM pair based on its traffic amount, and then allocates the bandwidth on multiple equal cost paths iteratively, such that the sum of bandwidth each communicating VM pair receives in the network is proportional to its weight. With end-to-end rate control, *Promenade* scales up with the number of communicating VMs easily. We evaluate the performance of *Promenade* within emulated fat-tree datacenter topologies, using Mininet [10] as our network emulation testbed.

The remainder of this paper is organized as follows. In Sec. 3, we motivate the use of random network coding to solve the packet reordering problem, and present its potential challenges. In Sec. 4, we formulate the rate allocation problem as a convex optimization problem with an objective of providing weighted proportional fairness among multiple tenants. Based on the distributed solution from Sec. 4, Sec. 5 presents the design of *Promenade* in detail. In Sec. 6, we evaluate the performance of *Promenade* in an emulated fat-tree datacenter topology. We discuss related work and conclude the paper in Sec. 2 and Sec. 7, respectively.

## 2 Related Work

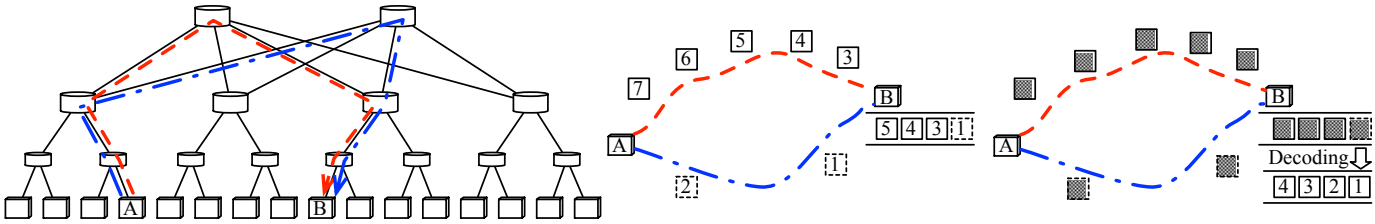
With the mainstream acceptance of cloud computing, a substantial amount of research efforts have been devoted towards improving the load balance and overall throughput in datacenter networks. Different from uncertain networks where the connectivity determination problem needs to be solved [11], [12], datacenter networks have deterministic topologies and each pair of source and destination is connected with multiple paths. With respect to flow-level traffic splitting, Hedera is proposed to adaptively schedule active flows to non-conflicting paths to maximize the aggregated network utilization [13]. Similarly, Mahout is proposed to identify elephant flows using end-host mechanisms, and then schedule flows dynamically [14]. Prior to them, VL2 and Monsoon use per-flow Valiant Load Balancing [1], [15]. As effective as they are on performing load balancing within a datacenter network, none of them actually splits individual flows across multiple paths.

MPTCP represents the first effort of splitting traffic at a sub-flow granularity [7]. It splits an individual TCP flow into multiple sub-flows, and routes them across different paths using ECMP. Although MPTCP is shown to be able to improve the throughput substantially, and has been proposed to be implemented in both the Internet [16] and datacenter networks [6], it has not been widely deployed yet. Recent research has revealed that it suffers from three major challenges. *First,* Khalili *et al.* have shown that MPTCP is not pareto-optimal, meaning that upgrading some TCP users to MPTCP can potentially reduce the throughput of others, yet without any benefit to the upgraded users. They also pointed out that MPTCP users could become excessively aggressive towards TCP users [17]. *Second,* as packets belonging to different subflows follow different paths, hundreds of packets may need to be reordered when merging the subflows at a receiver [18]. *Finally,* MPTCP is not efficient for the short flows that dominate datacenter networks, due to its high signaling and connection establishment complexity [7], [19].

With *random packet spraying* [4], packets of every flow are randomly assigned to one of the available shortest paths to the destination, and traditional TCP is used as the transport protocol. Compared to MPTCP, the simplicity of random packet spraying makes it a good candidate for short flows in datacenter networks, and Dixit *et al.* [4] has shown using a small-scale experimental testbed with a fat-tree topology that TCP is able to tolerate the reordered packets without much performance degradation, since multiple equal-cost paths exhibit similar queue lengths with symmetric topologies. *Digit-Reversal*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems






(a) Splitting a flow over multiple paths in a multi-rooted datacenter topology (b) Without network coding (c) With network coding

Fig. 1: Random network coding is able to solve the problem of packet reordering as each flow is split to be transmitted over multiple paths in datacenter networks.

*Bouncing* [20] also achieves per-hop balancing through per-packet spreading of traffic in a round robin fashion, which would become inefficient with asymmetric topologies due to link failures and significant load variation [21]. CONGA [22] splits traffic into *flowlets*, which are routed based on load estimates using in-network congestion feedback. Presto [23] divides flows into *flowcells*, which are source-routed without load awareness. DRILL [24] performs per-packet load balancing at each switch using randomized algorithms based on local queue occupancies to distribute load. Different from these works, our *Promenade* implements a simple rate control protocol in the context of multipath transmissions, using random network coding to eliminate the need to mitigate the adverse effects of both packet reordering and packet losses.

With respect to fairness in sharing datacenter networks, a substantial amount of research attention has been attracted recently [25]–[35]. Lam *et al.* proposed NetShare, where each tenant gets a predefined network weight, which are used to achieve weighted max-min fairness on congested links in the network [25]. Its implementation requires the switches to support weighted fair queueing with aggregated queues. Different from NetShare, Seawall is proposed based on per-VM weights [26]. It stipulates that on all network links, the share of bandwidth obtained by a VM serving as the traffic source is proportional to its weight. However, its rate control algorithm is only an extension of a simple strawman approach that is not based on a theoretical foundation. In comparison, our rate control protocol in *Promenade* is based on a rigorous formulation of the problem, with the objective of achieving an optimal solution during our design. Furthermore, *Promenade* is designed to utilize per-VM-pair weights rather than per-tenant or per-VM weights.

## 3 The Use of Random Network Coding: Motivation and Solutions

With the observation of the packet reordering problem in multipath datacenter networks, we are motivated to mitigate its adverse effect by exploiting the salient benefits brought by random network coding. In what follows, we illustrate the performance degradation caused by packet reordering, introduce the favorable properties of random network coding and present the design of our solution called *Promenade*.

### 3.1 Packet Reordering in Multipath Networks

Most application traffic — in both large (elephant) and small (mice) flows — in the datacenter networks today requires reliable and in-order delivery of packets. This fits well with the design of TCP as a transport protocol, which is robust against packet losses and a small amount of packet reordering. However, when packets belonging to a single flow are spread and transmitted along multiple paths, the problem of packet reordering becomes more severe, as packets following different paths may enter queues of different lengths, and the larger the queue length differential, the more severe packet reordering is.

In Fig. 1, we show an illustrative example of the packet reordering problem when individual flows are split along multiple equal-cost paths in a datacenter network. Today's operational datacenter networks are typically constructed with a multi-rooted tree topology, as shown in Fig. 1 (a), or a fat-tree topology [1] which employs more network switches, reduces the bandwidth oversubscription and achieves a full bi-section bandwidth. In these topologies, for any given pair of VMs, A and B, on two physical machines, there exists multiple equal-cost paths, as illustrated by the red and blue dashed lines in the figure, respectively. Now suppose an individual flow from VM A to VM B is split along these two paths to achieve a better load balance, as shown in Fig. 1 (b), B may receive the first four packets in an order of $1, 3, 4, 5$, due to different queue lengths and sub-flow rates on the two paths. Since a TCP sender interprets three duplicate acknowledgments as a packet loss, and activates fast retransmit with a reduced sliding window size, the throughput between A and B is unnecessarily affected by packet reordering (but without any lost packets).

### 3.2 The Benefits of Random Network Coding

With random linear network coding [8], [9], packets in a flow are coded within a *coding window*. A coding window

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



consists of a certain number of packets, say $n$ (called the coding window size). These packets are denoted as $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$, where each packet has a fixed number of bytes, $k$. To code a new coded packet $x_j$ within a coding window, the source first independently and randomly chooses a set of coding coefficients $[c_{j1}, c_{j2}, \cdots, c_{jn}]$ in the Galois Field $GF(2^8)$, one for each original packet. It then produces one coded packet $x_j = \sum_{i=1}^{n} c_{ji} \cdot b_i$. The destination decodes as soon as it has received $n$ linearly independent coded packets $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$. It first forms an $n \times n$ coefficient matrix $\mathbf{C}$, using the coefficients of each packet $b_i$, which are embedded in the packet. Each row in $\mathbf{C}$ corresponds to the coefficients of one coded packet. It then recovers the original packets $\mathbf{b} = [b_1, b_2, \ldots, b_n]^T$ as $\mathbf{b} = \mathbf{C}^{-1}\mathbf{x}$. Gauss-Jordan elimination is used in such a decoding process, performed progressively as coded packets are being received. The inversion of $\mathbf{C}$ is only possible when its rows are linearly independent, *i.e.*, $\mathbf{C}$ is full rank.

The benefit introduced by random network coding to mitigate the adverse effects of packet reordering may seem intuitively simple at first glance. If random network coding is used instead, packets between A and B are transmitted in a coded form as shown in Fig. 1 (c), produced with random linear codes within a coding window of 4 packets. In this simple example, no matter in what order the four coded packets are received at the receiving side B, the four original packets can be decoded in the order $1, 2, 3, 4$, as long as the coded packets are decodable, in that they are linearly independent with each other. To put it simply, due to the rateless property of random linear codes, the order of receiving coded packets, as well as any potential packet losses, no longer matter when random network coding is used. As long as the receiving side has received a sufficient number of coded packets, the same number of original packets can be recovered in order through the decoding process.

Granted, the use of random network coding does not have the ability to reduce the potentially significant number of reordered packets as they follow different paths with a large queue length differential. Coded or not, packets will be reordered due to the different queueing delays. The disadvantage of network coding is also clear: they introduce an additional overhead, in the form of both coding complexity and decoding delays. The *coding complexity* may not be a concern: with modern multi-core CPUs commonly used in physical servers in operational datacenters, it has been shown that random network coding can be performed with good performance [36], especially for small sizes of the coding window. On the other hand, the *decoding delay*, which is the amount of time a packet needs to wait in a queue for later packets to arrive for successful decoding, may be a problem if the size of the coding window is too large. With full knowledge of the overhead induced by random network coding, what, after all, are its main benefits with respect to mitigating the problem of packet reordering? We have identified three main advantages with the use of random network coding.

***There is no longer a need to use active queue management schemes on the switches.*** Dixit *et al.* [4] have shown that, with a large queue length differential along different paths, the performance of TCP suffers significantly. It then proposed to use active queue management schemes to provide early feedback as the queue length builds up due to congestion, with the hope of equalizing the length of queues along different paths. While using active queue management on switches may indeed mitigate the problem of packet reordering, the feature may not be readily available on off-the-shelf edge switches. With random network coding on the end hosts, we do not require active queue management on any of the switches in a datacenter network.

***The severity of packet reordering no longer matters.*** Since long (elephant) flows may be negatively affected by the burstiness of small (mice) flows in datacenter networks, we are doubtful that active queue management schemes will be sufficiently effective, in that the queue lengths are equalized to an extent that does not affect TCP performance (*i.e.*, no more than three duplicated acknowledgments). With random network coding, however, the severity of packet reordering no longer matters. This provides us a substantial amount of additional freedom to redesign the rate control protocol, so that the flow rate on different paths can be computed and assigned by the new protocol without fear from packet reordering.

***A flow no longer needs to be striped to be transmitted along different paths.*** In MPTCP, a flow will need to be striped to different sub-flows and then transmitted along different paths, and a packet belongs to only one of the sub-flows. If a packet is lost in its sub-flow, it will need to be retransmitted, coupled with a reduced window size on the TCP sender. With random network coding, there is no particular sequence among coded packets in the same coding window, and packets can be transmitted along any of the eligible paths. Packet losses on one of the paths can be easily remedied by transmitting additional coded packets along *any subset* of the eligible paths.

Given these advantages, we argue that the use of random network coding gives rise to the opportunity of designing a new rate control protocol in the context of multipath transmissions, with no need to mitigate the adverse effects of both packet reordering and packet losses with intricate protocols. It leads to a simpler protocol design, in that packets being transmitted on all the eligible paths are able to "collaborate" with one another perfectly, at any flow rate on each path.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



### 3.3 Implementing Random Network Coding as a Shim Layer

It has been proposed in the literature that random network coding is used in coordination with TCP as the transport protocol [37], referred to as TCP/NC. The gist of the idea is that, implemented as a layer below TCP, a sender transmits random linear combinations of packets in its sliding congestion window, and advances the window as it receives an acknowledgment from the receiver. These acknowledgments are in the form of the *degree of freedom* of the coding buffer that the receiver holds: an original packet is acknowledged as it is received in a coded form that is linearly independent (*i.e.*, useful towards the decoding process), even before decoding is complete with Gauss-Jordan elimination. Due to the overhead of carrying coding coefficients in the header, the coding window is a relatively small subset of the TCP sliding window (TCP/NC [37] uses a size of 3 in its experiments).

The design principle within TCP/NC can be readily used in *Promenade*, as it is designed for unicast flows, which is predominant in datacenter traffic. Similar to TCP/NC, in the design of *Promenade*, random network coding is implemented as a *shim layer* between the applications and the network interface, to be deployed to all servers in the datacenter. Yet, different from TCP/NC that relies on TCP to control the flow rates, *Promenade* uses a similar design as Seawall [26], in that all traffic in the datacenter network is to be transmitted via *rate controlled* logical tunnels, implemented within a shim layer that intercepts all the packets entering or leaving the physical server. Such rate controlled tunnels are most suitable for UDP flows, as a full burst UDP flow immediately uses all the rate that the tunnel allows. For TCP flows, its congestion control needs to be deferred to the *Promenade* shim layer, in that each TCP flow queries the rate controller in the shim to see whether it is allowed to send.

Within the shim layer on the sender, we maintain a coding buffer with a sliding coding window for each flow. As shown in Fig. 2, In the sliding coding window, the sender shim transmits outgoing packets in their coded form, along any of the eligible paths. The paths can be established using different sender port numbers, hashed to different paths between the sender and the receiver by ECMP, supported by most modern switches used in datacenters. Alternatively, they can also be determined dynamically using random packet spraying [4] if the switches support the mechanism. Upon receiving an acknowledgment from the receiver, the sender shim advances the sliding coding window, removes the acknowledged packet from its coding buffer, and hand over the acknowledgment to the sender in the application. On the receiver side, the receiver shim sends an acknowledgment back to the sender shim when it receives a *seen* packet (as formally defined in [37]), where the number of seen

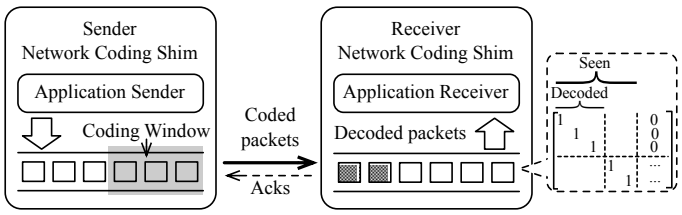



Fig. 2: Implementing random network coding as a shim layer. The sender shim sends outgoing packets in their coded form, and advances the coding window upon receiving an acknowledgment from the receiver. The receiver acknowledges all *seen* packets.

packets is equal to the degrees of freedom of all the received packets that have not yet been decoded so far. Once the receiver shim completely decodes a packet, it will be delivered to the application's receiver. By implementing random network coding within the shim layer, packets will be acknowledged to the sender even if their coded counterpart arrives out of order. This substantially mitigates the problem of duplicate acknowledgments due to packet reordering.

As in TCP/NC, we impose a maximum size for the coding window, for the sake of managing the coding complexity and the overhead of the coding header. One question remains that may be potentially tricky: what is the most appropriate value for the maximum coding window size in the sender shim? The coding window size determines the maximum ability to tolerate packet reordering, and as a result, it would be beneficial for the size to be as large as possible, in order to tolerate as many reordered packets as possible. That said, a large coding window leads to a long decoding delay at the receiver side, as well as a high coding complexity.

In *Promenade*, we use a small coding window of 4-8 packets, since we wish to guarantee that the network coding engine is able to keep up with the bandwidth of the network interface. For this reason, we use our heavily optimized implementation of random linear codes, with both multi-threading and SIMD (single instruction, multiple data) acceleration to further relieve the stress on the CPUs. Our real-world experiments on coding performance will be presented in Sec. 6.

## 4 Rate Control with Proportional Fairness

### 4.1 Design Objectives

Towards the design of a new protocol at the shim layer in *Promenade* to split flows among multiple equal-cost paths in the datacenter network and control their flow rates, we aim to achieve two design objectives: *fairness among tenants* and *simplicity*.

***Fairness among tenants.*** The use of random network coding within the shim layer introduces more freedom for rate control on each flow before packets leave the physical

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



server. This is a blessing rather than an inconvenience, as we may use this opportunity to achieve more desirable fairness objectives than what TCP provides.

Unlike other resources such as CPU or memory in a VM, inter-VM bandwidth in datacenter networks today is free of charge, and is typically shared in a best-effort fashion. It has been well understood that the performance of a cloud application heavily depends on its received share of inter-VM bandwidth, *i.e.*, when the traffic flow of an application is throttled due to congestion, it takes more time to complete the same task, which then translates to a higher cost to the tenant on reserving VMs in the datacenter network. To encourage more tenants to use a public cloud service, it is natural that the notion of fairness should be enforced at the tenant level when it comes to sharing bandwidth resources in datacenter networks. Since the performance of a tenant's application depends on the aggregated rate of all initiated flows between its communicating VMs, traditional TCP, which only guarantees fairness between flows or sub-flows, is clearly not sufficient.

***Simplicity.*** As a protocol in the datacenter network, our rate control algorithm should be conceptually *simple* to operate. Each sending VM should only need to adjust its sending rate adaptively in a *distributed fashion*, based on the feedback from a central monitor. The changes required in the datacenter network should be minimized, in that only a *shim layer* has to be installed in each of the VMs to perform rate control and send feedback, and neither the network topology nor the hardware needs to be modified. In addition, the rate control algorithm should not only scale with the number of VMs naturally, but also with the number of flows. As fairness is provided at the tenant level, and every individual flow may be split into multiple sub-flows along different paths, we believe that the flow rate between each VM pair should be optimized in an aggregated fashion. In other words, our rate control algorithm will determine the optimal rate for traffic from one VM to another, no matter how many flows are initiated in between.

### 4.2 The Problem of Rate Control with Proportional Fairness

In a nutshell, the key idea in the design of our rate control algorithm in *Promenade* is to take full advantage of splitting individual flows in the inter-VM traffic, so that proportional fairness is achieved across all tenants. We precede our protocol design with a theoretical formulation of this problem.

Since we are trying to control the rate between each VM pair to achieve a certain tenant-level proportional fairness, a critical question is how should we assign weights in the network. Should it be per-tenant weight, or per-VM pair weight? Our design in this paper has intentionally left the decision *open* with respect to how weights should be assigned, as existing literature has already covered this complementary problem quite well [27].

We propose an intuitive alternative in this paper. Each tenant is associated with a weight based on the number of communicating VMs it has initiated in the datacenter, *i.e.*, the total payment that the tenant pays for inter-VM data transmission. Then, each tenant itself will divide and allocate its weight to all of its communicating VM pairs, based on its application traffic pattern. We believe that it is necessary for tenants to have the flexibility to dynamically adjust the local weights of its communicating VMs, as recent datacenter traffic studies have shown that tremendous variation in the communication matrix exists over space and time [3], [38]. Traffic from one VM to another exhibits many small transactional-type RPC flows (*e.g.*, search results), as well as a few large transfers (*e.g.*, backups and propagations) during different time periods. The weights for those VM pairs, hence, should be adjusted accordingly by the tenant.

In the design of our rate allocation algorithm, we expose the following simple abstraction. Given a *network weight* $w_i$ for traffic from a VM X to another VM Y, which is considered as a session $i$ in this paper, we ensure that along all equal-cost network paths, the aggregated share of bandwidth obtained by the VM pair is proportional to its weight. The fairness criterion we considered in this paper is *weighted proportional fairness*, which is one of the most commonly used fairness criteria. To be exact, our objective is to find a set of rates $\mathbf{r}$ for any pair of communicating VMs along each of its equal-cost path in the datacenter network, such that: (1) $\mathbf{r}$ is feasible, that is, it is non-negative and obeys the link capacity constraint; and (2) for any other feasible set $\mathbf{r}'$, the aggregate of proportional changes is zero or negative [39]:

$$\sum_{\forall i} \frac{r_i' - r_i}{r_i} \leq 0. \tag{1}$$

Kelly *et al.* have proved in their seminal work [40] that the optimal rate allocation in achieving proportional fairness is equivalent to maximizing the social utility in the network, with $\log$ as the utility function. If we consider a datacenter network as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ indicates the set of physical servers, and $\mathcal{E}$ indicates the set of directed edges inter-connecting them, the rate control problem in the datacenter network can be formulated as follows:

$$\max_{r_i(p)} \quad \sum_{\forall i} w_i \log\Big(\sum_{p \in \mathcal{P}_i} r_i(p)\Big) \tag{2}$$

$$s.t. \quad \sum_{\forall i} \sum_{p \in \mathcal{P}_i(e)} r_i(p) \leq C(e), \forall e \in \mathcal{E}, \tag{3}$$

$$r_i(p) \geq 0, \forall i, \forall p, \tag{4}$$

where $\mathcal{P}_i$ represents the set of equal-cost paths in session $i$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



TABLE 1: Important mathematical notations

| Notation | Definition |
|---|---|
| $\mathcal{V}$ | the set of physical servers in a datacenter |
| $\mathcal{E}$ | the set of directed edges connecting servers in $\mathcal{V}$ |
| $C(e)$ | the available capacity on edge $e$ |
| $w_i$ | the network weight of session $i$ |
| $\mathcal{P}_i$ | the set of equal-cost paths in session $i$ |
| $r_i(p)$ | the allocated flow rate in session $i$ along a path $p \in \mathcal{P}_i$ |
| $\mu_e$ | the dual cost of an edge $e$ |
| $l_i(p)$ | the dual cost for flow along path $p$ in session $i$ |
| $\Lambda(e)$ | the load of edge $e$ |

In the formulation, the optimization variable $r_i(p)$ is the allocated flow rate in each session $i$ on every path $p$, where $p \in \mathcal{P}_i$. $\mathcal{P}_i(e)$ is the set of paths in $\mathcal{P}_i$ that cover edge $e$, and $C(e)$ is the available edge capacity. The objective is to maximize the social welfare of all sessions with a logarithm utility function, as we have discussed before. Constraints (3) and (4) state that the allocated rate can not exceed the edge capacity on all the edges in the network, and are non-negative. Important notations used in this paper are summarized in Table 1.

The Hessian matrix of the objective function $f$ is the square matrix of the second-order partial derivatives $\frac{\partial^2 f}{\partial r_i(p)\partial r_j(q)}$ of $f$, which is derived as:

$$\mathbf{H} = \begin{bmatrix} \frac{-w_1\mathbf{J}_{|\mathcal{P}_1|}}{(\sum_{p\in\mathcal{P}_1} r_1(p))^2} & \mathbf{0}_{|\mathcal{P}_1|\times|\mathcal{P}_2|} & \cdots & \mathbf{0}_{|\mathcal{P}_1|\times|\mathcal{P}_I|} \\ \mathbf{0}_{|\mathcal{P}_2|\times|\mathcal{P}_1|} & \frac{-w_2\mathbf{J}_{|\mathcal{P}_2|}}{(\sum_{p\in\mathcal{P}_2} r_2(p))^2} & \cdots & \mathbf{0}_{|\mathcal{P}_2|\times|\mathcal{P}_I|} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{|\mathcal{P}_I|\times|\mathcal{P}_1|} & \mathbf{0}_{|\mathcal{P}_I|\times|\mathcal{P}_2|} & \cdots & \frac{-w_I\mathbf{J}_{|\mathcal{P}_I|}}{(\sum_{p\in\mathcal{P}_I} r_I(p))^2} \end{bmatrix}$$

where $I$ is the total number of sessions, $|\mathcal{P}_i|$ represents the total number of available paths for session $i$, $\mathbf{J}_n$ represents the $n \times n$ all-ones matrix where every element is one, and $\mathbf{0}_{m\times n}$ represents the $m \times n$ zero matrix. Given any non-zero column vector $\mathbf{a} = [a_{11}, a_{12}, \ldots, a_{1|\mathcal{P}_1|}, \quad \ldots, \quad a_{I1}, a_{I2}, \ldots, a_{I|\mathcal{P}_I|}]^T$ with real elements, we have:

$$\mathbf{a^T H a} = \frac{-w_1(\sum_{p\in\mathcal{P}_1} a_{1p})^2}{(\sum_{p\in\mathcal{P}_1} r_1(p))^2} + \cdots + \frac{-w_I(\sum_{p\in\mathcal{P}_I} a_{Ip})^2}{(\sum_{p\in\mathcal{P}_I} r_I(p))^2},$$

which is obviously negative as the weight $w_i$ is positive. Therefore, the Hessian matrix is negative definite and thus the objective is concave [41].

As an optimization problem with a symmetric, non-decreasing, and concave objective function, (2) can be solved by a distributed algorithm based on the primal-dual method [41]. We use $\mu_e$ to denote the dual cost of an edge $e \in \mathcal{E}$, and $l_i(p)$ as the dual cost for the flows along a path $p$ in session $i$, where $l_i(p)$ is given by $\sum_{e\in p,\, p\in\mathcal{P}_i} \mu_e$ for a given path $p$ in session $i$. Define $\Lambda_e$ as the load on an edge $e$, *i.e.*, $\Lambda_e = (\sum_i \sum_{p\in\mathcal{P}_i(e)} r_i(p))/C(e)$, and use $\Lambda_e^t$ to denote their values at the $t$-th iteration. The algorithm can be stated in Algorithm 1, where the initialization for the parameters in lines 1-7 follows the convention [42] to guarantee the algorithm feasibility and running-time properties.

**Algorithm 1** *Distributed solution to problem* (2).

**Input:**
The number of sessions: $n$; the number of edges: $m$; weight $w_i$ and the set of paths $\mathcal{P}_i$ for each session $i$; link bandwidth capacity $C(e)$ at each edge $e \in \mathcal{E}$.

**Output:**
Rate allocation $r_i(p)$ for each session $i$ along each of its paths $p \in \mathcal{P}_i$.

1: *Initialization*. $R = \frac{\max_e C(e)}{\min_e C(e)}$, $\rho = \max\{n, m, R\}$ and $\delta = 12\ln\rho + 2$.
2: **for** Each edge $e \in \mathcal{E}$ **do**
3: $\quad \mu_e^0 = \frac{\delta}{2\rho^3}$
4: **end for**
5: **for** Each flow in every session $i$ **do**
6: $\quad r_i(p)^0 = \frac{\min_e C(e)}{2n}$
7: **end for**
8: $t = 0$
9: *Iteration*. Each session computes the shortest path $p^*$ from the sending VM to the receiving VM, *i.e.*, the path with the minimum dual cost $l_i(p)$.
10: **while** there is any flow across all sessions with $l_i(p^*)^t < 1$ **do**
11: $\quad t = t + 1$
12: $\quad$ **for** Each session $i$ **do**
13: $\quad\quad$ **if** $l_i(p^*)^{t-1} < 1$ **then**
14: $\quad\quad\quad r_i(p^*)^t = r_i(p^*)^{t-1} + w_i\frac{\min_e C(e)}{n\delta}$
15: $\quad\quad$ **else**
16: $\quad\quad\quad r_i(p^*)^t = r_i(p^*)^{t-1}$
17: $\quad\quad$ **end if**
18: $\quad$ **end for**
19: $\quad$ **for** Each edge $e \in \mathcal{E}$ **do**
20: $\quad\quad \mu_e^t = \mu_e^{t-1}(1 + \delta(\Lambda_e^t - \Lambda_e^{t-1}))$
21: $\quad$ **end for**
22: **end while**

The algorithm works in the following way. Initially, the dual costs on all the edges ($\mu_e^0$) and the sending rates of all the sessions ($r_i(p)^0$) are set to be very small, as illustrated in lines 2–7. Each session $i$ iteratively increases its flow rate on the shortest path $p^*$ in a weighted fashion, as expressed in line 14, until its dual cost ($l_i(p^*)^t$) becomes no less than 1 at a certain iteration $t$. At this point, the session finds out that even the shortest path between its transmission VM pair is too expensive, so it stops increasing its flow rate, as shown in line 16. Meanwhile, the dual cost at each edge keeps on increasing based on the equation in line 20, until the iteration stops when there is no room for any session to increase its flow rate. It has been proved that Algorithm 1 is feasible and $O(\log\rho)$-approximation guaranteed for all canonical utility functions, with a polynomial number of iterations [42].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



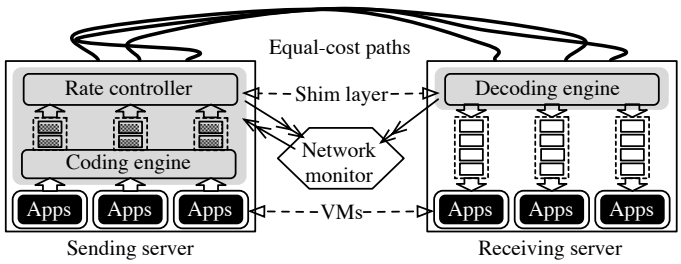
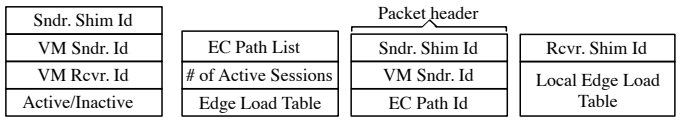



Fig. 3: An overview of the protocol design in *Promenade*.




(a) From sender shim to network monitor (b) From network monitor to the sender shim (c) From sender shim to receiver shim (d) From receiver shim to network monitor

Fig. 4: The content of packets.

## 5 Promenade: Protocol Design

To achieve the optimal rate allocation, *Promenade* relies on communication among the shim layers at all the physical servers, as well as a central network monitor, which collects and monitors the current network condition. In this section, we present a detailed description of our protocol design.

In order for the design of our rate control protocol to be topology-agnostic and require no hardware changes, similar to random network coding, our rate control protocol is also deployed in a shim layer on all servers in the datacenter network. When random network coding and rate control are combined, *Promenade* intercepts all the packets entering and leaving the server, performs random network coding (both encoding and decoding) in its coding engine, and then transmits them between VMs. As shown in Fig. 3, packets are first encoded at the shim layer on the sending server, with one queue (serving as the coding buffer) associated with packets from one session, *i.e.*, traffic from one VM to another. The coded packets will then go through a rate controller that determines the allowed rate on each path in every session, before they are sent to the destination VMs via multiple equal-cost paths. The shim layer at the receiving server decodes the received packets, and directs the decoded packets to their corresponding VMs.

Through the interpretation of both primal and dual variables in Algorithm 1, it appears that the number of sessions as well as the current edge load along its equal-cost paths are required to determine the optimal sending rate in each session. As a result, there is a centralized *network monitor* enabled in the datacenter network[1]. As Fig. 3 shows, both the sending and receiving sides send feedback periodically to the network monitor, which stores a global view of the current network condition. The rate controller corresponding to each physical server fetches the network information from the network monitor to adapt the allowed rate in each session. The rate controller takes the network weights of each session as input, works independently of each other, and together ensures that the bandwidth allocation across different tenants is proportionally fair. In what follows, we will discuss the communication among different components and the protocol in each component in detail.

***From the sender shim to the network monitor.*** The purpose for the sender shim to communicate with the network monitor is that each rate controller will need to know the number of active sessions in the network when it determines the optimal rate for its own sessions. In order to obtain the global information, the shim at the sender sends feedback packets at regular time intervals, containing the sender shim id, the VM sender id, the VM receiver id, and its current active/inactive status. The VM sender and receiver ids are used to uniquely identify a session, and the network monitor uses this feedback information to update $n$, which is the number of sessions in the datacenter network.

More specifically, at the beginning of each time interval, the central monitor collects feedback packets from the sender shims. If the session <VM Sndr. Id, VM Rcvr. Id>, which corresponds to session index $i$ in Sec. 4, is new and the session status is active, the monitor records the session, and increases $n$ by 1; if the session has been recorded before and its status now changes to inactive, the network monitor decreases $n$ by 1, as summarized in Algorithm 2.

**Algorithm 2** *Update the number of sessions*

```
Check <VM Sndr. Id, VM Rcvr. Id> in the session list
if <VM Sndr. Id, VM Rcvr. Id> is new then
    Add it to the session list
    if <VM Sndr. Id, VM Rcvr. Id> is active then
        n = n + 1 ← the number of active sessions
    end if
else
    if <VM Sndr. Id, VM Rcvr. Id> is inactive then
        n = n − 1
        Remove it from the session list
    end if
end if
```

***From the network monitor to the sender shim.*** When the sender VM is ready to send packets out, the shim on the host machine will fetch information from the network monitor for the purpose of rate control. The feedback packets sent from the network monitor to the sender shim has the content shown in Fig. 4 (b). The monitor

1. Here we assume that the datacenter does not have OpenFlow-enabled switches. We will briefly discuss the possibility of using OpenFlow-enabled switches at the end of this section.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



will provide a set of equal-cost paths, *i.e.*, $\mathcal{P}_i$, based on the current network condition. It also informs the sender shim the current number of active sessions, $n$, as well as the edge load along all equal-cost paths, $\Lambda_e$, to the sender shim, such that the rate controller within the shim can use this information to adjust the optimal sending rate based on Algorithm 1. The sender shim will fetch the updated information at the beginning of every iteration.

***From the sender shim to the receiver shim.*** Since the network monitor needs to have the global view of the current edge load in the datacenter network, it is designed to rely on the reports of receiving rates from receiver shims in *Promenade*, where dropped packets are no longer counted. One challenge is that packets are transmitted along multiple equal-cost paths to each receiver shim, and in order to obtain the local view of edge loads at receiver shims, it is necessary for the receiver shim to know which path the packet is transmitted from. In *Promenade*, every packet is stamped its equal-cost path id (EC Path Id) in a header before it is sent out in the sender shim. Once the receiver shim receives a packet, it can updates the path load statistics in the path load table easily, with a simple operation:

$$\text{PathBytes[<VM Sndr. Id, VM Rcvr. Id>][EC Path Id] += packet size.} \quad (5)$$

***From the receiver shim to the network monitor.*** As we have discussed, upon receiving packets, the receiver shim is able to obtain a local view of the traffic amount, *i.e.*, a PathBytes table. Divided by the period of time, the receiver shim is able to obtain the local edge load along every path in each session, which is denoted by $r_i(p)$ in Sec. 4. The receiver shim also measures the local edge load periodically, and sends back the information via a feedback packet. To measure the local edge load, the algorithm that the receiver shim follows is summarized in Algorithm 3. After collecting the feedback packets from receiver shims, the network monitor is able to readily obtain the global edge load condition $\Lambda_e$.

**Algorithm 3** *Update the local edge load*

1: At every period $T$
2: Resets the local edge load table to $0$
3: **while** $t < T$ **do**
4: Update the path load statistics upon receiving packets
5: **end while**
6: Obtains the local edge load table by PathBytes[<VM Sndr. Id, VM Rcvr. Id>][EC Path Id]$/T$
7: Sends the local edge load table to the network monitor

So far, our protocol design in *Promenade* has assumed that the switches in the datacenter network are not OpenFlow-enabled. As operational datacenters are increasingly migrating to software-defined networks, it is conceivable that OpenFlow-enabled switches are widely used in the near future. With the most recent OpenFlow Switch specification [43], detailed per-flow and per-port statistics are collected by the switch and reported to the controller, with which the global edge load condition $\Lambda_e$ can be easily obtained. In this case, the shim layers on all the servers, where *Promenade* is deployed, simply need to collect the necessary information from the OpenFlow controller, so that rate control can be performed in a distributed fashion to maintain proportional fairness across the tenants.

## 6 Performance Evaluation

We evaluate our implementation of *Promenade* in the Mininet 2.0 emulation testbed [10], over a $k = 4$ fat-tree topology with 4 pods, 16 end hosts, and 20 switches, with a link capacity of 1 Mbps on all the links in the network. Mininet emulates such a fat-tree topology using a number of lightweight Linux containers running on the same Linux kernel. Our implementation consists of an optimized network coding engine that performs coding in user space, as well as our proposed rate control algorithm. We use RipL, a Python library to build our OpenFlow controller that supports ECMP, which hashes the 5-tuple of each flow to determine its path. We have also implemented our OpenFlow controller to record the path that each flow follows in the fat-tree topology. In addition, we use a handy network bandwidth monitoring tool called *Bandwidth Monitor NG* (`bwm-ng`) to accurately measure the edge load at each network interface that the flows pass through. Every second, edge load statistics obtained by our bandwidth monitor are provided as input to our implementation of the rate control algorithm in *Promenade*.

To take full advantage of the rate controlled logical tunnels, we use UDP as the transport protocol for all our flows in the network, and random network coding is used to transmit packets along multiple equal-cost paths in the topology between a pair of VMs, henceforth referred to as a *session* (since no VMs are used in the Mininet emulation environment). The main objective in our experiments is to evaluate and illustrate the effectiveness of both the network coding engine and the rate controller in *Promenade*, when it comes to reducing delays and achieving weighted proportional fairness in an emulated fat-tree datacenter network.

***The effectiveness of random network coding.*** Our first task is to evaluate the effectiveness of network coding, as it mitigates the problem of packet reordering. We run two sessions in Mininet, one using *Promenade* with random network coding and a coding window size of 8 packets (each with a size of 1 KB), and the other uses MPTCP, which is installed as a transport protocol in the Linux

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



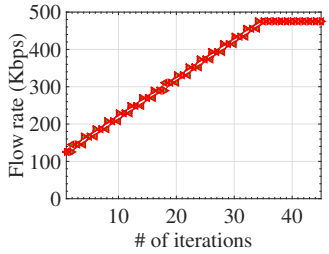



Fig. 5: The convergence of flow rates with a small group of 4 sessions, each with two paths.

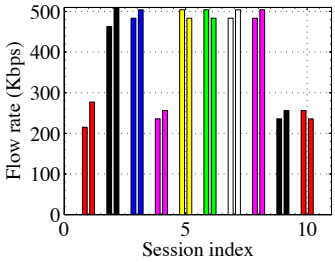



Fig. 6: The converged flow rates of the two sub-flows for each of the 10 sessions.

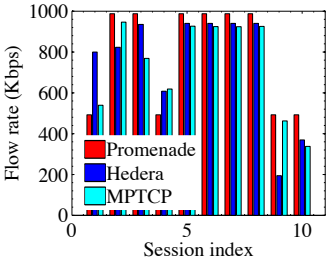



Fig. 7: The converged flow rates of the sessions (pairs of VMs on the end hosts) in *Promenade*, as compared to both Hedera [13] and MPTCP [6].

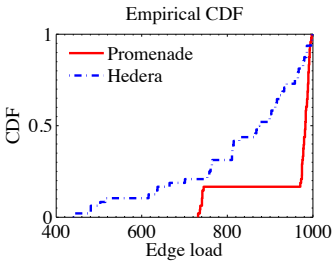



Fig. 8: The CDF of edge load statistics.

kernel. Both sessions go through all four paths in the $k = 4$ fat-tree topology. We have also generated random TCP and UDP cross traffic to saturate the network, creating a queue length differential across different paths. As shown in Table 2, with the use of network coding, both the average packet transmission delay and the standard deviation have been reduced as compared to MPTCP, demonstrating the superior ability for random network coding to mitigate the packet reordering problem.

| Protocol | Average delay | Std. dev. |
|---|---|---|
| **Random network coding** | 3.036 ms | 1.106 ms |
| **MPTCP** | 3.421 ms | 1.582 ms |

TABLE 2: Packet transmission delays with four equal-cost paths and cross traffic: a comparison between random network coding and MPTCP.

With our highly optimized implementation of random network coding with both multi-threading and SIMD acceleration, we wonder what the computational load on modern CPUs may be. To evaluate the performance of network coding, we ran our coding engine on a physical server with dual dual-core Intel Xeon 5160 CPUs running at 3.0 Ghz, with a total of 4 cores. We use a packet size of 1 KB, and perform a series of tests to evaluate the CPU load when the Gigabit Ethernet interface reaches its saturation point. Table 3 have demonstrated that with our suggested coding window sizes (4-8 packets), the CPU load for saturating the Gigabit Ethernet interface is acceptable.

| Coding window size | CPU load for encoding | CPU load for decoding |
|---|---|---|
| **4** | 12.61% | 13.90% |
| **8** | 25.76% | 27.26% |
| **12** | 37.81% | 40.26% |
| **16** | 50.70% | 52.96% |

TABLE 3: The CPU load while using random network coding with various sizes of the coding window.

***Rate control algorithm: convergence.*** To evaluate the effectiveness of the rate controller in *Promenade*, we first illustrate how flow rates converge over time. we start our experiment with a small group of 4 sessions, each splitting into two sub-flows by *Promenade*, transmitted along their respective equal-cost paths. Fig. 5 presents a visual illustration on how the rates of all four flows are able to converge after a period of 35 iterations with our distributed algorithm, and the convergence leads to a proportional fair allocation of rates to all four flows.

***Rate control algorithm: weighted proportional fairness.*** To show the effectiveness of *Promenade* on enforcing weighted proportional fairness, we run our experiments with a larger group of 10 sessions with equal weights, each splitting their flows into two sub-flows, transmitted along their respective equal-cost paths (determined by ECMP). In our experiments, we compare with both Hedera [13] and MPTCP [6], for the same source-destination pairs of end hosts in all 10 sessions. Hedera is proposed to adaptively schedule active flows to non-conflicting paths to maximize the aggregated network utilization without using multiple paths, and MPTCP is the most representative multi-path transport protocol, implemented in the Linux kernel. To make a fair comparison, though Hedera uses a single path for its TCP flow in each session, we ensure that it is hashed to one of the paths used by *Promenade* in the corresponding session. With respect to MPTCP, we use the same set of paths as *Promenade* for all 20 sub-flows in 10 sessions.

Fig. 6 shows the resulting flow rates for each sub-flow after their convergence in *Promenade*, while Fig. 7 shows the aggregate flow rates for each session in *Promenade*, as compared to their counterpart in both Hedera and MPTCP. For both figures, we can easily see that *Promenade* has achieved weighted proportional fairness perfectly, with equal shares of bandwidth allocated to flows when they compete for a congested link. For flows with no competition, they are able to saturate the link capacities along their paths. In comparison, both Hedera and MPTCP are able to achieve similar flow rates as *Promenade* when there is no competition among the flows; yet with competing flows, both fail to allocate the bottleneck bandwidth fairly according to the flow weights. Observed from the edge load CDF from all 48 edges in the network as shown in Fig. 8, we can see that *Promenade* is able to utilize link

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2019.2915638, IEEE Transactions on Parallel and Distributed Systems



bandwidth better than Hedera, and to achieve a higher total flow rate when all the flows are considered.

To magnify the illustration of our fairness comparison, Fig. 9 has singled out the only two pairs of sessions competing with each other for link capacities: Flow 1 vs. 9; as well as Flow 4 vs. 10. From this illustration, we can easily see the major difference between *Promenade* and Hedera/MPTCP with respect to fairness. The conclusion is loud and clear: *Promenade* is able to outperform both Hedera and MPTCP with respect to fairness between the sessions.

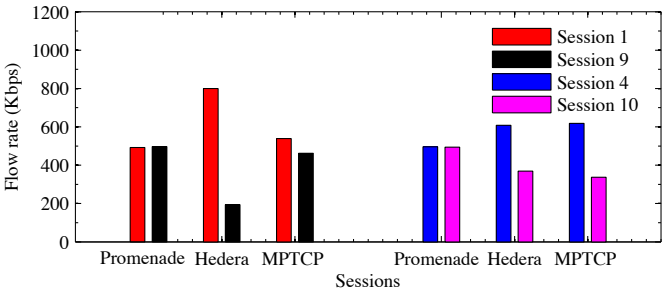



Fig. 9: In a three-way comparison with Hedera [13] and MPTCP [6], only *Promenade* has achieved weighted proportional fairness.

## 7 Concluding Remarks

To utilize the bi-section bandwidth and improve the flow rates between a pair of VMs in a datacenter network, it is best to transmit the packets along multiple equal-cost paths, on which packets may be reordered. In this paper, we propose *Promenade*, with the use of random network coding to mitigate the adverse effects from packet reordering, as well as a new rate control protocol that converges to weighted proportional fairness across different tenants in the network. Rather than proposing a heuristic without a theoretical foundation, the upshot of our new rate control protocol is that it is based on our formulation of an optimization problem, accounting for the tenant requirements in cloud datacenters, that can be solved by a distributed algorithm using the primal-dual method. *Promenade* is topology-agnostic and does not require hardware changes, and is to be deployed in the shim layers installed on all the physical servers, intercepting flows from the applications and transmitting them along multiple paths in a coded fashion. Despite the fact that the general techniques of random network coding and rate allocation are well-known, we are the first to design and implement a protocol that integrates them in such an optimal and practical fashion, to fully exploit their advantages in improving flow rates and meeting fairness requirements of tenants. Our experiments on the Mininet emulation testbed have validated the effectiveness of the *Promenade* protocol.

## References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. ACM SIGCOMM*, 2008.

[2] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992, IETF, 2000.

[3] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Data Center Traffic: Measurements & Analysis," in *Proc. 9th ACM SIGCOMM conference on Internet Measurement Conference (IMC)*, 2009, pp. 202–208.

[4] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the Impact of Packet Spraying in Data Center Networks," in *Proc. IEEE INFOCOM*, 2013.

[5] M. Laor and L. Gendel, "The Effect of Packet Reordering in A Backbone Link on Application Throughput," *IEEE Network*, vol. 16, no. 5, pp. 28–36, 2002.

[6] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proc. ACM SIGCOMM*, 2011, pp. 266–277.

[7] S. Barre, C. Paasch, and O. Bonaventure, "MultiPath TCP: From Theory to Practice," in *Proc. IFIP Networking*, 2011, pp. 266–277.

[8] P. Chou, Y. Wu, and K. Jain, "Practical Network Coding," in *Proc. Allerton Conference on Communications, Control and Computing*, 2003.

[9] T. Ho, R. Koetter, M. Médard, M. Effros, J. Shi, and D. Karger, "A Random Linear Network Coding Approach to Multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[10] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments Using Container-Based Emulation," in *Proc. 8th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2012.

[11] L. Fu, X. Wang, and P. Kumar, "Are We Connected? Optimal Determination of Source-Destination Connectivity in Random Networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 751–764, 2017.

[12] L. Fu, X. Fu, Z. Xu, Q. Peng, X. Wang, and S. Lu, "Determining Source-Destination Connectivity in Uncertain Networks: Modeling and Solutions," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3237–3252, 2017.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. 7th USENIX NSDI*, 2010.

[14] A. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection," in *Proc. IEEE INFOCOM*, 2011, pp. 1629–1637.

[15] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards A Next Generation Data Center Architecture: Scalability and Commoditization," in *Proc. ACM workshop on Programmable Routers for Extensible Services of Tomorrow (PERSTO)*, 2008, pp. 57–62.

[16] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *Proc. 9th USENIX NSDI*, 2012, pp. 16–29.

[17] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "MPTCP is Not Pareto-Optimal: Performance Issues and a Possible Solution," in *Proc. ACM CoNEXT*, 2012, pp. 1–12.

[18] S. Barré, O. Bonaventure, C. Raiciu, and M. Handley, "Experimenting with Multipath TCP," in *Proc. ACM SIGCOMM*, 2010, pp. 443–444.

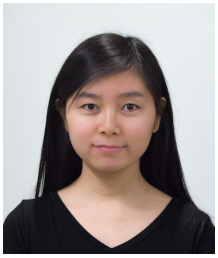
[19] B. Chihani and D. Collange, "A Multipath Tranport Protocol for Future Internet," in *Proc. International Conference on Networking and Future Internet (ICNFI)*, 2011.

[20] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-Packet Load-Balanced, Low-Latency Routing for Clos-Based Data Center Networks," in *Proc. ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013.

[21] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers," in *Proc. EuroSys*, 2014.

[22] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese *et al.*,

"CONGA: Distributed Congestion-Aware Load Balancing for Datacenters," in *Proc. ACM SIGCOMM*, 2014.
[23] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-Based Load Balancing for Fast Datacenter Networks," in *Proc. ACM SIGCOMM*, 2015.
[24] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro Load Balancing for Low-latency Data Center Networks," in *Proc. ACM SIGCOMM*, 2017.
[25] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing Data Center Networks across Services," *University of California, San Deigo, Tech. Rep. CS2010-0957*, 2010.
[26] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the Data Center Network," in *Proc. 8th USENIX NSDI*, 2011, pp. 10–23.
[27] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the Network in Cloud Computing," in *Proc. ACM SIGCOMM*, 2012, pp. 187–198.
[28] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A Cooperative Game Based Allocation for Sharing Data Center Networks," in *Proc. IEEE INFOCOM*, 2013.
[29] J. Guo, F. Liu, H. Tang, Y. Lian, H. Jin, and J. C. Lui, "Falloc: Fair Network Bandwidth Allocation in IaaS Datacenters via A Bargaining Game Approach," in *Proc. IEEE ICNP*, 2013.
[30] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: Practical Network Performance Isolation at the Edge," in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2013.
[31] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty Tenants and the Cloud Network Sharing Problem," in *Proc. USENIX Networked Systems Design and Implementation (NSDI)*, 2013.
[32] L. Popa, P. Yalagandula, S. Banerjee, and J. Mogul, "ElasticSwitch: Practical Work-Conserving Bandwidth Guarantees for Cloud Computing," in *Proc. ACM SIGCOMM*, 2013.
[33] L. Chen, Y. Feng, B. Li, and B. Li, "Towards Performance-Centric Fairness in Datacenter Networks," in *Proc. IEEE INFOCOM*, 2014.
[34] L. Chen, B. Li, and B. Li, "Barrier-Aware Max-Min Fair Bandwidth Sharing and Path Selection in Datacenter Networks," in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, 2016.
[35] ——, "Surviving Failures with Performance-Centric Bandwidth Allocation in Private Datacenters," in *Proc. IEEE International Conference on Cloud Engineering (IC2E)*, 2016.
[36] H. Shojania and B. Li, "Parallelized Progressive Network Coding With Hardware Acceleration," in *Proc. IEEE International Workshop on Quality of Service (IWQoS)*, 2007.
[37] J. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network Coding Meets TCP: Theory and Implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011.
[38] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding Data Center Traffic Characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 92–99, 2010.
[39] F. Kelly, "Charging and Rate Control for Elastic Traffic," *European transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.
[40] F. Kelly, A. Maulloo, and D. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportioanl Fairness and Stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, 1998.
[41] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[42] S.-W. Cho and A. Goel, "Pricing for Fairness: Distributed Resource Allocation for Multiple Objectives," *Algorithmica*, vol. 57, no. 4, pp. 873–892, 2010.
[43] OpenFlow Switch Specification 1.3.2. [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow



**Li Chen** is an assistant professor at the Department of Computer Science, School of Computing and Informatics at the University of Louisiana at Lafayette. She received her Ph.D. degree from the Department of Electrical and Computer Engineering at the University of Toronto in July 2018, where she also received her M.A.Sc. degree in January 2015. She received her B.Engr. degree from the Department of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2012. Her research interests include big data analytics, machine learning systems, cloud computing, datacenter networking, resource allocation and scheduling in networked systems.

**Yuan Feng** received her B.Engr. from the School of Telecommunications, Xidian University, Xi'an, China, in 2008, and both her M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Canada, in 2010, and 2013, respectively. Her research interests include optimization and design of large- scale distributed systems and cloud services.

**Baochun Li** received his Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He is a member of the ACM and a Fellow of the IEEE.

**Bo Li** is a Professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. He holds the Cheung Kong Chair Professor in Shanghai Jiao Tong University. Prior to that, he was with IBM Networking System Division, Research Triangle Park, North Carolina. He was an adjunct researcher at Microsoft Research Asia-MSRA and was a visiting scientist at Microsoft Advanced Technology Center (ATC). He has been a technical advisor for ChinaCache Corp. (NASDAQ CCIH) since 2007. He is an adjunct professor in Huazhong University of Science and Technology, Wuhan, China. His recent research interests include: large-scale content distribution in the Internet, Peer-to-Peer media streaming, the Internet topology, cloud computing, green computing and communications. He is a Fellow of IEEE for "contribution to content distributions via the Internet". He received the Young Investigator Award from the National Natural Science Foundation of China (NSFC) in 2004. He served as a Distinguished Lecturer for IEEE Communications Society (2006-2007). He was a co-recipient for three Best Paper Awards from IEEE, and the Best System Track Paper in ACM Multimedia (2009).