Improving Data Center Network Utilization Using Near-Optimal Traffic Engineering

Fung Po Tso, *Member*, *IEEE*, and Dimitrios P. Pezaros, *Member*, *IEEE*

Abstract—Equal cost multiple path (ECMP) forwarding is the most prevalent multipath routing used in data center (DC) networks today. However, it fails to exploit increased path diversity that can be provided by traffic engineering techniques through the assignment of nonuniform link weights to optimize network resource usage. To this extent, constructing a routing algorithm that provides path diversity over nonuniform link weights (i.e., unequal cost links), simplicity in path discovery and optimality in minimizing maximum link utilization (MLU) is nontrivial. In this paper, we have implemented and evaluated the Penalizing Exponential Flow-spliTing (PEFT) algorithm in a cloud DC environment based on two dominant topologies, canonical and fat tree. In addition, we have proposed a new cloud DC topology which, with only a marginal modification of the current canonical tree DC architecture, can further reduce MLU and increase overall network capacity utilization through PEFT routing.

Index Terms—Data center routing, data center topology, multipath routing, traffic engineering, load balancing, cloud computing

1 Introduction

Recent years have witnessed a significant growth of cloud computing services that substantially reduce capital and operating costs by sharing resources such as CPU time and storage among a large number of tenants. The underlying infrastructure is provided by data center (DC) networks embracing dedicated hierarchical tree topologies, as shown in Fig. 1a, with expensive switches in the higher layers, and lower end edge switches that connect to thousands of servers [10].

Recent research advocates "scale-out" topologies, as shown in Fig. 1b, that can horizontally expand DC architectures to provide for increasing aggregate bandwidth among all communicating hosts by interconnecting a large number of inexpensive commodity switches [5], [16]. However, research has also demonstrated that supporting protocols for these new architectures fail to leverage topological advantages [11], [26]. Most notably, recent measurement work [22], [8], [14] reveals that current DC networks are largely underutilized and therefore there is significant room for operators to optimize their network infrastructures before considering expanding their network or upgrading to new fabrics.

Most of current DC networks employ equal cost multipath (ECMP) forwarding [19] to leverage the path diversity provided by topological redundancy, by splitting traffic across multiple paths through hashing packets' headers. However, per-flow ECMP is link load and flow agnostic in which resulting hash collision can hinder load-balancing among links [6]. Recent research on DC network measurement has confirmed that congestion happens even when average link utilization is low [8], [21]. Therefore, although DC networks are often overprovisioned, a small but significant fraction of link congestion can largely deteriorate the overall network performance, demanding DC operators to expand or upgrade their networks. In light of this, we argue that the overall performance of network infrastructure, i.e., the period of time the network can operate efficiently without congestion and without adding extra networking components, can be significantly improved through the mitigation of congestion and better load balancing on otherwise bottlenecked links.

In this paper, we seek to answer the following question: *Are traffic engineering (TE) techniques based on a simple link-state routing protocol able to fully exploit path diversity in DC networks*? The Penalizing Exponential Flow-spliTing (PEFT) routing algorithm is just such a protocol [34]. It is a TE technique with hop-by-hop forwarding, i.e., routers running PEFT make forwarding and traffic splitting decisions locally and independently of each other. Moreover, packets can be forwarded through a set of unequal cost paths but the longer paths are penalized based on total link weights along the paths. PEFT consists of two detached components, namely link-state routing, including traffic splitting, and link weight optimization.

Despite PEFT having been proven to achieve optimal TE for wide-area ISP networks in [34], its applicability for DC networks remains largely unanswered because both traffic patterns and network topologies are enormously different in many ways due to the nature of cloud DC applications [14]. In this paper, we have implemented and evaluated a reactive online version of PEFT for a DC network environments. Our contributions are threefold:

- We have provided a practical implementation of PEFT (in C++) for DC networks.
- We have evaluated PEFT for canonical and fat-tree DC network topologies. The results indicate that PEFT

• *The authors are with the School of Computing Science, University of Glasgow, Lilybank Gardens, Glasgow, United Kingdom, G12 8RZ. E-mail: {posco.tso, dimitros.pezaros}@glasgow.ac.uk.*

Manuscript received 1 Mar. 2012; revised 5 Dec. 2012; accepted 9 Dec. 2012; published online 21 Dec. 2012.
Recommended for acceptance by V.B. Misic, R. Buyya, D. Milojicic, and Y. Cui.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2012-03-0226.
Digital Object Identifier no. 10.1109/TPDS.2012.343.

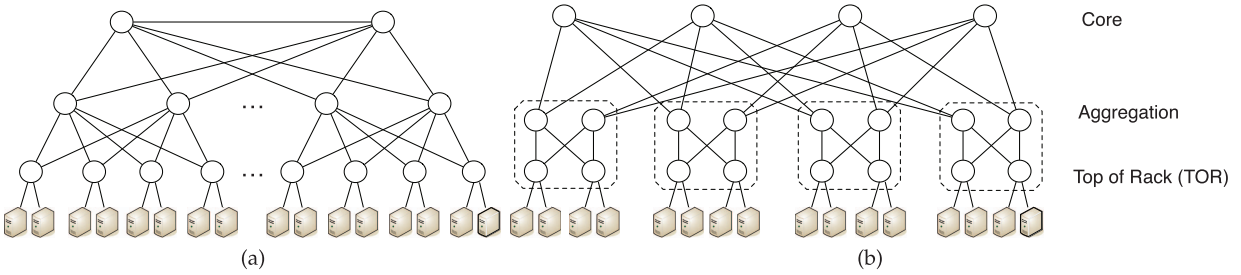

Fig. 1. (a) Canonical tree and (b) Fat-tree DC network topology.

falls only 3-5 percent short of optimal TE in DCs. At the same time, PEFT provides performance gain of at least 20 percent over ECMP in such topologies.

- We propose to interlink edge switches to increase redundant paths and take advantage of PEFT's ability to route packets over unequal-cost paths. Results show that PEFT in this topology provides for substantial increase DC network link utilization, which in turn improves DC capacity by about 10 percent.

The rest of this paper is organized as follows: We present our modified PEFT for DC networks in Section 2. We extensively evaluate the performance of PEFT in both canonical and fat-tree topologies in Section 3. A new interlink DC topology is proposed and evaluated it in Section 4. Runtime efficiency on hardware platform is given in Section 5. Section 6 discusses related work and Section 7 concludes this paper.

2 TE For DC Networks

TE is an indispensable tool used on the Internet to select routes that make efficient use of network resources. While many TE techniques have been proposed for the Internet, TE for DCs is still at a primitive state. Most DC operators simply move Internet TE onto a DC environment. Operators often adopt ECMP to spread traffic flows across multiple redundant paths using flow hashing. However, recent research has showed that ECMP fails to efficiently leverage path redundancy in DC networks. Studies have demonstrated that network redundancy cannot completely mask all failures, implicitly pointing to the inefficiency of ECMP [14]. Similarly, it is shown that ECMP's static mapping (hashing) of flows to paths does not account for either current network utilization or flow size, with resulting collisions overwhelming switch buffers and degrading overall switch utilization [6]. Moreover, Benson et al. [9] have tested ECMP with real DC traffic traces and found that the performance of ECMP is suboptimal. The consequence of such inefficiency is that, while most of the links in measured DC networks have relatively low utilization, a small but significant fraction of links appear to be persistently congested [8], [21]. As a result, operators will need to upgrade their networks even if they are generally underutilized.

Hedera [6], a centralized TE technique, schedules "elephant" flows exceeding 10 percent of the host-NIC bandwidth while the switches route "mice" flows using ECMP. However, there are two main limitations associated with Hedera. First, its traffic scheduler runs at every 5 seconds but [11], [25] shows that Hedera can only improve utilization by 1-5percent over ECMP and the scheduler needs to run at least up to every 500 ms for having a better improvement. At the same time, Hedera only schedules flows that exceed 10 percent of the NIC bandwidth. This is problematic too because whenever there are large host-limited flows (e.g., flows limited by host disk access) constantly transmitting at rates below 10 percent of the NIC bandwidth for a long time period, these flows will never be scheduled. In comparison, VL2 [16] uses Valiant Load Balancing to randomize packet forwarding on a per-flow basis, which is essentially an ECMP mechanism over a virtual layer-2 infrastructure. Other DC routing techniques [17], [22], [26] advocate using servers to determine routing path or rely packets. Such "server-centric" routing requires modification to end hosts.

Based on the above observations, we argue that the performance of current DC networks can be significantly improved if traffic flows can be adequately managed to avoid congestion on bottleneck links. This can be achieved by employing more elegant TE to offload traffic from congested links onto spare ones and alleviate the need for topological upgrades. Among a large number of available TE techniques, such as [12], [13], [20], [29], [30], [34], we have modified the PEFT routing algorithm to provide close to optimal TE for a variety of DC topologies [34]. PEFT is a *simple* and *link-state* protocol that can achieve optimal TE by using not only shortest paths, but also splitting traffic over longer paths with exponential penalisation.

2.1 Overview of PEFT

PEFT was introduced by Xu et al. [34] for ISP network operations, we hence summarize its key properties in this section.

Consider a network as a directed graph $G = (\mathbb{W}, \mathbb{E})$, where V is the set of nodes (where $N = |\mathbb{W}|$), E is the set of links (where $E = |\mathbb{E}|$), and link (u, v) has capacity $c_{u,v}$. The offered traffic is represented by a traffic matrix (TM) $D(s, t)$ for source-destination pairs indexed by (s, t). TE usually considers a link-cost function $\Phi(\{f_{u,v}, c_{u,v}\})$ that is an increasing function of $f_{u,v}$. When we consider the link utilization function $f_{u,v}/c_{u,v}$, then the PEFT's TE objective is to minimize $max_{(u,v)\in E}\Phi(f_{u,v}, c_{u,v})$. Optimal TE requires solving the following flow conservation and link capacity constraints given by [34], whose corresponding notation is given in Table 1

$$min \;\; \Phi(\{f_{u,v}, c_{u,v}\}) \tag{1a}$$

TABLE 1
Key Notations

Notation	Meaning
$D(s,t)$	Traffic demand from source s to destination t
$c_{u,v}$	Capacity of link (u,v)
$f_{u,v}$	Flow on link (u,v)
$f^t_{u,v}$	Flow on link (u,v) destined to node t

$$s.t. \sum_{v:(s,v)\in \mathbb{E}} f^t_{s,v} - \sum_{u:(u,s)\in \mathbb{E}} f^t_{u,s} = D(s,t) \forall s \neq t \quad (1b)$$

$$f_{u,v} \triangleq \sum_{t\in \mathbb{W}} f^t_{u,v} \leq c_{u,v} \forall (u,v) \quad (1c)$$

$$vars. \quad f^t_{u,v}, f_{u,v} \geq 0. \quad (1d)$$

To offload traffic from congested paths to less congested but slightly longer ones, PEFT allows exponential traffic splitting over unequal-cost paths, as shown in (2), where $p_{u,t}$ is the set of paths from u to t and $x^i_{u,t}$ is the fraction of forwarding a packet to the ith path, i.e., $p^i_{u,t}$

$$x^i_{u,t} = \frac{e^{-p^i_{u,t}}}{\sum_j e^{-p^j_{u,t}}}. \quad (2)$$

Similar to Open Shortest Path First (OSPF) protocol, PEFT-enabled switches make packet forwarding decisions independently based on link weight on a hop-by-hop basis. However, PEFT splits traffic along all possible paths, but penalizes longer paths exponentially. The protocol has two key components: It uses a *link-weight optimization* algorithm to compute optimal link weights for a given TM and then *link-state routing* computes and splits traffic flows according to the resulting link weights.

2.2 Modified PEFT

We have modified PEFT's link weight optimization as illustrated in Fig. 2. PEFT has originally been an offline TE designed for ISP networks where the TM is rather static and predictable. As shown in Fig. 2a, link weight optimization and link-state routing in PEFT are two separate and detached modules. In practice, the operator will run the link weight optimization module in a stand-alone machine while the link-state routing module will be embedded in PEFT-enabled switches. The operators will then need to measure the TM from their network over long timescales. Alongside forecasting techniques, the derived TM is used together with link capacities as inputs to the link-weight optimization module for computing the "best-fit" link weights for the best possible traffic distribution. Operators then install the resulting link weights to the PEFT-enabled switches that in turn invoke PEFT's link-state routing protocol at runtime to compute and accomplish the desirable traffic distribution in the network.

In contrast to aggregate traffic volumes in backbone ISP networks, previous measurement studies have revealed that traffic in a DC is highly bursty and is generally unpredictable as traffic patterns in DC networks change nearly constantly [16], [21]. The lack of short term TM

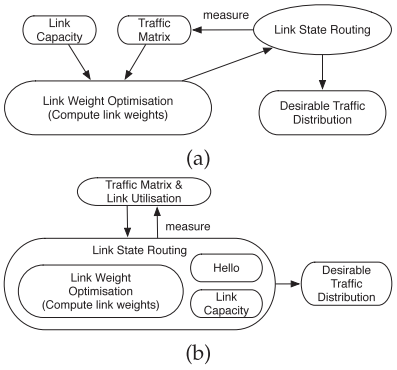

Fig. 2. (a) Original offline PEFT TE. (b) Modified online PEFT TE.

predictability is due to the use of random resource allocation to improve the (system-level) performance of DC applications since the distributed file system spreads data chunks randomly across servers for load distribution and redundancy.

In our implementation, we have integrated the link-weight optimization algorithm into the main routing control plane, meaning it is also included in PEFT-enabled switches running in parallel with PEFT's link-state protocol. We strive to make PEFT react more adaptively to the change of TM (online TE). We, therefore, incorporated a new network measurement module in PEFT, which queries the number of bytes transferred from all immediately associated servers and monitors link utilization of individual links. The former statistic will become the ToR-to-ToR (Top of Rack) TM, i.e., the number of bytes sent from servers under one ToR to servers under other ToR switches. The TM measurement is an proactive approach and, hence, the measured ToR TM will be exchanged over the network periodically. Upon completion, the TM will then be passed into PEFT's link-weight optimization to compute the optimal set of link weights. The link weights will be immediately used by the link-state protocol to derive the optimal traffic splitting ratio. Hence, instead of explicitly exchanging link weights, the modified PEFT propagates the TM over the network and subsequently each switch is able to independently compute link weights locally. It is reported that on a DC network, only a small fraction of traffic flows are destined outside the rack. So the ToR TM is in fact a sparse matrix which only contains 3-4 entries, each of 6 bytes in length. To this extent, the ToR TM can be efficiently exchanged through the network with only a little extra overhead. On the other hand, the link utilization is a reactive mechanism that detects abnormalities in link utilization. Upon detection, link weight computation and TM exchange are triggered immediately.

Locally measured TMs are exchanged over the network through a link-state advertisement (LSA). On this front we have also implemented a lightweight Hello protocol (based on OSPF's LSA) to facilitate LSA among switches running PEFT. LSAs are broadcasted at regular intervals, e.g., every 30 seconds, as well as whenever there is a change in link state. For example, at the network formation stage, LSAs carried by Hello messages are broadcasted to directly connected neighbours with $\mathrm{TTL} = 1$. Upon receiving the LSAs, a switch will update the TM and it will further

broadcast the LSA to notify the next-hop neighbours of the changes. If a LSA does not trigger any changes, it will be ignored and the local timer will be updated accordingly.

The amount of traffic to forward to the next hop over a given interface is determined by the traffic distribution ratio in PEFT. This is computed by the algorithm so that shorter paths will get allocated more traffic and longer paths will be penalized exponentially. In our implementation, PEFT's link-state routing computes and stores the traffic distribution ratio as a part of the routing table entry. So whenever a routing table is queried, the entry returned will contain an extra tuple for the traffic distribution ratio.

2.3 Implementation of Link-Weight Optimizer

Implementing PEFT requires solving a convex optimization problem, i.e., (1), for optimal traffic flow distribution. It is then followed by executing the link-weight optimization procedure to produce the link weights that would achieve this optimal traffic distribution. In this section, we present the implementation of these two processes.

2.3.1 Solving the Convex Optimization Problem

Solving (1) can be done through modeling it with a modeling language for mathematical programming (AMPL) [3] and by subsequently employing an appropriate solver, such as the CPLEX [1] and the Ipopt solver [2]. AMPL is highly readable with nearly one-to-one correspondence to the mathematical equation. Using AMPL to solve the equation is easier and more straightforward, but this is only suitable when PEFT is to be used in an offline manner because the resulting optimal distribution will need to be manually input into PEFT's link weight optimization module.

In our implementation, the modified PEFT needs to solve the optimization problem online in switches, in pace with the link-weight optimization module. As a result, we included the IPOPT's C++ library in our implementation to programmatically pass the optimal set of flow distribution to the link weight optimization module. Compared to AMPL, modeling the problem in C++ is more complicated.

On the problem setup, IPOPT asks for the Gradient of the objective function, the number of nonzero entries in the Jacobian of the constraints [32], and the number of nonzero entries in the Hessian [31]. Equation (1) has two classes of variables, $f_{u,v}^t$ and $f_{u,v}$. To reduce the number of variables to simplify the modeling and save memory space in runtime, (1) can be safely written as follows because $f_{u,v} \triangleq \sum_{t\in\mathbb{W}} f_{u,v}^t \forall(u,v)$

$$min\ max_{(u,v)\in E} \frac{\sum_{t\in\mathbb{W}} f_{u,v}^t}{c_{u,v}} \tag{3a}$$

$$s.t. \sum_{v:(s,v)\in\mathbb{E}} f_{s,v}^t - \sum_{u:(u,s)\in\mathbb{E}} f_{u,s}^t = D(s,t) \forall s \neq t \tag{3b}$$

$$\sum_{t\in\mathbb{W}} f_{u,v}^t \leq c_{u,v} \forall(u,v) \tag{3c}$$

$$vars. \quad f_{u,v}^t \geq 0. \tag{3d}$$

In (3), we only need to solve for the variable $f_{u,v}^t$. Assuming all edges are organized in a way that an individual edge can be indexed through an iterator i, the variable, $f_{u,v}^t$, is stored in an array, and can be indexed by jth *destination*, t and ith *edge index*, (u,v).

2.3.2 Link-Weight Computation

The procedural pseudocode for link-weight computation has been given in [34], one could just follow the procedures to implement it without much difficulty.

However, throughout our implementation and evaluation we have observed from our empirical settings that if we set initial values for link weights to the inverse of the link capacity, fewer link-weight computation iterations are needed. Since the majority of traffic is still being split over the shortest physical paths, setting the initial value to the inverse of link capacity can reduce the number of iterations and the resulting set of link weights would only vary slightly from the initial values.

To conclude, in spite of sharing the same theoretical foundation, the modified PEFT is fundamentally different from the original design in the operational context. Original PEFT computes routing plans from a mix of measured and forecast TMs offline, whereas our modified PEFT computes routing plans online as it them to adapt to changing network conditions through probing. As part of our contribution in this paper, we have made our implementation available to the research community at [27].

3 Performance Evaluation

While PEFT was proven to closely approximate TE optimality [34], its performance in a DC environment still unknown due to the highly unstable traffic demands that result in a continuously evolving TM. In this section, we evaluate the performance of PEFT with respect to improved load-balancing derived from path diversity, reduced maximum link utilization, and the overall network capacity gain.

3.1 Simulation Setup

We have evaluated PEFT for canonical tree (32 ToRs, 32 servers per rack), fat tree ($k = 8$) as well as for the interlinked tree (32 ToRs, 32 servers per rack) (discussed in Section 4) in Network Simulator 3 [4].

Based on this setup, each of the simulated DC environment contains 1,024 hosts. We believe this scale is large enough to reflect routing protocol properties used in the various DC fabrics. We assume 16-port commodity switches are used in fat-tree topology and all ports are used up for dense interconnect. All links connecting switches and servers in this architecture are 1 Gb/s [5]. As for the canonical and the interlink tree, we assume 48-port switches are in use. The host-to-switch links are 1 Gb/s and links between switches are 10 Gb/s [10]. Nonetheless, ToR interconnect links in the interlink topology are still 1 Gb/s because those ports were originally used to connect to servers.

3.1.1 Traffic Generator

While it is challenging to simulate the overall DC traffic characteristics, we have approximated the ToR TM and

generated traffic patterns according to recent measurement results, with respect to flow size distribution, number of concurrent flows as well as the distribution of in-rack and cross-rack recipients. Discussed as follows, we believe these distributions are sufficient to accurately capture the typical DC traffic properties.

We first review the traffic characteristics in the DC environment. Canonical DC traffic is carefully engineered so that servers communicate mostly within a cluster (e.g., the same rack), whereas in the cloud DCs, more cross-rack communication takes place due to the nature of distributed applications in the DC, such as MapReduce [8], [21]. Cloud DCs use randomness to improve application performance because the distributed file system spreads data chunks randomly across servers for load distribution and redundancy. In a cloud DC, more than 50 percent of the time, an average machine has about 10 concurrent flows, but at least 5 percent of the time it has more than 80 concurrent flows. It is also reported that more than 100 concurrent flow are rarely seen. Compared to the Internet, the distribution of traffic flows in a DC is simpler and more uniform. The reason is that in DCs, internal flows arise in an engineered environment driven by careful design decisions, for instance, the 100-MB data chunk size is driven by the need to amortize disk-seek times over read times [16]. Most of the flows are small (mice) and consist of HELLO messages and queries. Almost all the bytes in the DC are transported in flows whose lengths vary from about 100 MB to about 1 GB (elephants) [21].

Our traffic generator models the aforementioned traffic characteristics. It destines about 20 percent of traffic to servers outside the rack. It generates 10 concurrent flows 60 percent of the time, 80-100 concurrent flows 5 percent of the time, while the rest of the time zero to nine concurrent flows are generated. Ninety nine percent of the generated flows are smaller than 100 MB and the rest are between 100 MB and 1 GB. We have employed connectionless flows such that packets can be routed independently. Reliable and in-order packet transmission can be readily achieved through multipath congestion-aware protocols, such as multipath TCP (MPTCP) [25] and Packetscatter [24]. MPTCP is currently an IETF-draft [33] and Packetscatter is already in use in today's switch. We believe these protocols will become commodity in DC infrastructures because they enable more efficient resource usage by neither overloading nor underloading the topology, but rather using adequate congestion control over each path. Similar to our approach, Packetscatter randomly picks the output port on which to send packets when multiple shortest paths are available but with a more robust fast-retransmit algorithm. Whereas in MPTCP, packets on each path in PEFT can be treated as one of MPTCP's subflows and MPTCP will seamlessly maintain in-order packet transmission for applications.

3.1.2 Optimization Schedule

In our simulation, we set the simulator to run for 1,000 simulated seconds, and the modified PEFT is set to optimize the link weight every 50 s. In other words, the optimization process happens 20 times throughout the simulation. The initial set of link weights are from the last optimization in the previous run. We set a sparse optimization schedule because frequent TM updates exchange and optimizations are resource intensive. As it will become evident later in this section, the sparse optimization schedule only slightly sacrifices the optimality by about 3 percent, yet it provides for a significant performance gain. Moreover, our assumption is also supported by Benson et al. [8], which reveals that diurnal patterns (i.e., long term periodicity) and pronounced weekend/weekday variations exist in their measured DCs. In light of this, modified PEFT can be set to optimize the network on a hourly basis to approximate the change of TM.

Apparently, the optimality refers to having flows managed on link (u, v) such that the link capacity $c_{u,v}$ is fully utilized, preferably $f_{u,v} = c_{u,v}$. As a benchmark, we have solved (3) for optimal $f_{u,v}$ values and then compared them with measured ones.

TABLE 2
Average Path Stretch of PEFT

	Cross Aggregation	Cross Core
Canonical Tree	1	1.15
Fat Tree	1	1
Interlink Tree	1.25	1.28
Interlink Tree (NOP)	1.04	1.2

3.2 Load Balancing

In this section, we focus on the performance of PEFT with respect to network-wide load balancing over ECMP.

We list the average path stretch of PEFT for canonical tree, fat tree as well as for the interlink tree (discussed in Section 4) topologies in Table 2. We define the path stretch as the ratio of the length of actual path taken to the length of the shortest path.

The path stretch is relatively low because simulated traffic staying within the same rack follows the shortest path. Higher path stretch can be seen when traffic needs to flow across the core switches to servers residing in other subtrees. In spite of low stretch, PEFT can still effectively amortize link utilization by optimizing the distribution of flows in the DC network. As for the fat-tree case, PEFT has a path stretch factor of one, meaning that all packets are forwarded over the shortest paths because the topology provides enough redundant network capacity to carry the offered traffic without overloading any one of the shortest paths. Therefore, no detour is needed to achieve optimal traffic distribution. We will show in Section 4 that by physically increasing the unequal-cost redundant paths in the DC network, PEFT can greatly improve performance.

Performance of the DC network highly depends on the degree of path diversity. In the canonical tree structure, one can only find at most four equal-cost paths between any given pair of servers. In comparison, the number of redundant paths in the recently proposed fat-tree topology grows with the topology size, for example, 48-port commodity switches can yield a DC network with 576 $(43^3/4/48)$ equal-cost paths. Hence, with PEFT an immediate question arises: *Can PEFT exploit path diversity and better load-balance traffic in the network*? To answer this question,

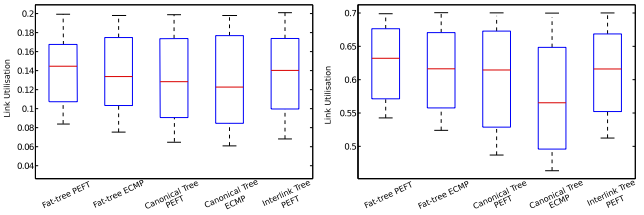

Fig. 3. Link utilization distribution for (a) lightly loaded network (MLU = 20 percent). (b) heavily loaded network (MLU = 70 percent).

we have looked into the average traffic distribution on links when a link of the aggregation switches reached 20 percent (lightly loaded) and 70 percent (heavily loaded, a commonly used threshold) utilization, respectively. It is worth noting here that a similar distribution can be seen when link utilization reaches 100 percent.

From Fig. 3 we can see that for both scenarios, ECMP constantly exhibits a wider spread in link utilization over PEFT, implying that traffic on the links across the network is highly unbalanced, where in heavily loaded networks the variance can be as high as 23 percent. On the contrary, when PEFT is applied the distribution variance for both cases is improved by approximately 5 to 10 percent. This is explicit evidence that PEFT, a path-based traffic splitting technique, schedules and splits traffic over longer paths to leverage path diversity, and, thus, better balancing of the distribution of traffic, preventing originally spare links from being idle.

3.3 Minimization of Maximum Link Utilization (MLU)

Fig. 4 shows the distribution (CDF) of MLU for PEFT and ECMP in both canonical and fat-tree topologies. PEFT and ECMP for both topologies were compared against optimal MLU. Fig. 4a compares the optimum versus actual ECMP performance we obtained from the simulation. From this figure, it is evident that for both topologies ECMP's distribution largely deviates from optimal. This implies that ECMP does not reasonably spread traffic load over redundant paths. Instead, it only evenly splits the traffic among all outgoing paths and does not avoid already congested links. In particular, in the fat-tree topology, which has an incredibly large number of equal-cost paths, we can see that some links still become highly utilized in ECMP, implying there is a large room for improvement.

Fig. 4b shows PEFT MLU performance against optimal for both topologies. It is clearly shown that the protocol deviates only by a small percentage, about 3 percent, from

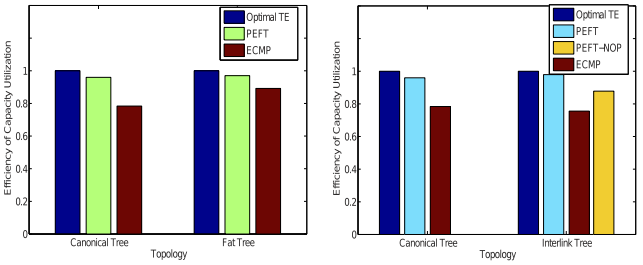

Fig. 5. Comparison of capacity utilization for (a) canonical tree and fat-tree (b) canonical tree and interlink tree.

optimal. The reason for such significant improvement is that PEFT optimizes distribution of flows such that they are unevenly split over all outgoing paths. For example, if a server wants to transmit to another server in a neighbouring rack, it physically has two equal-length shortest paths of three hops. But with PEFT the two paths may become unequal (reflected in the sum of link weights along the path) after optimization. Then, the traffic is exponentially split over the outgoing interfaces. However, we can still find the deviation between PEFT and optimal distribution because, similar to other TE techniques, PEFT needs to measure TM regularly and then update link weight and compute new traffic splitting ratios accordingly. Reactive and sparse TM updates prevent PEFT from reacting to changes in a timely manner.

Fig. 4c demonstrates a significant improvement of PEFT over ECMP on the order of 10-20 percent. With PEFT, the number of congested links is significantly reduced because it is very rare (only 2 percent of the time) that MLU will reach 100 percent in the canonical tree case. At the same time, there is a narrower difference for the fat-tree topology, because ECMP greatly benefits from the large number of redundant paths between any given pair of servers. Cross-rack traffic, even if it is routed through the core layer, can be amortized on each link.

3.4 Capacity Utilization

Next, Fig. 5 depicts the "efficiency of capacity utilization," which is defined as the percentage of traffic demand satisfied by a TE scheme when MLU reaches 100 percent, when compared to that of optimal TE. The improvement over efficiency of capacity utilization is commonly referred to as the "Internet capacity increase," but without changing the definition, we can safely call it "DC capacity increase" [131], [34].

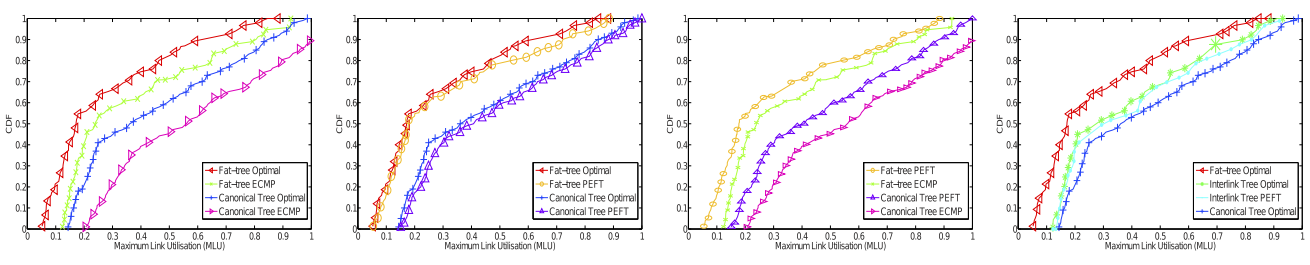

Fig. 4. Comparison of MLU for (a) optimal versus ECMP, (b) optimal versus PEFT, (c) PEFT versus ECMP in fat-tree and canonical tree topologies; and (d) optimal versus PEFT in interlink tree topology.

Our simulation results demonstrate that PEFT can significantly increase the capacity utilization efficiency over ECMP by 18 and 9 percent for canonical and fat-tree topologies, respectively. In both cases, PEFT falls only 3 and 2 percent below optimal utilization, respectively.

3.5 Protocol Stability

Protocol stability is a major issue for load-adaptive routing protocols such as online TE. Unstable protocol behavior typically leads to oscillations in the control plane, where a route flaps between two different links.

PEFT regularly performs optimizations, which will eventually result in a change of link weights. Although frequent link weights changes can cause route oscillation, PEFT can prevent this in two ways. First, PEFT's performance optimization can be set to a relatively long timescale, for example, every few minutes, allowing the routing protocol to converge such that the routes will not constantly alter over time. Second, through our study we found shortest paths (in terms of hop count) are always selected and most traffic flows are carried over the shortest paths. This observation implies that whenever path failures occur due to changes in link weights, it is safe to transmit packets over shortest paths during the new link-weight computation phase to avoid potential transient loops.

4 Interlinked Topology

In spite of topological differences, conventional and new DC fabrics share one common property—using multipath load balancing as the underpinning routing protocol. The higher degree of path diversity, the better multipath routing can perform. The difference in path diversity between conventional and new topologies has lead us to tackle the following challenge: *How to provide additional path diversity in current DC topologies without significant modification or investment*. One way to do this is to optimize the routing protocol. We have already seen that PEFT can significantly improve the performance over ECMP. Another way is to physically further increase the number of interconnects in the network. Fat tree adopts this latter approach, albeit it is not cost-effective for a conventional DC topology to upgrade to fat tree because it requires complete replacement and highly complex rewiring of the network. The alteration to the DC topology should, therefore, be marginal and should not incur substantial additional cost. After investigating PEFT in DC network further, we have found that due to the symmetric tree structure, the longer paths, if any, are at least two or multiples-of-2 hops further. Some of these paths in PEFT will eventually be exponentially penalized and will not be used.

We, thus, propose to interlink ToR switches (core switches are already interlinked in the reference design in [10]) as shown in Fig. 6. This way, we can physically increase the path diversity. More importantly, these asymmetric links create numerous paths only one-hop longer than the shortest one, and therefore the probability of seeing multiple unequal cost paths that will not be prohibitively penalized in PEFT has been greatly improved. Thus, traffic flows in the highly oversubscribed links in the upper hierarchy can be offloaded to these interlinks to avoid congestion. There are two further

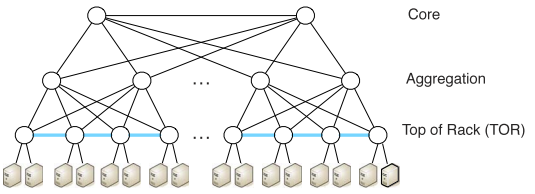

Fig. 6. Interlinked tree topology.

factors that make this proposal viable: The interlinking is technically simple because it does not require tailor-made configuration or wiring techniques (authors in [5] have acknowledged that fat-tree topologies can impose significant wiring complexity in large networks). From a fiscal viewpoint, the interlinking incurs no extra hardware cost as it only alters the interconnect between existing ToR switches. Interlinking switches will sacrifice two ports to servers per ToR switch (one for the two edge ToR switches), resulting in $m \times 2 - 2$ less number of servers to be connected to the topology, where m is the number of ToR switches. However, we argue that the performance gain from the network will outweigh loss of server computation in the long run because both the infrastructure's performance will be substantially improved. As a result, the target DCs can provide a better subscribers satisfaction, which will eventually lead to better business revenue. In addition, given that server computation resource utilization is mostly under 50 percent in today's DCs [18], the lost computation power can be effectively compensated through virtualisation on other machines.

The newly added links in this interlinked topology, however, impose some path asymmetry on the network. This is essentially unsuitable for ECMP because its underpinning OSPF protocol will either concentrate traffic on the "shortest" links connecting ToR switches, or it will exclude these links from the equal-cost shortest path sets. As we can see from Fig. 5b, ECMP over the interlinked topology has slightly worse capacity utilization over the canonical tree. On the contrary, this topology is particularly favorable to the PEFT due to its ability to include unequal-cost paths into routing decisions, combining the path diversity gain in both physical (the topology) and logical (the routing protocol) extents. It is also evidenced in Table 2 that PEFT in the interlinked tree topology exhibits a high path stretch irrespective of employment of link weight optimization.

We have extensively evaluated PEFT's performance over the interlinked topology through simulation and the results are shown in Fig. 4d. It is obvious that the optimal distribution of flows is 5-10 percent better than that of canonical tree, while PEFT's performance is very close to optimal, seeing only 0-3 percent difference in general. Even though without dynamic link optimization, PEFT is not likely to approximate optimality, we can see in Fig. 7 PEFT without optimization (denoted as PEFT-NOP) over the interlinked topology offers comparable performance to PEFT with optimization over the canonical tree, still achieving good capacity utilization as shown Fig. 5b. This finding emphasizes the efficacy of minor topological reconfigurations in maintaining low MLU even with static routing configurations.

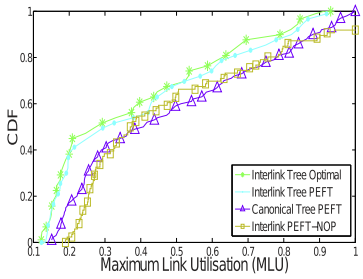

Fig. 7. (a) Comparison of PEFT in interlink tree. PEFT-NOP denotes using PEFT without optimization.

We argue that further implications of this finding are equally significant. First, it is very expensive for existing cloud DC operators to upgrade their network by vertical or horizontal expansion because the expansion either requires purchasing expensive aggression switches or prohibitive wiring complexity. The interlinked topology is a straightforward expansion for existing DC infrastructures. Second, applying PEFT to the interlinked topology can have the best composite effect on redundant path utilization. But even if operators wish to resolve to a minimal complexity static routing solution, they can simply apply PEFT without its optimization components.

Nevertheless, the interlink technique is not applicable to fat-tree topologies as in this topology all switch ports are used for dense interconnect, leaving no room for further interconnecting edge switches.

5 Runtime Requirements

In this section, we present the runtime efficiency of the component our modified PEFT in a real hardware environment.

5.1 PEFT Optimizer

We tested our PEFT optimizer in a machine running Ubuntu 10.04 Linux with an Intel P4-3-GHz processor and 1-GB RAM. There are two attributes of particular interest, the time required for computing the multicommodity flow with IPOPT and the time required for deriving the traffic distribution ratio (including link weight optimization) with PEFT.

We have tested our implementation by averaging the measured times over 100 executions. The evaluation results are shown in Table 3. Clearly, the total time (IPOPT link weight and PEFT flow splitting ratio computation) spent for the PEFT weight optimizer for the canonical and interlink tree is around 2 seconds and is about 4 seconds for the fat-tree topology. The reason for this gap is due to the large number of switches employed in the architecture. Nonetheless, the results indicate that the overhead for computing the TE will not be a bottleneck, if PEFT is scheduled to optimize the network at the interval of every few minutes or more.

5.2 Hardware-Assisted Monitoring System

During the past few years, several commercial switches now support the emerging OpenFlow standard. Such software-defined networking (SDN), i.e., OpenFlow, has created a new networking paradigm in which programmability in the network data path has become possible. With support from the network equipment vendors, the low-cost programmable hardware is becoming commodity for DC equipment. For example, Google is deploying Openflow-enabled switches in one of its DCs [15]. Borrowing the idea of having a smarter network switch, in our modified version of PEFT we have used a low-cost dedicated hardware component in a ToR switch to gather traffic statistics and link utilization. We have tested an open source *monitoring system* [7] on a NetFPGA [23] programmable router platform. This system enables real-time packet monitoring on hardware, and the captured statistics are then passed on to the software plane for finer grain processing. This monitoring system just resembles the role of a TM monitoring component in our proposal.

TABLE 3
Average Running Time for Modified PEFT Weight Optimizer

Topology	IPOPT (seconds)	PEFT (seconds)
Canonical Tree	1.232	0.758
Fat Tree	1.956	1.795
Interlink Tree	1.424	1.050

In this test, we have connected two host machines (as servers) via a NetFPGA router (acting as the ToR switch). The two host machines transfer data over 1 Gb/s links to the NetFPGA router that passively monitors traffic (byte counts) on its input interfaces and then passes it on to the appropriate output interface. We have compared the NetFPGA's forwarding efficiency to examine whether enabling a line-speed monitoring would degrade the hardware's forwarding performance. The test results are shown in Fig. 8. Fig. 8 plots the throughput ratio of the instrumented system (with monitoring functionality) over that of a plain switch. It is evident that the hardware-assisted traffic monitoring implementation incurs no significant deterioration on the router's forwarding efficiency, because the throughput ratios are mostly in the range between 0.99 and 1. This suggests that including line-speed monitoring in the routing protocol algorithm can be seamlessly accommodated using inexpensive hardware acceleration.

6 Related Work

TE techniques can be broadly classified as online and offline. The difference between online TE and offline TE is

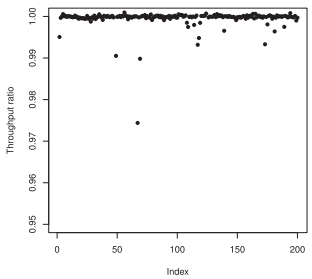

Fig. 8. Throughput ratio of instrumented (monitoring) system over plain switch.

the time scale when objective values, such as link weights, traffic splittings or scheduling of routing are adjusted. Numerous TE algorithms have been proposed, e.g., [12], [13], [20], [29], [30], [34], for wide-area Internet topologies. However, these TE techniques either do not achieve optimal routing or are too complicated to implement as they require tunneling. DC operators largely reuse or tweak such common TE mechanisms to manage DC topologies, yet these are ineffective due to the diverse DC traffic characteristics [9]. In contrast to other TE techniques, PEFT [34] only relies on the TM and achieves both optimality and simplicity since packet forwarding decisions are only made on a hop-by-hop basis.

TE for DC networks is still in a preliminary stage. Recent works include [6], [9], [19], [28]. Most of the current DC topologies rely on ECMP [19] forwarding, the most dominant TE for DCs, to split traffic among multiple paths of equal cost. An improved centralized TE technique, Hedera [6] flow scheduler schedules "elephant" flows exceeding 10 percent of the host-NIC bandwidth while the switches route "mice" flows using ECMP. In comparison, VL2 [16] uses Valiant Load Balancing to randomize packet forwarding on a per-flow basis over a virtual layer-2 infrastructure.

The most relevant work to ours is MicroTE [9], a system that adapts to traffic variations by leveraging the short term and partial predictability of the DC TM, to achieve fine grained TE for DC. However, MicroTE only slightly outperforms ECMP and it also exhibits 1 to 15 percent deviation from optimum. This is because the predictability of TM is largely due to the behavior of individual applications. Without knowing the application-layer information, prediction of TM is very difficult in a DC environment. This proposal also requires additional hardware investment and is complex to implement. In addition, [21] has concluded that traffic in large DCs is bursty in nature and unpredictable, which makes performance of TM prediction-based TE questionable. In comparison, our proposal does not reply on inaccurate prediction of the TM and still achieves near-optimal performance.

7 Conclusion

In this paper, we have advocated the use of online unequal cost TE as an efficient and viable mechanism to improve load-balancing and performance over DC topologies, offering substantial improvements over the commonly employed ECMP and other TE technique for DC such as MicroTE and Hedera, both can only provide 0-5 percent improvement over ECMP. We have implemented and thoroughly evaluated a variant of PEFT targeted at DC environments that exhibit rapid fluctuations in traffic demands. Based on PEFT algorithm, our protocol forwards packets over multiple unequal cost paths, whereas traffic splitting decisions are independently made based on the total link weight over all reachable paths, and exponentially penalize longer ones. We have demonstrated its applicability for DC network architectures through rigorous and extensive simulation. Evaluation results reveal that PEFT achieves near-optimal TE and outperforms ECMP in many ways, including fairer network-wide traffic load-balancing, minimizing MLU, and increasing network capacity. We have proposed a novel architecture that interlinks edge switches to further increase physical path diversity between any communicating server pairs. This straight-forward modification provides significant performance gains both in reducing MLU as well as in increasing network capacity, without requiring additional investment from DC network operators. Future work will concentrate on a prototype testbed implementation of the PEFT algorithm on a programmable-hardware router platform, and investigate the effects of increased packet reordering on application performance with MTCP and Packetscatter.

References

[1] IBM ILOG CPLEX Optimizer, www.ibm.com/software/integration/optimization/cplex-optimizer/, 2013.

[2] Interior Point OPTimizer (Ipopt), https://projects.coin-or.org/Ipopt, 2013.

[3] A Modeling Language for Math. Programming (AMPL), http://www.ampl.com/, 2013.

[4] ns-3, http://www.nsnam.org/, 2013.

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

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," *Proc. Seventh USENIX Symp. Networked Systems Design and Implementation (NSDI '10),* 2010.

[7] G. Antichi and A.W. Moore, "Monitoring System," http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/MonitoringSystem, 2013.

[8] T. Benson, A. Akella, and D. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," *Proc. Internet Measurement Conf. (IMC),* 2010.

[9] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," *Proc. ACM CoNEXT,* 2011.

[10] Cisco Systems, "Data Center: Load Balancing Data Center Services Solutions Reference Network Design," Mar. 2004.

[11] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and B.S, "Devoflow: Scaling Flow Management for High-Performance Networks," *Proc. ACM SIGCOMM,* 2011.

[12] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," *Proc. IEEE INFOCOM,* 2001.

[13] B. Fortz and M. Thorup, "Increasing Internet Capacity using Local Search," *Computational Optimization and Applications,* vol. 29, no. 1, pp. 13-48, 2004.

[14] P. Gill, N. Jain, and N. Nagappan, "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications," *Proc. ACM SIGCOMM,* 2011.

[15] O. Google. http://www.opennetsummit.org/talks/ONS2012/hoelzle-tue-openflow.pdf, 2013.

[16] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," *Proc. ACM SIGCOMM,* 2009.

[17] I.M.T.W. Group, http://datatracker.ietf.org/wg/mptcp/, 2013.

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "A Scalable and Fault-Tolerant Network Structure for Data Centers," *Proc. ACM SIGCOMM,* 2008.

[19] U. Hoelzle and L.A. Barroso, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan and Claypool Publishers, 2009.

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

[21] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," *Proc. ACM SIGCOMM,* 2005.

[22] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The Nature of Datacenter Traffic: Measurements & Analysis," *Proc. Internet Measurement Conf. (IMC),* 2009.

[23] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J.C. Mogul, "SPAIN: COTS Data-Center Ethernet for Multipathing Over Arbitrary Topologies," *Proc. Seventh USENIX Conf. Networked Systems Design and Implementation (NSDI '10),* 2010.

[24] NetFPGA, http://www.netfpga.org/, 2013.
[25] L. Popal, C. Raiciu, I. Stoica, and D. Rosenblum, "Reducing Congestion Effects in Wireless Networks by Multipath Routing," *Proc. IEEE Int'l Conf. Network Protocols (ICNP)*, 2006.
[26] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," *Proc. ACM SIGCOMM*, 2011.
[27] M. Schlansker, Y. Turner, J. Tourrilhes, and A. Karp, "Ensemble Routing for Datacenter Networks," *Proc. ACM/IEEE Sixth Symp. Architectures for Networking and Comm. Systems (ANCS)*, 2010.
[28] F.P. Tso and D. Pezaros, "Online PEFT Implementation in C++," http://www.dcs.gla.ac.uk/posco/weightsolver/html/index.html. 2013.
[29] T. Viet, Y. Deville, O. Bonaventure, and F.P, "Traffic Engineering for Multiple Spanning Tree Protocol in Large Data Centers," *Proc. 23rd Int'l Teletraffic Congress (ITC)*, 2011.
[30] H. Wang, H. Xie, L. Qiu, R. Yang, Y. Zhang, and A. Greenberg, "Cope: Traffic Engineering in Dynamic Networks," *Proc. ACM SIGCOMM*, 2006.
[31] Y. Wang, Z. Wang, and L. Zhang, "Internet Traffic Engineering without Full Mesh Overlaying," *Proc. IEEE INFOCOM*, 2001.
[32] Wikipedia, http://en.wikipedia.org/wiki/Hessian_matrix, 2013.
[33] Wikipedia, http://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant, 2013.
[34] D. Xu, M. Chiang, and J. Rexford, "Link-State Routing with Hop-by-Hop Forwarding Can Achieve Optimal Traffic Engineering," *IEEE/ACM Trans. Networking*, vol. 19, no. 6 pp. 1717-1730, Apr. 2011.

Fung Po Tso received the PhD degrees in computer science from City University of Hong Kong (CityU HK) in 2011. He is currently a SICSA research fellow at the School of Computing Science, University of Glasgow, United Kingdom. He has participated in a number of Hong Kong ITF funded projects on the areas of mobile computing and network management. His current research interests include cloud data center (DC) networks architecture, management as well as traffic engineering; distributed computing and mobile computing. He is a member of the IEEE and the ACM.

Dimitrios P. Pezaros received the BSc degree in 2000 and the PhD degree in 2005 in computer science from Lancaster University, and has been a doctoral fellow of Agilent Technologies (2000-2004). He is a lecturer (assistant professor) at the School of Computing Science, University of Glasgow. His research is currently focusing on dynamic resource allocation for Data Center networks, traffic classification, and measurement-based network control through software-defined networking. He has also developed an interest in building scaled-down cloud infrastructures from Raspberry Pi's. Previously, he has worked as a postdoctoral and senior research associate on a number of United Kingdom Engineering and Physical Sciences Research Council (EPSRC) and EU-funded projects, on the areas of performance measurement and evaluation, network management, cross-layer optimization, QoS analysis and modeling, and network resilience. He is a member of the IEEE and the ACM, and a fellow of the HEA.