

# Improving Multipath Routing of TCP Flows by Network Exploration

J. ALVAREZ-HORCAJO <sup>ID</sup>, (Member, IEEE), D. LOPEZ-PAJARES <sup>ID</sup>, (Member, IEEE),  
I. MARTINEZ-YELMO <sup>ID</sup>, J. A. CARRAL <sup>ID</sup>, AND J. M. ARCO <sup>ID</sup>

Departamento de Automática, Alcalá University, 28805 Alcalá de Henares, Spain

Corresponding author: I. Martinez-Yelmo (isaias.martinezy@uah.es)

This work was supported in part by grants from the Comunidad de Madrid under Project TIGRE5-CM (S2013/ICE-2919) and Project TAPIR-CM (S2018/TCS-4496), in part by the Ministerio de Economía y Competitividad through the Programa de Garantía Juvenil under Grant PEJ-2014-A-41867, in part by the Programa Propio de Investigación from the University of Alcalá under Grant CCGP2017-EXP/001, and in part by the Microsoft Azure Research Award under Grant CRM:0518647.

**ABSTRACT** Ethernet switched networks are widely used in enterprise and data center networks. However, they have some drawbacks, mainly that to prevent loops, they cannot take advantage of multipath topologies to balance traffic. Several multipath routing proposals use link-state protocols and equal cost multi-path routing (ECMP) to distribute the load over multiple paths. But, these proposals are complex and prone to flow collisions that may degrade performance. This paper studies TCP-path, a protocol that employs a different approach. It uses a distributed network exploration mechanism based on broadcasting the TCP-SYN packet to identify and select the fastest available path to the destination host, on the fly. Our evaluation shows that it improves on ECMP by up to 70% in terms of throughput for elephant flows and by up to 60% in terms of flow completion time for mouse flows. Indeed, network exploration offers a better, yet simple alternative to ECMP-based solutions for multipath topologies. In addition, we also study TCP-path for elephant flows (TFE) which restricts TCP-path application to elephant flows to reduce the exploration broadcast overhead and the size of forwarding tables thus improving its scalability. Although elephant flows represent a small fraction (about 5%) of total flows, they have a major impact on overall performance, as we show in our evaluation. TFE reduces both the overhead incurred during path setup and the size of the forwarding tables by a factor of almost 20. Moreover, it achieves results close to those obtained by TCP-path for elephant flows, especially when working with high loads, and yields significant improvements for all types of flow at medium- and high-load levels.

**INDEX TERMS** Data networks, ECMP, Ethernet, flow completion time, load balance, multipath, network exploration, throughput, TCP.

## I. INTRODUCTION

Ethernet switched networks are widely used in enterprise and data center environments due to their advantages, such as low cost, high speed, plug & play, etc. However, they present some drawbacks, mainly that they can have loops and do not take advantage of multipath topologies to balance traffic load among the different available paths/links. ECMP is the default multipath routing mechanism that has been employed in many routing protocols such as Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS). Furthermore, it is also used in L2 protocols such as TRILL [1] and SPB [2]. Nevertheless, ECMP presents some problems such as collisions between large flows [3], [4]. Its load balance does not take into account either the link

traffic load or the flow size. Moreover, it shows poor performance under failure conditions [5]. These drawbacks may reduce the delivered traffic by up to 40% despite the existence of different alternative paths.

To overcome these issues we propose TCP-Path, a simple protocol with some multipath features which belongs to the All-Path family of exploration protocols [6]. It has been designed considering that, most network traffic nowadays is TCP [7], [8]. The design can be explained according to the questions proposed in [9]. TCP-Path uses path discovery via network exploration instead of route computation (*Q1, Route Computation?*). The decision metric is based on latency (*Q2, Routing Metric?*), since only the fastest path is selected (*Q4, Number of Paths to Use?* and *Q5, How*

*Multiple Paths are Used?*). When a host opens a new TCP connection, an additional path is discovered by exploration. This achieves load balance (*Q3, Load Balancing?*) by continuously adapting to the current network traffic load at the moment of path selection. TCP-Path uses the same address learning and locking mechanisms to prevent loops as ARP-Path [6], which is a layer 2 protocol. It is based on broadcasting encapsulated TCP-SYN packets to explore the network and select the fastest path to the destination host. TCP-Path is a multilayer approach since the information learned from the TCP-SYN packets includes not only the source but also the destination Medium Access Control (MAC) address as well as the source and destination TCP ports. Hence, different application flows between a pair of hosts can use different paths (multipath in the broad sense), which are transparent to end hosts. TCP-Path is suitable for deployment in scenarios such as campus, small data center, cluster, and enterprise networks. A preliminary study on TCP-Path was presented at IEEE CloudNet 2017 [10].

Additionally, we also propose TFE (TCP-Path for Elephant Flows), to improve TCP-Path scalability. TFE restricts the application of TCP-Path logic to large flows (also known as elephant flows), while the rest of the flows are routed via layer 2 routing protocols (such as ECMP, Spanning Tree Protocol - STP, or ARP-Path). Our evaluation shows that TFE achieves very good performance, close to that of TCP-Path, both for elephant (large) and short-lived (mouse) flows, which are key to the application scenarios, while drastically reducing the state stored at switches and the broadcast control overhead.

This paper is structured as follows: related work is discussed in Section 2 and TCP-Path and TFE are described in Sections 3 and 4. In Section 5, we report the evaluation performed and the results obtained. Finally, we present our conclusions in Section 6.

## II. RELATED WORK

ECMP [11] is the most widely used algorithm to optimize multipath switched networks. It is a simple yet efficient routing strategy to balance traffic flows between a given pair of source and destination endpoints, among several equal cost paths. Traditionally, ECMP relies on a link-state routing protocol to compute several equal cost shortest paths (usually based on link bandwidth metrics) in advance. When required, each flow is assigned to one of the precomputed paths, following a random scheme, by hashing the flow packet header. As stated in the previous section, it works irrespective of link load and flow size, which leads to an undesirable sub-optimal performance (for example, when two or more large flows collide in the same path). Many solutions have been proposed to overcome this drawback, which mainly focus on improving the load balance to achieve better use of the available resources.

According to Alizadeh's classification for load balancing mechanisms [5], these solutions can be either centralized or distributed. Centralized flow multipath routing usually relies on the Software Defined Networking (SDN) paradigm,

where an SDN central controller schedules the flow path, as in Hedera [12] or VL2 [7]. Centralized solutions may be hard to scale out due to the single point of failure of the controller, and exhibit slow reaction times in data center networks. Although some studies have explored the possibility of deploying redundant and/or federated controllers such as Onix and ONOS [13]–[15], the lack of a standard definition of westbound and eastbound interfaces between SDN controllers renders consensus or advances in this area difficult.

Distributed solutions can be further divided into host-based or in-network. The former are hard to deploy and may increase incast (the convergence of many traffic flows on the same switch interface over a short period of time [8]). The packets arriving at the interface may exhaust either the switch memory or the maximum allocated buffer for that interface, resulting in heavy packet losses for some of the flows and leading to TCP timeouts). Moreover, host-based solutions require updates on legacy systems and sometimes even on the applications, as in MP-TCP [4] or FDALB [16].

In-network solutions can be local, either stateless or congestion-aware, or global/congestion-aware. Flare [17] and LocalFlow [18] are local congestion-aware solutions that simply acquire congestion information from the link output buffer. SPB [2] and TRILL Rbridges [1] are examples of the stateless group. They use a layer 2-based link-state routing protocol (IS-IS) to obtain the shortest paths between bridges and statically distribute the load by using ECMP. Thus, they suffer from the same flaws as ECMP. In general, local solutions perform poorly, especially when working with asymmetric network topologies and with symmetric topologies under failure conditions. Finally, many global/congestion-aware solutions rely on link state routing to compute congestion-aware paths, rendering them unstable, complex, and hard to deploy [19]. There are also other global/congestion-aware solutions which are based on overlay networks, as in Conga [5]. These overlay-based solutions have the additional need to exchange signaling traffic to monitor the network, which can limit their scalability.

Besides the above classification, in-network proposals can also be categorized according to traffic splitting; thus, they can be per flow, per packet or per flowlet (bursts of packets spaced at a minimum interval [5], [17]). Most of the revised proposals use a per flow strategy as in Conga [5], Hedera [3] and VL2 [7]. The basic algorithm consists of applying a hash function to the packet header to determine the selected path for a flow, as previously stated. Per packet approaches are simple to implement but usually require packet reordering as in [20]. Finally, per flowlet approaches split each flow into equal-sized segments and send them over multiple paths as in Presto [21]. At the receiver, packets are temporarily buffered to prevent reordering. Both [20] and [21] are not congestion aware, which reduces their performance during link failures.

Some theoretical studies have analyzed the load distribution problem in multipath networks. For example, [22] studies the problem of minimizing the cost of carrying traffic and

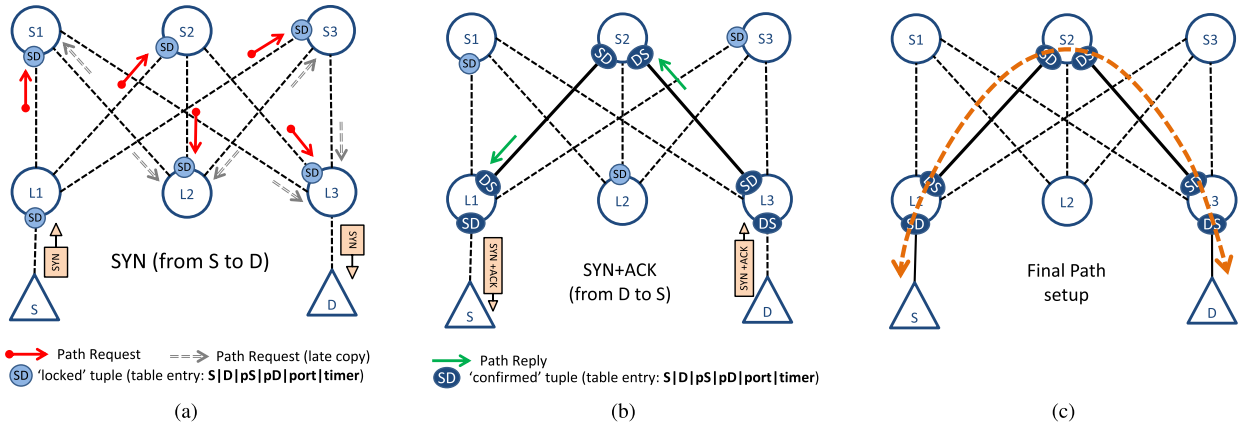


FIGURE 1. TCP-Path behavior example. (a) Network exploration. (b) Fastest path confirmation. (c) Flow setup.

proposes splitting the flows to balance the load, but this can provoke traffic disorder. Moreover, it is necessary to know the traffic matrix in advance, which is not always possible. Reference [23] analyses ECMP in Clos networks under a static flow model, which is far from a real-world scenario. Reference [24] formulates some load balance algorithms and studies their performance based on the *fluid limit* concept. However, it is necessary to know the exact flow size in advance, which is not always feasible.

Our proposed protocols, TCP-Path and TFE, explore the network at the moment of path setup, with a broadcast probe packet, to find the fastest available path. Thus, they can be classified as distributed, in-network, global/congestion-aware and per flow solutions. Moreover, they can be deployed in any network topology due to their simplicity, which is another advantage with respect to solutions designed for a specific target such as Conga, Presto and HULA [25]. Compared to ECMP, which is a solution based on (advance) path computation, our proposals distribute traffic flows among alternative paths more efficiently. They choose the fastest available path for a flow at the moment of flow setup instead of randomly picking one of the precomputed paths as ECMP does, thus mitigating undesirable collisions of large flows on the same path.

### III. TCP-PATH

TCP-Path is a path exploration protocol (from the All-Path family of protocols) designed to establish a TCP connection (or flow) over the fastest available path in a switched network, at the time of connection setup, between two end hosts. It finds the fastest path between a pair of edge switches using network exploration by broadcasting a protocol control frame over the entire topology. The fastest copy arriving at the destination edge switch sets the path to be used by that flow. Since we leverage TCP-SYN and SYN-ACK packets (encapsulated in a Path Request and a Path Reply packet) to explore the network and confirm the fastest path discovered respectively, no additional signaling traffic is needed.

TCP-Path relies on the locking mechanism designed for All-Path [6] to avoid broadcast loops without a spanning

tree or any other link prohibition protocol while exploring the entire topology. The main idea behind the All-Path protocol is to explore all available paths between a source and destination host with a broadcast frame, on demand. To prevent broadcasting loops while exploring the entire network topology, an All-Path switch associates the ingress-port of the first received copy of the broadcast frame with its source MAC address. Other copies of that frame may be received at other ports later, but they will be discarded as late frames because their source MAC address is already associated with another port. The first protocol of the family, named ARP-Path, leverages the IP to MAC address resolution process of the Address Resolution Protocol (ARP) (the *ARP Request* and *Reply* control frames) to simultaneously explore the network and set up the path. The exploration is performed by the *ARP Request* broadcast packet while the *ARP Reply* packet is used to confirm the fastest path discovered in the exploration phase.

Furthermore, TCP-Path is a switch only protocol; endpoints (hosts) are not aware of the path setup process and, thus, do not require any modification. The mechanism devised for path setup and frame forwarding is further explained below using a simple example as an illustration.

#### A. SWITCH INITIALIZATION

A TCP-Path capable switch periodically broadcasts *Hello* packets to its neighbors to announce itself. Hello packets have the main address of the switch as source address and a special *EtherType* value reserved for the protocol. When a Hello packet arrives at a TCP-Path capable switch, it creates (renews) an entry {neighbor MAC - Arrival Port} in its neighbor table. The remaining ports, not assigned in the neighbor table, are marked as potential host ports and will not participate in the path setup process.

#### B. PATH SETUP

Fig. 1 shows the network exploration mechanism in a Spine-Leaf topology with 3 switches (the spine) that provides multipath connections to a second row of 3 switches (the leaves).

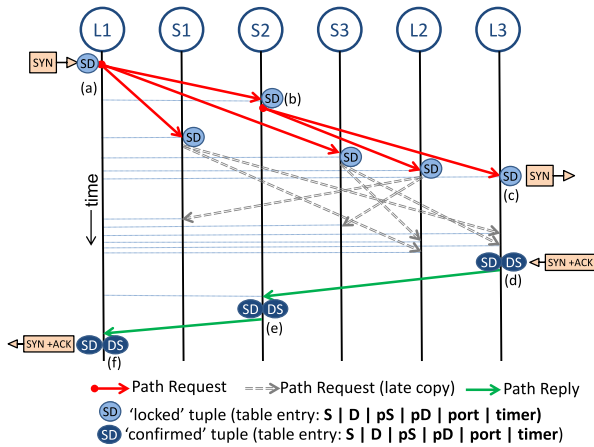


FIGURE 2. TCP-Path time diagram.

Fig. 2 represents the timeline of events. Each leaf-switch in Fig. 1, behaves as a classical ToR (Top of Rack) switch that is connected to several servers (hosts). Now consider server S, connected to leaf-switch L1, and server D, connected to leaf-switch L3. There are 3 possible shortest paths from L1 to L3, one across each spine switch connecting them (namely S1, S2, and S3). There are also, longer paths, crossing an extra pair of spine to leaf and leaf to spine hops. If server S tries to establish a new TCP connection to server D, it must first obtain server D’s MAC address using a MAC resolution mechanism (for instance classical ARP over a spanning tree or ARP-Path [6]). Once it obtains D’s MAC address, server S sends the TCP-SYN packet to negotiate the TCP connection setup.

In Fig. 1a, when the TCP-SYN frame is received at the S edge switch (L1), it is encapsulated in a special protocol control frame, a *Path Request*, using a specific EtherType value reserved for TCP-Path. The MAC address of the source server (S), the TCP protocol port identifying the connection at S (pS), the MAC address of the destination server (D), and the TCP protocol destination port identifying the connection at the D side (pD) form the SD tuple {S, pS, D, pD}, which is stored together with a timeout in a forwarding table, associated with the incoming port at L1, for (later) forwarding purposes. Then, L1 broadcasts the Path Request frame through all its network ports (those connected to spine switches in Fig. 2 (a)). When the Path Request frame arrives at an L1 neighbor switch (for instance, spine switch S2 in Fig. 1a), it also creates a new entry in its own forwarding table, which contains the SD tuple {S, pS, D, pD} associates this with its incoming port and forwards the Path Request frame through all its network ports except the incoming one (see also Fig. 2 (b)). This procedure is repeated by all switches in the network, which creates a temporary exploration tree with the root on the edge source switch. Finally, the fastest copy of the Path Request arrives first at the edge switch connected to server D (leaf switch L3). Switch L3 stores the SD tuple (as any other TCP-Path switch), de-encapsulates

the original TCP-SYN frame and sends it through the port connected to server D (see also Fig. 2 (c)).

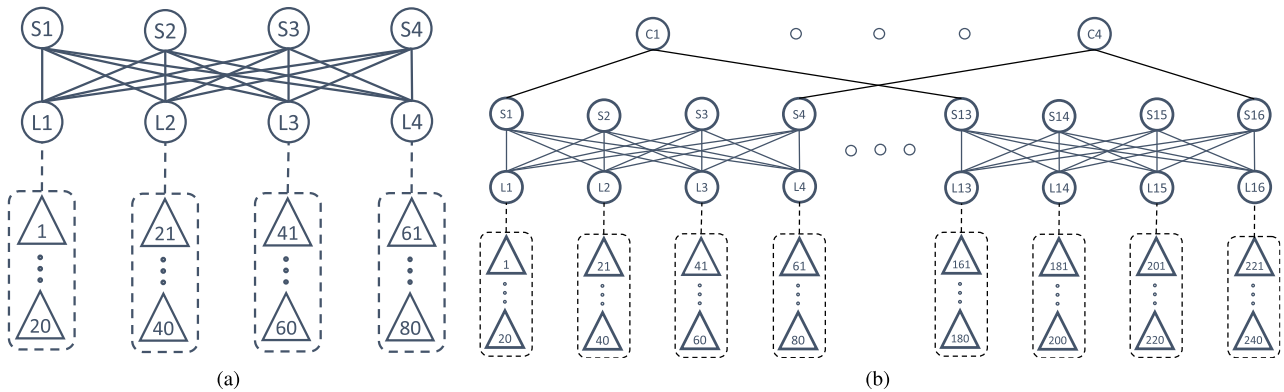
At this point, the chain of switches traversed by this frame (L1-S2-L3) is the fastest path available from L1 to L3. Newly created SD tuples remain in a locking state to avoid loops until they are confirmed or have expired, whichever happens first. Late copies of the original Path Request may be received at any switch due to the exploration (broadcast) process performed by the Path Request frame. However, they are discarded as late copies, provided that a non-expired SD tuple associated with a different port already exists at the switch, to prevent broadcast loops. Hence, only a single copy, the first one to arrive, is broadcast at every switch.

As shown in Fig. 1b, after receiving the TCP-SYN frame, server D replies with the corresponding TCP SYN+ACK to acknowledge the connection (see Fig. 1b)). When this reply arrives at L3, it confirms the SD tuple {S, D, pS, pD}, creates a new DS tuple {D, S, pD, pS} associated with the receiving port (the one connected to D), and confirms and updates the validity (timeouts) of both tuples. Now, both SD and DS tuples are in a confirmed state (see also Fig. 2 (d)). Then, it encapsulates the frame in a new protocol control frame, a *Path Reply* (identified by TCP-Path EtherType), and sends it through the port associated with the SD tuple. Each switch receiving the Path Reply also confirms this SD tuple, creates and confirms the corresponding DS tuple and forwards the Path Reply back (S2 in Fig. 2 (e)) until it reaches L1. Once the Path Reply has been processed at L1, the original TCP SYN+ACK is deencapsulated and sent to server S (see also Fig. 2 (f)). At this point, a bidirectional symmetric path between L1 and L3 (identified by tuples SD and DS at each switch) exists, which is able to forward the traffic between S and D. This is the fastest available path in the network at the setup time. Unconfirmed SD tuples on other ports at switches will simply expire (see Fig. 1c). There is no additional delay incurred in the path setup other than the time needed to process the Path Request and Path Reply packets at edge switches (encapsulate and de-encapsulate TCP-SYN and SYN-ACK). The path exploration process takes the same time as traditional TCP-SYN forwarding or less, since it goes via the fastest path between the edge switches. Additionally, Fig. 3 shows the processing of a frame, which carries TCP data, on reception at a TCP-Path capable switch.

### C. DATA FORWARDING

When a data frame is received at a TCP-Path switch, it first checks whether a matching TCP-Path tuple exists in the forwarding table. If there is a match, the switch retrieves the forwarding port from the matching entry, and forwards the frame accordingly. Finally, it renews the expiration timer for the matched tuple. If no matching entry is found, the switch starts a path recovery process or simply falls back to legacy (non-TCP-Path) forwarding based on ARP-Path, STP, etc.





**FIGURE 5.** Evaluation topologies. (a) Spine-Leaf 4-4-20 evaluation topology. (b) Three-layer topology derived from Facebook-Altoona.

As we will show later when we describe the experimental setup, several studies have used real traffic traces to characterize the traffic in data centers. These studies show that only about 5% of flows carry more than 10 MB of data (we call these Elephant Flows), and that these are responsible for 50-95% of the total bytes transmitted in the network. As such, we think they represent a promising candidate for our protocol, and decided to call it TFE (TCP-Path for elephant flows). Switches running TFE must be able not only to detect frames carrying TCP data as for TCP-Path, but also to classify them as a function of the type of TCP flow they belong to, for instance by inspecting the TCP *port*.

## V. EVALUATION

We aim to compare the performance of TCP-Path and TFE versus ECMP in terms of throughput (average Mbps per flow) and FCTs under different network load and flow traffic distribution conditions.

### A. TESTBED

Our hardware infrastructure consists of 7 computers powered by Intel(R) Core(TM) i7 processors with 24 GB of RAM, all of which are interconnected via a GbE Netgear GS116 switch, for emulation and simulation. We also have a F16 instance in the Microsoft Azure platform, powered by Intel(R) Xeon(R) E5-2673 v3 with 32GB RAM, intended solely for very high demand simulation purposes since our infrastructure does not have sufficient power to emulate large scenarios. We use the well-known *ns-3 network simulator* [29] as our primary platform to evaluate the proposed protocols. *ns-3* features a built-in TCP/IP stack, so it is only necessary to develop TCP-Path and TFE switch models.

To validate both the simulation model itself and its results, we have also developed TCP-Path and TFE software switch implementations based on *OfSoftSwitch* [30]. Implementations of TCP-Path and TFE is tested using the *Mininet* [31] emulation platform, which makes it possible to evaluate our protocols using a real Linux TCP/IP stack. Thus, the same sets of experiments are performed on both *ns 3* and *Mininet*

platforms, running at different link rates depending on hardware limitations.

### B. EXPERIMENTAL SETUP

We use a 4-4-20 *Spine-Leaf* network topology (two rows of 4 switches with 20 servers per leaf switch for a total of 80 servers) [5], [21], [32] as shown in Fig. 5a. In addition, we run some simulations on a three-layer topology derived from Facebook-Altoona [33] as shown in Fig. 5b to validate the results obtained with the *Spine-Leaf*.

Traffic flows are randomly distributed between any pair of servers attached to two different leaf switches with no further restrictions. In addition, we consider two different flow size distributions, *Data Mining* and *Web Search*, derived from experimental traces taken from actual data centers [7], [8]. Fig. 6a shows the Cumulative Distribution Function (CDF) of both distributions and also illustrates how flows are classified according to their size.

Flows with less than 10 KB of data are considered *mouse* flows, while those carrying more than 10 MB are considered *elephant* flows, as explained in [7]. The remaining flows are identified as *rabbit* flows. Fig. 6b shows the percentage for each type of flow as well as the percentage of bytes transmitted by each type of flow. Lastly, we calculate the average *flow Inter-Arrival Time* (IAT) to achieve an average offered network load of 10%, 20%, and 40% with respect to the full capacity of links, according to either the *Web search* or *Data mining* flow size distributions. Our TFE implementation relies on the sender application to mark elephant flows by using a specific TCP source port range.

Each experiment runs for 1800 seconds and it is repeated 10 times to later compute 95% confidence intervals. Table 1 summarizes the full setup of the experiments conducted.

We also use ARP-Path as a layer 2 (default) forwarding protocol since it is easy to implement and allows us to use the full network infrastructure. Unlike other protocols such as STP, it prevents loops while allowing broadcasting over the full network, and no link prohibition is needed [6].

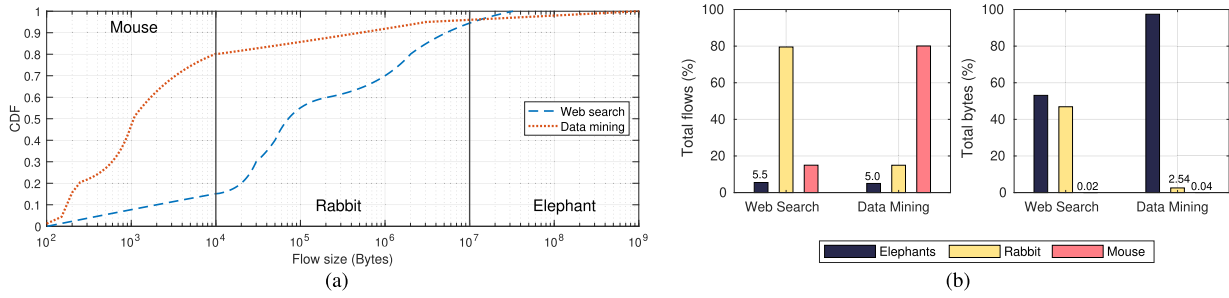


FIGURE 6. Flows characterization. (a) Flow size distributions. (b) Total flows and total bytes per flow type.

TABLE 1. Experimental setup.

Network topology	Spine-Leaf (4 - 4) [5] & Three-layer (derived from Facebook-Altoona)
Servers per leaf switch	20
Flow distribution	Random inter-leaf
Flow size distributions	Web search [8] & Data mining [7]
Network offered load (%)	10, 20, 40 & 60%
Link speed (Mbps)	10Mbps, 100Mbps & 1Gbps
Run length (s)	1800 s
Warm up time (s)	800 s
Number of runs	10

### C. RESULTS

First, we carry out a set of ns-3 experiments with 1 Gbps link rates to compare ECMP, TFE, and TCP-Path in terms of throughput and FCTs. Then, a second set of experiments are carried out both in ns-3 (simulation) and Mininet (emulation), but with link rates limited to 10 Mbps (due to limitations in the maximum switching capacity of our prototype switches) to validate the results obtained by simulation.

Regarding the first set of experiments, Fig. 7 compares the throughput of ECMP, TFE and TCP-Path, for the different types of flow (*elephant*, *rabbit*, or *mouse*). The results obtained using the Web Search traffic distribution are shown on the left-hand side of the figure and those obtained using the Data Mining distribution on the right-hand side. As expected, throughput decreases with the offered load regardless of the protocol or traffic distribution in use. When the number of flows contending for the same resources (link capacity) increases, each flow obtains a smaller share of the available resources. Furthermore, they spend longer in the network, aggravating this effect.

TCP-Path and TFE outperform ECMP at all levels of load. Clearly, TCP-Path and TFE take advantage of their native load balance feature derived from their network exploration capability and fastest path selection mechanism. The improvement achieved increases with network load, and reaches the level of 50% with respect to ECMP at 40% load and for elephant flows, because the collision of more elephant flows in the same link is handled more efficiently by TCP-Path (and TFE). We can also see that TFE approaches TCP-Path performance in the case of Data Mining distribution but clearly falls behind it with Web Search distribution

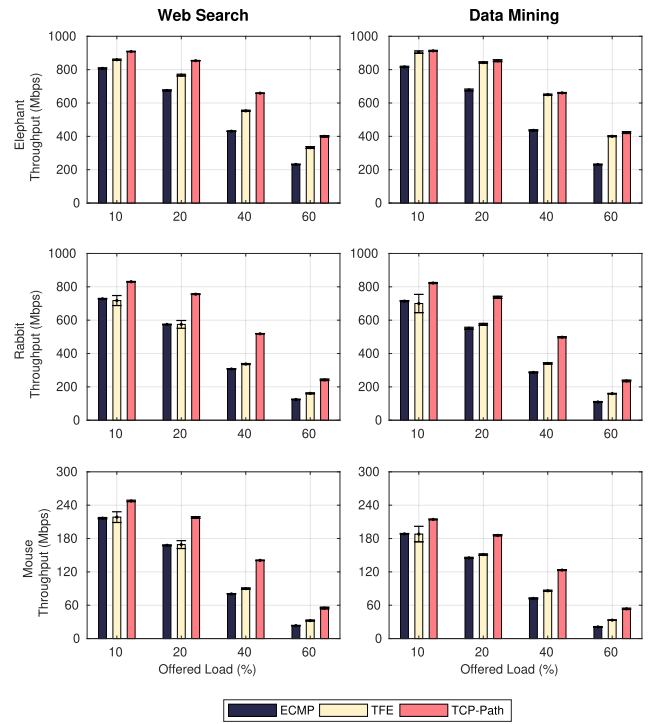


FIGURE 7. Throughput on Spine-Leaf (4-4-20) 1 Gbps topology.

because in the former case, the number and importance of elephant flows is greater, whereas in the latter case, rabbit flows (not treated by TFE) constitute a substantial share of the traffic. For rabbit and mouse flows, the improvement is negligible at low load levels, but again increases with load to reach values of 10-20% at 40% load, which is noteworthy given that these flows are not directly treated by the protocol. Fig. 8 shows the exact improvement ratios achieved versus ECMP in all cases. It also includes the results obtained at 60% load. Although these show even better improvement ratios we consider that this load exceeds real network scenarios.

Conversely, as shown in Fig. 9, FCTs exhibit the opposite behavior since throughput and FCT are inversely proportional. We can see that TCP-Path and TFE obtain a larger decrease in FCTs as the load increases compared with ECMP and ARP-Path. Moreover, mouse flows show the greatest FCT reduction. This is particularly important since FCT is

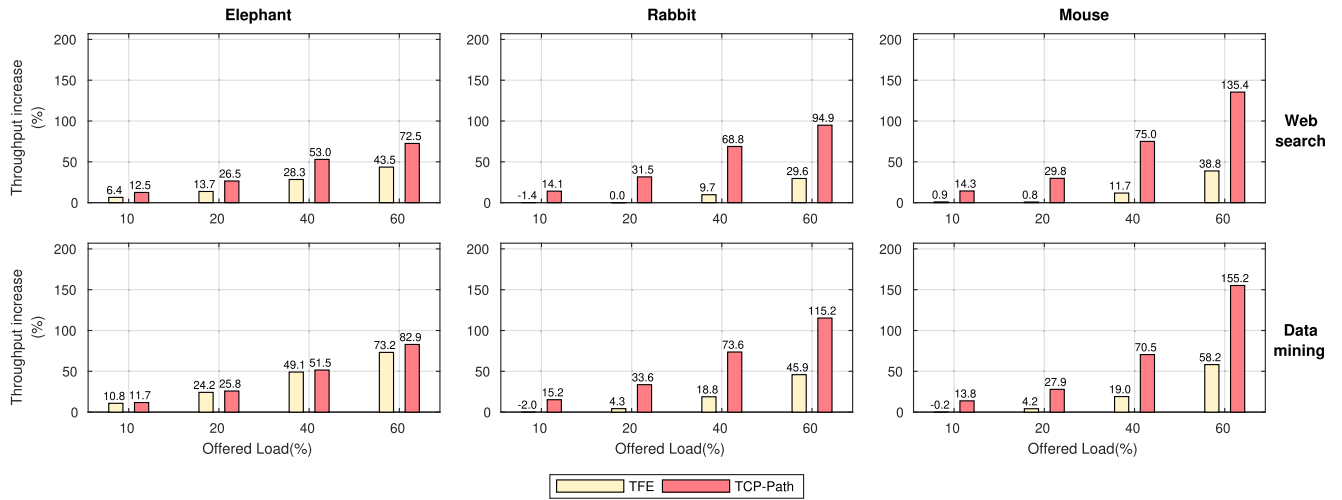


FIGURE 8. Throughput improvement ratios versus ECMP.

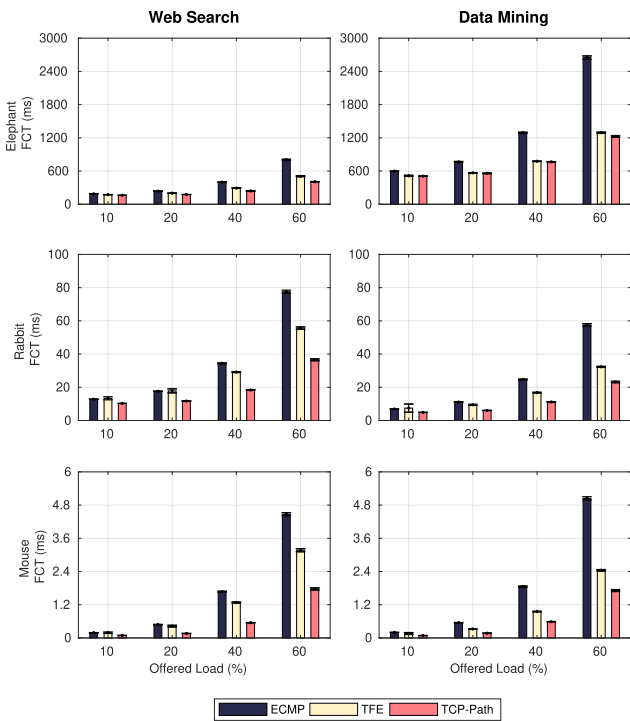


FIGURE 9. FCT on Spine-Leaf (4-4-20) 1 Gbps topology.

the key performance indicator for this type of flow, according to [8]. Mouse flows are usually associated with time constrained applications: thus, reducing their FCTs may also have a considerable economic impact [34]. Again, we can see that TFE achieves similar results as TCP-Path for elephant flows especially when working with high loads and the Data Mining distribution, and yields significant improvements for all types of flow at medium and high load levels. Fig. 10 shows the improvement ratio achieved versus ECMP in all cases.

Finally, Fig. 11 shows the evolution of the average throughput for elephant flows over time (only the flows effectively finished during the simulated time and already started at a given simulated time are considered). The information is processed in 50-second batches. For example, a point at  $t = 350$  seconds represents the average throughput of the elephant flows started after  $t = 350$  seconds and finished before the simulation end time. We choose to show elephant flows because they are the scarcest type and are therefore also prone to exhibit a larger deviation and take more time to stabilize. We can see that the average is quite stable right from the beginning of the simulation. However, if we reduce the link rate down to 10 Mbps (as we did in Mininet), we can see in Fig. 12 that the average takes almost 800 seconds to stabilize (especially with the Data Mining distribution). Consequently, our measure interval was set at between 800 and 1400 seconds (we also exclude the last 300 seconds because some flows have not yet finished).

### 1) VALIDATION

In order to validate the results obtained by simulation and shown in the above section, we carry out a second set of experiments on two platforms: ns-3 (simulation) and Mininet (emulation). To this end, we developed software switch implementations of TCP-Path and TFE based on OfSoftSwitch [35]. The tests are run with link rates restricted to 10 Mbps due to limitations in the maximum switching capacity of our prototype switches.

Fig.13 shows the throughput for elephant, rabbit, and mouse flows for both ns-3 and Mininet. A comparison of the results for both platforms shows that they are fairly similar. However, a closer inspection reveals some differences. For instance, the ns-3 simulator offers a lower throughput for mouse flows than Mininet. Consequently, the FCT of mouse flows is higher in the ns-3 simulator than in Mininet. There are two reasons for this. The first is related to



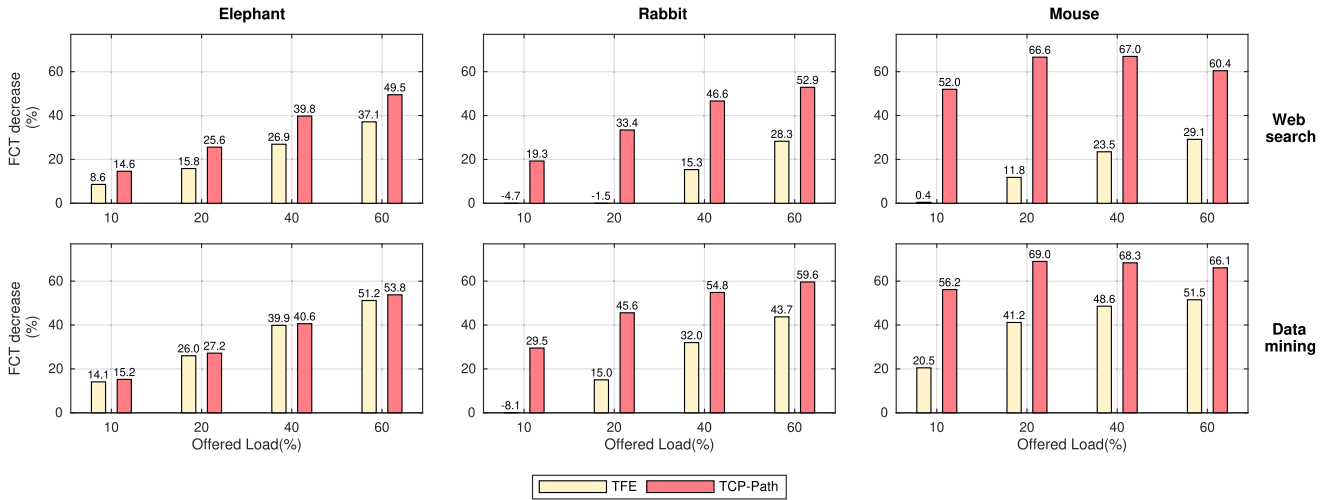


FIGURE 10. FCT improvement ratios versus ECMP.

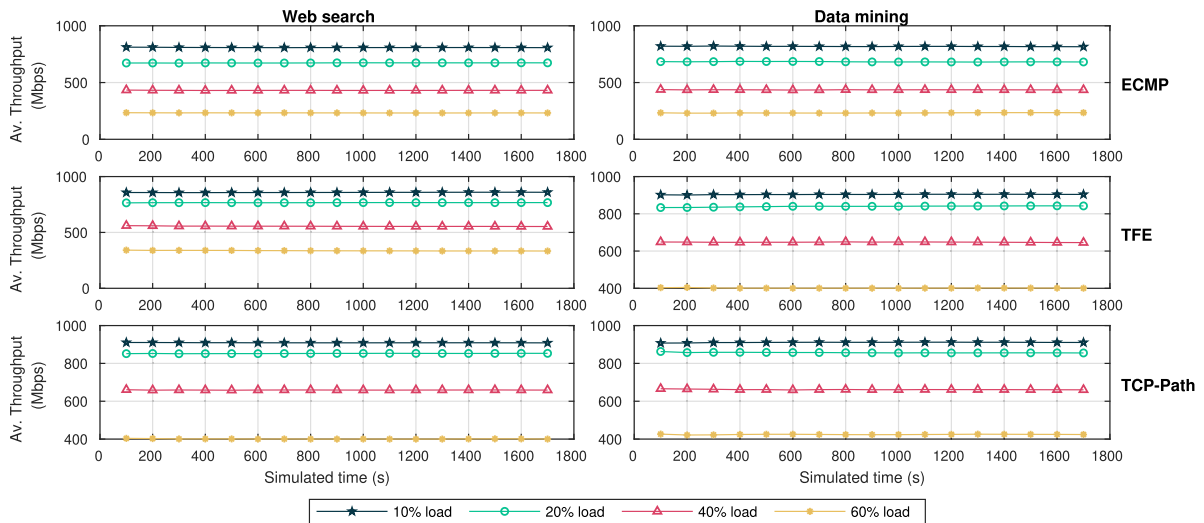


FIGURE 11. Throughput evolution along the time (elephant flows) at 1Gbps.

memory management: the Mininet platform allows bursts of packets to be sent to the TCP sockets for their transmission in the network; however, the ns-3 implementation has memory limitations that oblige us to send the data to the TCP socket on a packet basis, which prevents the allocation of a burst of data packets. Moreover, the ns-3 TCP model features a TCP slow-start phase, while the Mininet TCP stack does not implement this initialization phase. Together, all these differences result in a slower transfer speed at the beginning of a flow that mostly affects smaller flows (rabbit and mouse). The effect on larger flows is negligible because these last longer in the network and most of the data transfer is accomplished in the congestion avoidance phase, thus achieving maximum throughput. Therefore, if we compare these results with the results shown in the previous section,

we can conclude that our ns-3 simulator yields very similar results once we scale out the link rates. To confirm this, we also ran the experiments in ns-3 with 100Mbps link rates. The results as shown in Fig. 14 reflect a perfect scale when compared with those obtained with 10Mbps and 1Gbps link rates.

We also carry out simulations on a three-layer topology derived from Facebook-Altoona [33], as shown in Fig. 5b. This offers four disjoint paths (one traversing each top-layer switch) between any pair of ToR switches, identical to the Spine-Leaf we use extensively in the evaluation, but arranged in three layers. The results, shown in Fig. 15a and Fig. 15b, confirm that adding a third layer of switches does not affect the results provided that the number of multiple disjoint paths between hosts is kept constant.

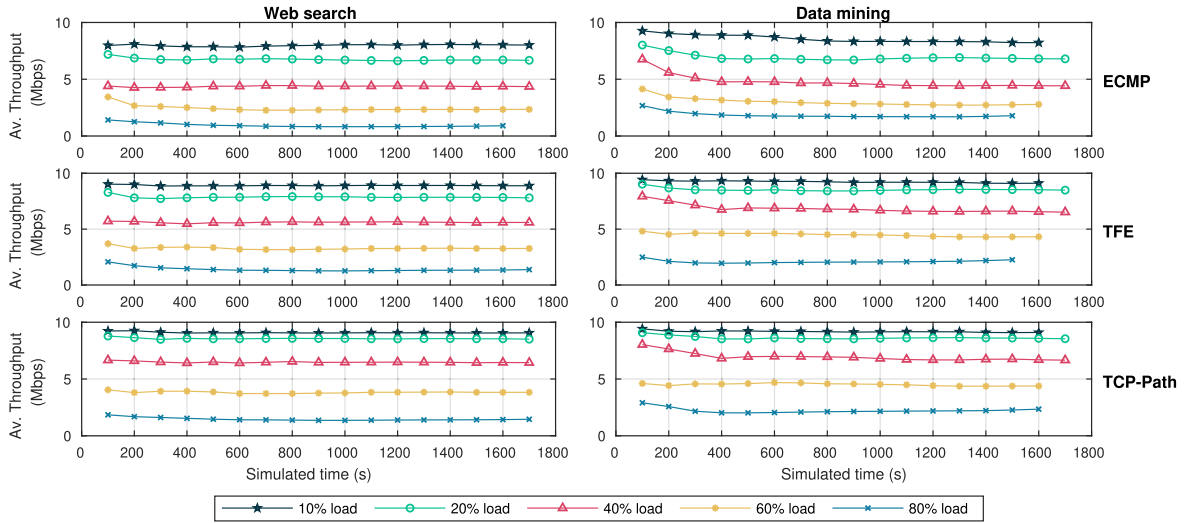


FIGURE 12. Throughput evolution over time (elephant flows) at 10Mbps.

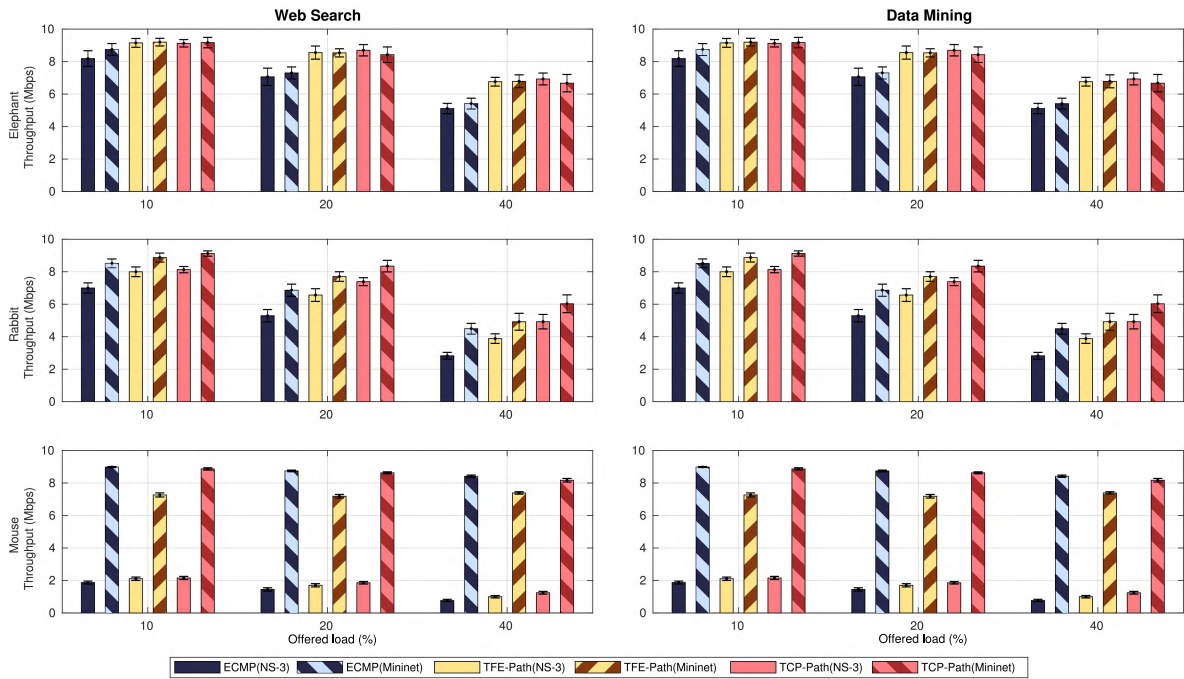


FIGURE 13. Throughput in ns-3 and Mininet on Spine-Leaf (4-4-20) 10Mbps topology.

D. SCALABILITY

In section V-C we showed that TCP-Path outperforms TFE in terms of throughput and FCT. However these performance parameters may not offer a complete vision in some cases, because they do not take into account other relevant parameters that affect packet processing, such as, the forwarding state or the broadcast control traffic needed to set up the paths. This section aims to illustrate that TFE may help improve TCP-Path scalability when working with very large scenarios where the number of concurrent flows is huge and therefore so is the control traffic needed to discover and set up the paths

and the number of entries to be stored at switches. In both cases, we first theoretically analyze the issue and then provide some experimental results to validate the study.

1) FORWARDING TABLE SIZE

TCP-Path sets up the path for every flow by adding one entry in the TCP-Path forwarding table of every switch traversed by the selected path. In contrast, TFE only establishes the path for *elephant* flows, which compromise a small fraction of the total (about 5% considering the data in Fig. 6b). Thus, we can conclude that TFE adds far fewer entries than TCP-Path.

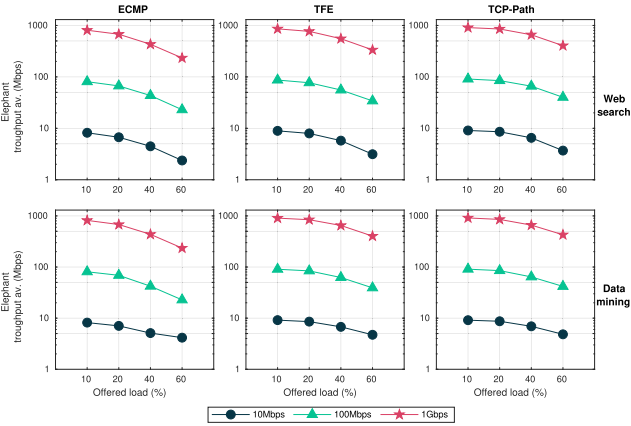


FIGURE 14. 10-100-1000 Mbps link rate comparison.

However, we also have to take into account that the rest of the TCP flows (rabbit and mouse) must be forwarded in some way, as well as non-TCP flows. This task is left to the legacy (default) forwarding protocol, ARP-Path in our evaluation, which relies on another table, the ARP learning table, that switches must maintain in all cases. Therefore, the main difference in the state maintained by the switches is the size of the TCP-Path forwarding table, and TFE drastically reduces this.

To validate this conclusion, we measure the average number of entries in the TCP-Path forwarding table on the ns 3 simulation platform. Using the same set of experiments designed to characterize and compare the performance of TCP-Path and TFE, we sample the table size every second to later compute the average. It is also important to note that we set a 10-second aging timeout for each unused entry.

Fig.16 shows the average number of entries in the TCP-Path forwarding table obtained for TCP-Path and TFE. TFE table sizes are reduced to about 5.6% for Web Search and 5.1% for Data Mining distributions. These figures are slightly above the expected 5% because, as we explained regarding Fig. 9, elephant flows last a bit longer in the network with TFE than with TCP-Path due to the impact of the other flows (rabbit and mouse) it does not handle. Again, this effect has a larger impact when working with the Web Search distribution where rabbit flows account for a substantial share of the traffic. On the whole, we reduce the size of the TCP-Path forwarding table by a factor of almost 20.

## 2) BROADCAST CONTROL TRAFFIC

The second issue that limits TCP-Path scalability is related to the broadcast overhead incurred to set up the paths. Let's consider the worst case scenario, where hosts do not use ARP cache entries (we assume they are unknown) so that each new flow requires an ARP resolution process (ARP Request broadcast and ARP Reply) as the first step. Thus, TCP-Path broadcasts a control frame to explore the network and find the fastest path available at that moment, and confirms the path with a unicast reply control frame. Therefore, we need

two broadcast frames and two unicast frames to set up the path. This is also applicable to TFE, but only for elephant flows (5% of total flows). The remaining flows are handled by ARP-Path (default forwarding), which relies on the ARP resolution exchange to set up the path; thus, no new control frames are needed. Hence, if we omit the ARP resolution process, the only difference remaining between TCP-Path and TFE control overheads is related to the second exploration, which is always performed in the former case and only for 5% of flows in the latter.

Let's now consider a Spine-Leaf topology made up of  $N_S$  spine switches and  $N_L$  leaf switches. When a new TCP flow needs to be installed in the network, the leaf switch serving the source host broadcasts a request packet to explore the network; thus, each spine switch receives the request and broadcasts it down to all the leaf switches except the originating one. Therefore, every leaf switch other than the originating one receives and processes  $N_S$  copies of the original request. Considering that all flows are inter-leaf and that the destination leaf switch is randomly selected, we can compute the total amount of broadcast control bytes overhead (OH) that a single leaf switch needs to process on average, as the total number of copies of the request frame received at leaf switches (for a flow) multiplied by the size (in bytes) of that frame ( $L_B$ ) and the total number of flows to be installed ( $N_F$ ) and then divided by the number of leaf switches:

$$Bcast\_OH_{LEAF} = \frac{N_S \cdot (N_L - 1)}{N_L} \cdot L_B \cdot N_F \quad (1)$$

The above equation is also valid for TFE simply by applying the 5% factor to  $N_F$ .

We can also compute the amount of data bytes to be processed by each switch leaf in a similar way. Given that each data packet needs to be processed by two leaf switches (source and destination edges), it can be computed as twice the total number of flows multiplied by the average length ( $L_F$ , in bytes) of a flow and divided by the number of leaf switches:

$$Ucast\_Data_{LEAF} = \frac{2 \cdot N_F}{N_L} \cdot \bar{L}_F \quad (2)$$

It is worth noting that we do not consider the impact of ACK frames because these depend on stack implementation and amount to a very small total when compared with data bytes or retransmissions due to frame losses. Regardless, the impact on the final figure of merit that we compute below is a worst-case scenario.

Finally, we compute the broadcast control to unicast data ratio as:

$$\frac{Bcast\_OH}{Ucast\_Data} = \frac{N_S \cdot (N_L - 1)}{2} \cdot \frac{L_B}{\bar{L}_F} \quad (3)$$

Taking into account that the broadcast request frame length is 64 bytes and the average flow size in Web Search and Data Mining distributions is 1.90 MBytes and 2.97 MBytes, respectively, Table 2 shows the theoretical ratio values

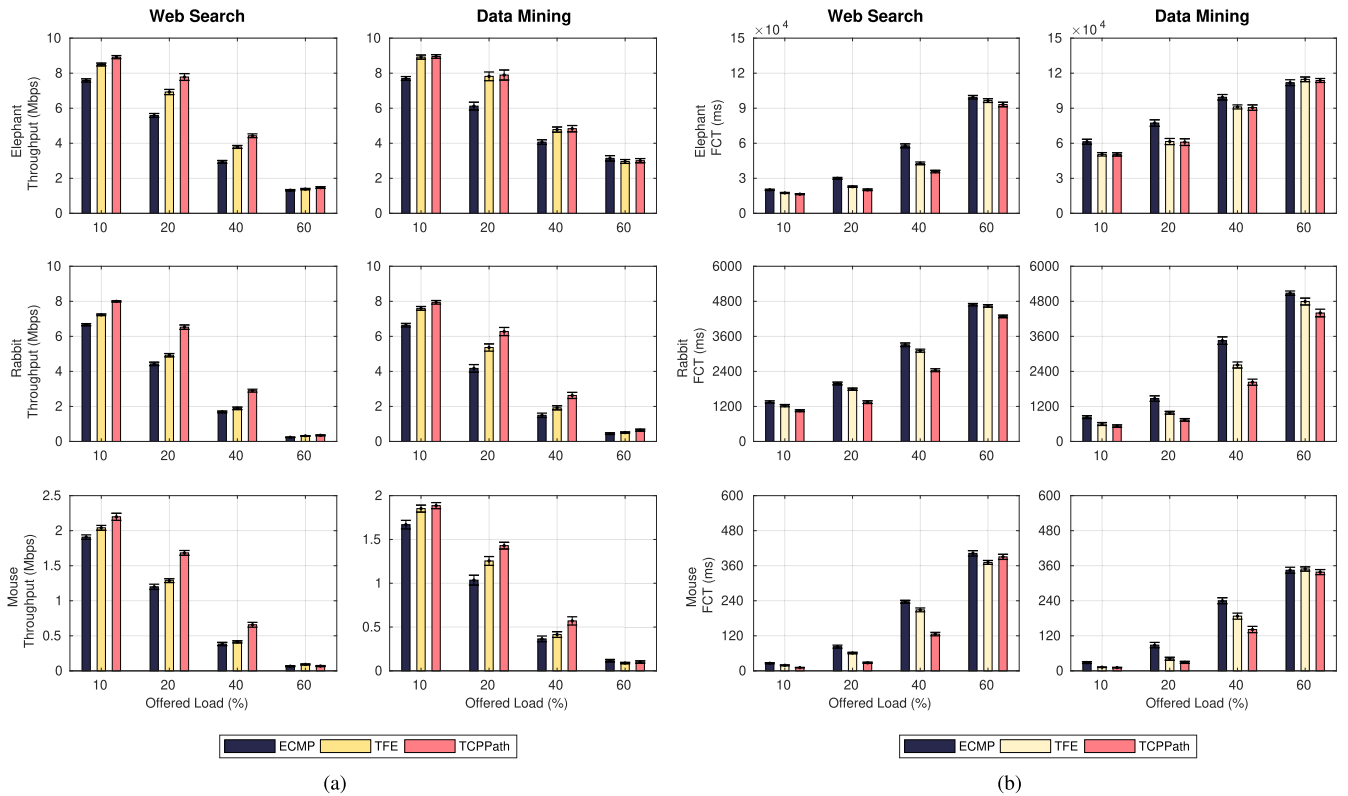


FIGURE 15. Three-layer 10 Mbps topology results. (a) Throughput. (b) FCT.

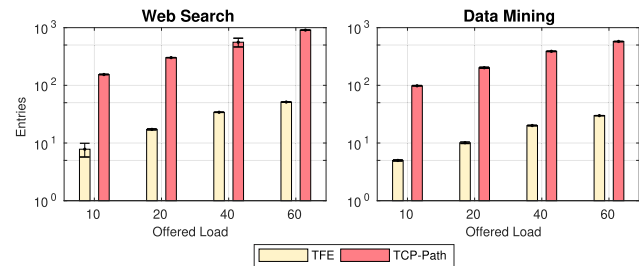


FIGURE 16. TCP-Path table size on Spine-Leaf (4-4-20) 1 Gbps topology.

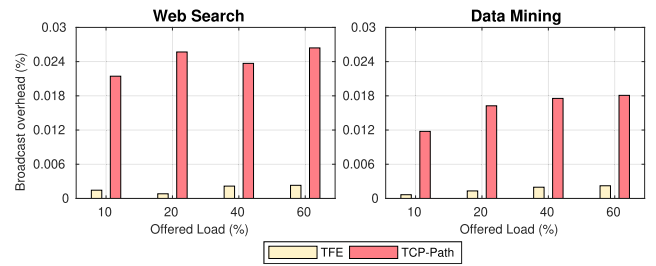


FIGURE 17. Broadcast control to unicast data ratio on Spine-Leaf (4-4-20) 1 Gbps topology.

TABLE 2. Theoretical broadcast control to data bytes ratio at a leaf switch (%).

	$N_S$	$N_L$	TCP-Path	TFE
Web Search	4	4	0.02	0.001
	16	16	0.4	0.020
	64	64	6.8	0.34
	128	128	27.4	1.4
Data Mining	4	4	0.014	0.001
	16	16	0.28	0.014
	64	64	4.8	0.24
	128	128	19.3	0.96

obtained for TFE and TCP-Path for increasing Spine-Leaf topology sizes.

We can see that the ratio is almost 20 times lower for TFE than for TCP-Path and stays at the 1% threshold even for a 128 leaf switch scenario.

To validate the theoretical study we also measure the traffic (broadcast and unicast) processed at leaf switches in our ns-3 simulator. Fig. 17 shows the average results obtained for a Spine-Leaf 4-4-20 topology with 1 Gbps link rates. It is important to note that these are real figures, and we cannot separate the different types of broadcast traffic or the data traffic due to Acknowledgment (ACK) or frame retransmissions. However, the results are fairly similar to those shown in Table 2.

VI. CONCLUSIONS

TCP-Path is a new switching protocol for enterprise and data center networks. Based on simultaneous exploration of all network paths, it selects the fastest available path.

Compared with path computation protocols, TCP-Path is simple and completely transparent to hosts. Moreover, it distributes traffic among alternative paths more efficiently than ECMP, which randomly selects an existing path instead of the fastest available one. We found that TCP-Path improved the throughput obtained for all kinds of flow (elephant, rabbit, and mouse). The throughput improvement for elephant flows, the most important ones, reached up to 70% for the highest offered loads. Moreover, the FCT was also improved for all flows. In mouse flows, where the FCT is a critical parameter since it has an important economic impact, the decrease in FCT was over 60% for medium and higher offered loads, which is substantial. To validate the accuracy of our ns-3 simulation model of TCP-Path, we developed a software switch implementation and tested it with Mininet at lower link rates (due to hardware limitations).

We devised TFE (TCP-Path for elephant flows), which restricts the application of TCP-Path to a subset of flows (elephant flows) in order to improve TCP-Path scalability to deal with larger scenarios. TFE reduces both the overhead incurred during path setup and the size of the forwarding tables by a factor of almost 20. We included TFE in our ns-3 and Mininet comparisons and found that it achieves results close to those obtained by TCP-Path for elephant flows, especially when working with high loads and the Data Mining distribution, and yields significant improvements for all types of flow at medium and high load levels, which is noteworthy because it does not handle these flows.

## ACKNOWLEDGMENT

The source code associated with this paper can be found at <https://github.com/gistnetserv-uah/TCP-PATH>.

## REFERENCES

- [1] *Transparent Interconnection of Lots of Links (Trill)*. Accessed: Jan. 22, 2019. [Online]. Available: <https://datatracker.ietf.org/wg/trill charter/>
- [2] *802.1AQ—Shortest Path Bridging*. Accessed: Jan. 22, 2019. [Online]. Available: <http://www.ieee802.org/11/pages/802.1aq.html>
- [3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in *Proc. 7th USENIX*, 2010, p. 19.
- [4] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, “Improving datacenter performance and robustness with multipath TCP,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, 2011.
- [5] M. Alizadeh *et al.*, “CONGA: Distributed congestion-aware load balancing for datacenters,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, 2014.
- [6] E. Rojas *et al.*, “All-Path bridging: Path exploration protocols for data center and campus networks,” *Comput. Netw.*, vol. 79, pp. 120–132, Mar. 2015. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1389128615000055>
- [7] A. Greenberg *et al.*, “VL2: A scalable and flexible data center network,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.
- [8] M. Alizadeh *et al.*, “Data center TCP (DCTCP),” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 63–74, Oct. 2011.
- [9] J. Qadir, A. Ali, K.-L. A. Yau, A. Sathiseelan, and J. Crowcroft, “Exploiting the power of multiplicity: A holistic survey of network-layer multipath,” *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2176–2213, 4th Quart., 2015.
- [10] J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, and I. Martinez-Yelmo, “TCP-path: Improving load balance by network exploration,” in *Proc. IEEE 6th Int. Conf. Cloud Netw. (CloudNet)*, Sep. 2017, pp. 1–6.
- [11] D. Thaler and C. E. Hopps, “Multipath issues in unicast and multicast nexthop selection,” Tech. Rep. RFC 2991, Nov. 2000. Accessed: Jan. 22, 2019. [Online]. Available: <http://tools.ietf.org/rfc/rfc2991.txt>
- [12] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement.*, Berkeley, CA, USA: USENIX Association, 2010, p. 19.
- [13] T. Koponen *et al.*, “Onix: A distributed control platform for large-scale production networks,” in *Proc. 9th USENIX Conf. Operating Syst. Design Implement.*, Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1924943.1924968>
- [14] *ONOS—Open Network Operating System*. [Online]. Available: <http://onosproject.org/>
- [15] F. X. A. Wibowo, M. A. Gregory, K. Ahmed, and K. M. Gomez, “Multi-domain software defined networking: Research status and challenges,” *J. Netw. Comput. Appl.*, vol. 87, pp. 32–45, Jun. 2017.
- [16] S. Wang, J. Zhang, T. Huang, T. Pan, J. Liu, and Y. Liu, “Flow distribution-aware load balancing for the datacenter,” *Comput. Commun.*, vol. 106, pp. 136–146, Jul. 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0140366417303043>
- [17] S. Kandula, D. Katabi, S. Sinha, and A. Berger, “Dynamic load balancing without packet reordering,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007.
- [18] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, “Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing,” in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, New York, NY, USA, 2013, pp. 151–162. [Online]. Available: <http://doi.acm.org/10.1145/2535372.2535397>
- [19] S. Vutukury and J. J. Garcia-Luna-Aceves, “A simple approximation to minimum-delay routing,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 227–238, Aug. 1999. [Online]. Available: <http://doi.acm.org/10.1145/316194.316227>
- [20] J. Cao *et al.*, “Per-packet load-balanced, low-latency routing for clos-based data center networks,” in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, Santa Barbara, CA, USA, Dec. 2013, pp. 49–60.
- [21] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, “Presto: Edge-based load balancing for fast datacenter networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, Aug. 2015.
- [22] D. Xu, M. Chiang, and J. Rexford, “Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering,” *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1717–1730, Dec. 2011.
- [23] M. Chiesa, G. Kindler, and M. Schapira, “Traffic engineering with equal-cost-multipath: An algorithmic perspective,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 779–792, Apr. 2017.
- [24] M. Shafiee and J. Ghaderi, “A simple congestion-aware algorithm for load balancing in datacenter networks,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3670–3682, Dec. 2017.
- [25] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, “HULA: Scalable load balancing using programmable data planes,” in *Proc. Symp. SDN Res.*, New York, NY, USA, 2016, pp. 10:1–10:12. [Online]. Available: <http://doi.acm.org/10.1145/2890955.2890968>
- [26] D. Katz and D. Ward, “Bidirectional forwarding detection (BFD),” Tech. Rep. RFC 5880, Jun. 2010. [Online]. Available: <http://tools.ietf.org/rfc/rfc5880.txt>
- [27] D. Lopez-Pajares, J. Alvarez-Horcajo, E. Rojas, G. Ibanez, and J. A. Carral, “Iterative discovery of multiple disjoint paths in switched networks with multicast frames,” in *Proc. 43rd IEEE Conf. Local Comput. Netw. (LCN)*, 2018.
- [28] J. Alvarez-Horcajo, I. Martinez-Yelmo, E. Rojas, J. A. Carral, and D. Lopez-Pajares, “New cooperative mechanisms for Software Defined Networks based on Hybrid Switches,” *Trans. Emerg. Telecommun. Technol.*, vol. 28, no. 8, p. e3150, 2017.
- [29] *NS-3 Simulator*. Accessed: Jan. 22, 2019. [Online]. Available: <https://www.nsnam.org/>
- [30] *CPqD: OpenFlow 1.3 Software Switch*. Accessed: Jan. 22, 2019. [Online]. Available: <https://github.com/CPqD/ofsoftswitch13>
- [31] *Mininet: An Instant Virtual Network on Your Laptop (or Other PC)-Mininet*. Accessed: Jan. 22, 2019. [Online]. Available: <http://mininet.org/>
- [32] M. Alizadeh *et al.*, “pFabric: Minimal near-optimal datacenter transport,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, Aug. 2013.

- [33] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *Proc. Opt. Interconnects Conf.*, May 2013, pp. 49–50.
- [34] J. Zhang, F. Ren, and C. Lin, "Survey on transport control in data center networks," *IEEE Netw.*, vol. 27, no. 4, pp. 22–26, Jul./Aug. 2013.
- [35] *OpenFlow Switch Specification v1.3.2*. Accessed: Jan. 22, 2019. [Online]. Available: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf>



**J. ALVAREZ-HORCAJO** (M'17) received the master's degree in telecommunications engineering from Alcalá University, in 2017. After having worked at Telefonica as a Test Engineer for COFRE and RIMA networks, he received a grant for university professor training (FPU) at Alcalá University. He has participated in various competitive projects funded through the Community of Madrid Plan (TIGRE5). His research interests include software-defined networks (SDN), the Internet protocols, and new generation protocols. At present, he is especially interested in topics related to advanced switches and SDN networks.



**D. LOPEZ-PAJARES** (M'17) received the master's degree in telecommunications engineering, in 2017. He has been a Researcher with the GIST-NETSERV research group, since 2015, focusing on topics related to delay-tolerant and SDN networks, specifically low-latency routing solutions and the multipath problem. He is currently pursuing the Ph.D. degree with the basis of above-mentioned topics. In addition, he actively participates in several GIST-NETSERV research projects, such as TIGRE5-CM, EsPECIE, and SIMPONS.



**I. MARTINEZ-YELMO** received the Ph.D. degree in telematics from the Carlos III University of Madrid, Spain, in 2010. After working as a Postdoctoral Assistant with the Carlos III University of Madrid, he became and remains as a Teaching Assistant with the Automatics Department, Alcalá University, Spain. His research interests include peer-to-peer networks, content distribution networks, vehicular networks, NGN, and the Internet protocols. Nowadays, he is especially interested in advanced switching architectures and software-defined networks. He has participated in various competitive research projects funded by the Madrid Regional Government (Medianet, Tigre5), National projects (CIVTRAFF), and European projects (CONTENT, CARMEN, and so on). His research papers have been published in high-impact JCR indexed research journals, such as the *IEEE Communications Magazine*, *Computer Communications*, and *Computer Networks*, among others. In addition, he has been a Reviewer for high quality conferences (i.e., IEEE INFOCOM) and scientific journals (the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, *Computer Communications*, the *ACM Transactions on Multimedia Computing*, and so on). He was a Technical Program Committee Member of IEEE ICON, from 2011 to 2013.



**J. A. CARRAL** received the degree in telecommunications engineering from the Polytechnic University of Madrid (UPM), in 1993, and the Ph.D. degree from Alcalá University, in 2013, with a focus on advanced Ethernet switches. He was a Research Assistant with the Telematics Engineering Department, UPM, and was granted a four-year FPU-MEC (1994–1998) by the Spanish Ministry of Science and Education. He joined the Polytechnic School, Alcalá University, as a Teaching Assistant, in 1998, where he became a Teacher, in 2001, later promoted to Associate Professor. He currently collaborates with the Madrid Foundation as a member of the VERIFICA panel of experts. He has participated in several regional, national, and international (EU) research projects both at the UPM and Alcalá University. He has published over a dozen articles in selected international journals in the field of telecommunications and has participated in several international and national conferences.



**J. M. ARCO** received the Ph.D. degree in telecommunications engineering from Alcalá University, Spain, in 2001, where he has been an Associate Professor with the Automatica Department (Telematics Area), since 2002. He also teaches SDN and data center networks. He has participated in various competitive research projects funded by the Madrid Regional Government (Medianet, Tigre5), national projects (CIVTRAFF), and European projects (ALFA). His research papers have been published in high-impact JCR indexed research journals, such as the *IEEE COMMUNICATION LETTERS*, *Computer Communications*, and the *Annals of Telecommunication*. His current research interests include SDN, high-performance and scalable Ethernet, and data center networks. In addition, he has been a Reviewer for the high-quality scientific journal, the *IEEE Communication Magazine*.

...