uc3m | Universidad Carlos III de Madrid

Master on Telematics Engineering
Academic Year 2016/2017

Master Thesis

SEMPER: A Stateless Traffic Engineering Solution based on MP-TCP for WAN networks

Ginés García Avilés

Director
Pablo Serrano Yáñez-Mingot
Leganés, 28th of September 2017

Keywords: WAN, Traffic-Engineering, Multipath-TCP
Abstract: Current Enterprise Networks deployments must accomplish a strong set of requirements in terms of resiliency, reliability and resources usage. As current approaches are based on monolithic and expensive infrastructures built on dedicated overlay links, vendors are trying to move to economical hybrid solutions that encompasses private dedicated links with public/regular Internet connections. However, these usually rely on complex Traffic Engineering solutions hardware dependent or proprietary that are costly in terms of computational time and memory usage in the forwarding nodes. In this paper, we propose SEMPER: a lightweight traffic engineering (TE) solution based on MP-TCP which, in contrast to other TE solutions, moves the complexity to the endpoints of the connection, and relieves the forwarding elements from complex operations or maintaining state. As our evaluation shows, SEMPER efficiently makes use of all available paths between the endpoints while maintaining fairness, and properly adapts to variations on the available capacity.

SEMPER: A Stateless Traffic Engineering Solution based on MP-TCP for WAN networks

Gines Garcia-Aviles

***Abstract*—Current Enterprise Networks deployments must accomplish a strong set of requirements in terms of resiliency, reliability and resources usage. As current approaches are based on monolithic and expensive infrastructures built on dedicated overlay links, vendors are trying to move to economical hybrid solutions that encompasses private dedicated links with public/regular Internet connections. However, these usually rely on complex Traffic Engineering solutions hardware dependent or proprietary that are costly in terms of computational time and memory usage in the forwarding nodes. In this paper, we propose SEMPER: a lightweight traffic engineering (TE) solution based on MP-TCP which, in contrast to other TE solutions, moves the complexity to the endpoints of the connection, and relieves the forwarding elements from complex operations or maintaining state. As our evaluation shows, SEMPER efficiently makes use of all available paths between the endpoints while maintaining fairness, and properly adapts to variations on the available capacity.**

I. Introduction

Resiliency and Fault Tolerance are two of the most important requirements for enterprise wide area networks (WAN), and are becoming even more important in recent days, where the availability of different network links is the clear representation of the current "meshification" trends.

The enterprise wide area networks paradigm enables the creation of a virtual private area network (VPN) linking different branches of an enterprise with its headquarters. With the increasing availability of different (physical) paths among different offices, Traffic Engineering solutions that exploit multiple paths are currently very used and there is a lot of research effort focused on the improvement of the state of the art solutions. SD-WAN [1] is an example of "hands on" product that manages traffic over multiple links.

The most common enterprise WAN deployment consists on edge routers located in different campuses, called branches, and different links that interconnect them with a central headquarter. Typically, those solutions are based on private overlay models [1], [2].

However, to achieve increased resiliency, reliability and an optimal usage of the resources for computing assets distributed across several locations, these solutions have to rely on monitoring systems to assess the traffic level of each link at any time. This operation, which is essential to forward/reroute flows to the best path, is usually costly in terms of computation time and prone to errors as it is based on traffic probes. In this paper, we propose a novel traffic engineering technique, that overcomes the disadvantages of stateful approaches to provide efficient connectivity between edge and central headquarters.

Our solution is based on moving the complexity to the network endpoints (in this case, the end hosts), by using the multipath version of TCP (MP-TCP) [3]. By exploiting its congestion control capabilities, it simplifies the operation of edge routers in both branches and headquarters.

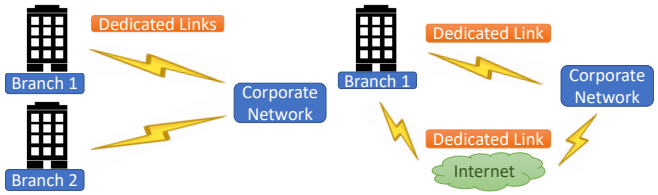
The rest of this paper is organised as follows. In Section II we review the evolution of the enterprise network deployments together with the different Traffic Engineering approaches applied. In Section III we discuss the use of MP-TCP as a traffic engineering (TE) solution, including its potential benefits and challenges. In Section IV we detail the design of our TE solution (SEMPER), a light-weight solution to efficiently and fairly distribute flows across existing paths between branches of the enterprise WAN. Section V describes the setup and methodology to perform the evaluation, which is provided in Section VI, showing the benefits of SEMPER. Finally, Section VII concludes the paper.

II. Background

As introduced in Section I, current enterprise WAN deployments are moving from using expensive dedicated links (as depicted in Fig. 1a) to different solutions that exploit shared and possibly disjoint links (Fig. 1b), that usually provide path diversity between branches and headquarters.

The aim of reducing link associated costs while maintaining reliability, availability and performance leads to a new approach called "Hybrid WAN". The *Hybrid WAN* architecture uses different connection types for the different links that conforms the deployment. In other words, hybrid WAN combines the usage of dedicated links with shared Internet connections, providing a more cost-effective and versatile way to connect all branch offices. With the latter approach, the enterprise still have a dedicated link for critical traffic and rely on regular Internet connections (broadband) for non-critical traffic Fig. 1b. Usually, the traffic is routed through a specific link depending on its criticality by a TE solution deployed at the edge routers. The current architectural approaches envision to the usage of redundant broadband links in order to maintain availability, reliability and performance. This reduces operational costs, as it avoids the usage of dedicated links. In this paper, we consider the latter approach.

Traffic Engineering (TE) is a tool for achieving control over how data packets are forwarded within a network. These techniques usually compute multiples paths among all the source-destination pairs and distribute the traffic load between them. Optimizing the performance is a process that must be



(a) Classic WAN: Dedicated links. (b) Hybrid WAN: shared and dedicated links.

Fig. 1: WAN Deployments

defined at the beginning, because there are different objectives that could be optimized at the same time [4]. TE solutions usually involve a set of components in the network, where the different techniques will be applied. These techniques may follow an adaptive feedback control system [5], providing optimal routing and resources management in the system.

As we pointed out previously, current WAN deployments consists on multiple paths between edge routers, *TE* techniques shall exploit path diversity to increase resiliency and reliability, also improving the overall performance. There are different TE solutions for path assignment in multipath environments, usually performing the assignment using information provided by a monitoring system [6].

Software Defined WAN (SD-WAN) [1] is a commercial solution for multipath enterprise WAN. It is the result of applying the Software Defined Networking (SDN) [7] architecture on a Wide Area Network. This approach shifts the control plane from the physical devices to a logical entity called controller. The controller entity allow the administrator to remotely configure the edge routers on each branch to change the behaviour of the network.

Equal-Cost-MultiPath (ECMP) [8] is a technique to perform packet routing among multiple paths with equal cost to reach a destination. In WAN, shortest path protocols are usually used to obtain the available paths between two edge routers (or configure MPLS labels in case of MPLS-enabled deployment). Then, ECMP runs over the edge routers and balance the load between them. In other words, ECMP exploits shortest-path diversity by splitting the traffic between them [6].

Usually, ECMP distributes traffic homogeneously among all the equal-cost paths, it does not take into account the level of congestion on each path. Without being aware of the congestion, it is possible that one path is more congested than the others leading to inefficiencies. In addition, ECMP cannot offload traffic to paths with different cost, because it only splits traffic between paths with the same cost.

For the reasons explained above, applying ECMP on the edge routers is not a valid solution. Conversely, Shifting the control plane to a logical entity, as SD-WAN proposes, allows the combination of different types of links (MPLS, broadband, wireless, ...) in the same WAN. This reduces or even avoids the usage of expensive dedicated links as it redirects non-sensitive traffic over cheaper links. In the end, it optimizes the usage of the resources having a centralized view of the network.

SD-WAN is, independently of the forwarding strategy, a stateful solution: the stored state information grows with the number of flows that are being managed. In addition, it requires a monitoring system, as the decision on the path to be selected is usually taken considering its congestion level. This is achieved by sending periodic small packets to the destination to infer/measure the congestion. As a result, a solution such as SD-WAN increases the CPU usage and needed storage making routers (and hence an enterprise WAN deployment) more expensive.

The TE solution proposed by this work combines the multi-flow approach of Multipath TCP [3] with the newest multi-path approaches for WAN deployments and it relies on MP-TCP to perform traffic offload to less congested links. Our solution performs a stateless and efficient path assignment, exploiting the heterogeneity of the multi-path network. In addition, it avoids the usage of monitoring systems and stateful strategies for path assignment, reducing the computational and memory requirements of the hardware deployed at the edge.

III. Using MP-TCP for Traffic Engineering

As discussed above, efficiently exploiting the availability of multiple (and possibly heterogeneous) paths is still an open research problem, despite the numerous attempts available in the literature. Since the introduction of Equal Cost Multi-Path routing (ECMP) [8], the research community strove to find a mechanism to efficiently deal with the availability of multiple paths. By "efficiently" we refer to solving a number of challenges entailed by the multi-path approach, such as, e.g., packet reordering, load balancing or fast re-routing in case of failures. In general, the main difficulty is that the solutions to these problems are complex to implement, and therefore multi-path is not a fully exploited paradigm in nowadays networks.

However, the introduction of MP-TCP [3] could completely change this situation. By coupling the congestion control of multiple subflows, MP-TCP can efficiently handle the shortcomings that the use of multiple paths may introduce, dynamically balancing the transmission window over different subflows [9]. However, despite the clear advantages of an extensive deployment of MP-TCP, its far from widely adopted, relegated to a few remarkable use-cases such as e.g. Apple Siri [10] or Proxy enhancements [11]. Still, even counting for these examples, for the case of end user applications the focus is typically on a couple of heterogeneous interfaces (i.e., a cellular and a WLAN), and not in the design of generic solutions that scale up to many paths.

These cases discussed above are typically based on one of the two possible modes of operation supported by MP-TCP, namely, the `full-mesh` mode, which supports exploiting path diversity by generating one flow per path available at the source node. The other mode of operation is the `ndiffports` mode, which generates multiple flows at the source irrespective of the number of interfaces available, and there relies on subsequent hops to exploit path diversity.

Indeed, MP-TCP `ndiffports` has already been proven an effective solution for single-homed hosts in datacenter

networks [12]. This motivates applying a similar strategy for this scenario can be a valid decision for solving the problems introduced in Section I. By activating MP-TCP on all the end users within the branch, the outgoing traffic from the branch is already "multipath-ready", so an edge router can effectively exploit multiple paths at a relative low cost by forwarding sub-flows to the different paths.

One of the main benefits of this approach (that we will detail in the next section) is that all the complexity of former traffic engineering solutions, based on complex monitoring operations (e.g., actively assessing the link capacity) and the corresponding load balancing schemes, is pushed to the branches of the network, as MP-TCP runs on the end hosts. With this approach, the design of the forwarding strategy (i.e., the path assignment) to be implemented at the edge routers becomes of paramount importance. We remark that it is a different problem from the one studied for ECMP forwarding, as we are not considering path assignment of single path flows, but rather path assignment of sub-flows belonging to same "parent" multipath flow.

In the next section, we design a solution for Traffic Engineering, which we refer to as SEMPER (StatEless Multi Path forwarding for Edge Routers). Two outstanding features of SEMPER are:

- It does not require any knowledge of the topology, this including key variables such as the number of disjoint paths between any two sites. This is in contrast with other solutions that might require the knowledge of this variable, to fix the number of sub-flows that have to be generated by MP-TCP accordingly.
- It does not require a complex scheme for the distribution of sub-flows between paths. Again, this is in contrast with other solutions that, knowing the "optimal" number of sub-flows, introduce the added complexity of a scheme to properly balance these among the set of available paths.

IV. SEMPER: A TE SOLUTION BASED ON MP-TCP

We next design SEMPER, a TE solution that does not require the knowledge of the topology nor the use of complex sub-flow balancing schemes on the forwarding elements. We first describe the MP-TCP configuration used in the end hosts, and then the simple forwarding functionality that the end routers connecting each of the branches need to implement.

To describe the changes to the end hosts, we start by describing MP-TCP by means of its main components, which are depicted in Fig. 2. The first component is the *path-manager*, which is responsible for the TCP subflows that will conform the MP-TCP connection. As described before, this component supports different policies: the `full-mesh` policy creates a sub-flow for each available (source IP, destination IP) pair, while with the `ndiffports` policy the host will create a concrete number of sub-flows for the same pair of IP addresses, changing the source port.[1]

[1]There is a also a `default` policy, where no flows are created but only accepted.

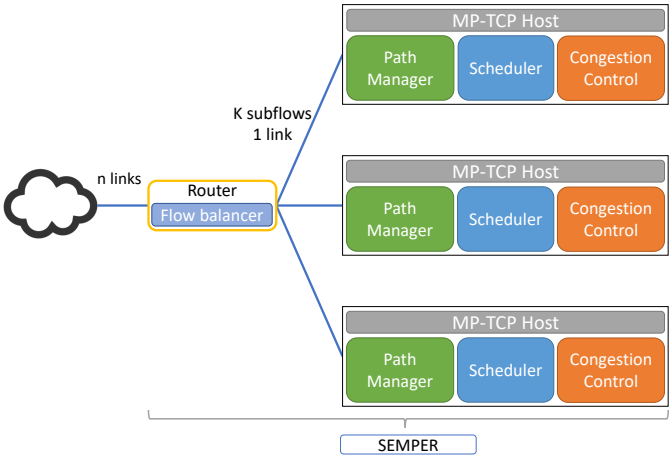


Fig. 2: SEMPER reference architecture

The second component is the *scheduler*. This module assigns higher-layer TCP segments over the existing sub-flows, taking into account the different characteristics of each subflow (e.g., the RTT), trying to optimise the overall performance. Here, among the different policies available, there are three worth mentioning: `default`, where paths with lowest RTT are preferred; `roundrobin`, which implements such policy across sub-flows, and `redundant`, where the same information is transmitted over all existing interfaces (for instance, this is the one used by Siri in Apple's iPhones, to maximise reliability). Finally, the *congestion control module* implements the congestion control algorithm, that computes the congestion window to be used on each existing sub-flow.

Number of sub-flows per host to generate

The first challenge when designing SEMPER is to select the proper number of sub-flows that each MP-TCP host has to generated, which we denote as K. Given an unknown number of available paths P between the two branches, the actual value of K that should be used has an impact on performance, depending on the number of hosts running in the branch n

- If $n \times K < P$, then not all capacity is used, as there are not enough sub-flows to occupy all the available paths.
- if $n \times K \geq P$, there are at least one sub-flow per available path, but there are two pending challenges: (i) how to assign sub-flows to paths, and (ii) if there is any penalty when using more sub-flows than the number of available paths.

We will discuss the first challenge in the next section. As for the second challenge, in our experiments we demonstrate that there is no major penalty in using an overly large value of K, which results in the following key building block for SEMPER:

> Each MP-TCP end host generates $K = 32$ sub-flows.

Mapping of sub-flows to paths

The other functionality to be designed, to run in the edge router of each branch, is the *flow balancer*. This module is responsible for assigning each sub- flow to one of the available paths. As mentioned, while this functionality falls outside MP-TCP, it is one the main components of SEMPER. This module is placed in the edge router, as illustrated in the architecture depicted in Fig. 2.

One of the key objectives of the designed solution is a reduced complexity, leveraging on the end hosts running MP-TCP. To continue with this reduced complexity, we discard the use of passive or active monitoring, and will leverage on MP-TCP's congestion control to properly adapt to the available capacity. More specifically, we will rely on MP-TCP for efficiently and fairly distribute resources among the hosts, and design a very simple flow balancer. The only requirement for this flow balancer is that TCP segments and acknowledgements from the same sub-flow have to be always assigned to the same path, so the congestion control can properly react.

The SEMPER flow balancer is based on performing a hash function (Fig. 3) on the 4-tuple composed by the source and destination IP addresses and port numbers, and map this hash to one of the available paths (numbered from 1 to n). For simplicity, we use a simple hash function based on the modulo operation, which will then map a sub-flow to a path with a probability $1/n$. We implement the corresponding (reverse) mapping function at the router at the other side, to support the required path symmetry.

> Use a hash function to randomly assign a TCP sub-flow to a path.

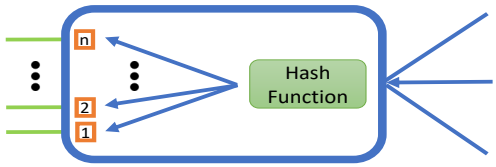


Fig. 3: SEMPER forwarding strategy

While this approach results in an extremely light-weight solution, there are two key issues that might result on non-optimal performance, and we will discuss in the performance evaluation:

1) Due to the random assignment of sub-flows to paths, the complete utilization of all the available path is not guaranteed (even when there are more sub-flows than paths).
2) Sub-flows belonging to the same MP-TCP connection may not all follow disjoint paths.

Indeed, source port generation is an OS dependent procedure that is regulated by a recommendation only [13]. However, source port number generation. Despite some remarkable exception such as some Windows flavours [14], almost all the state of the art solutions are using random hash values to generate the source port, either at a global level (i.e., different processes share the same random seed) or at local level (one random seed per process). Therefore, we cannot use any predictive technique to guess the "next source port" for a given port, and we have to rely on a purely random process to avoid keeping any state.

Benchmark: a stateful solution

To have a proper performance comparison during our experiments, we designed an alternative approach to SEMPER, which will serve as a reference benchmark for the use of MP-TCP for traffic engineering. This approach consists on the following configuration:

- Each MP-TCP host is aware of the number of existing paths between the end points P, and generates one sub-flow per path, i.e., $K = 32$.
- The *flow balancer* module distributes à la round-robin each of these sub-flows across the available paths.

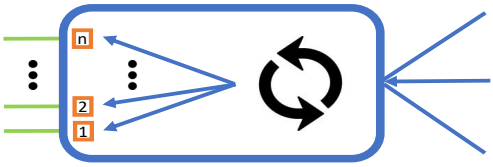


Fig. 4: Stateful forwarding strategy

With this approach, we guarantee that all sub-flows are evenly distributed across all the available paths (Fig. 4). To distribute the load of the TCP handshakes across paths, the first sub-flow from a given host is hashed as in SEMPER, while the next sub-flows are assigned sequentially (modulo n, the total number of paths). Note that the price to pay for this approach, which evenly distributes flows across paths, is to keep state at both routers.

V. Testbed and Methodology

Our motivation is to design a solution that can be immediately implemented in a real-life testbed. However, given the number of hosts and paths considered during our experiments, the cost of deploying an actual testbed of the required size would be prohibitive. Because of this, we decide to make use of virtualisation techniques to evaluate the performance under different conditions, which enables using the *real* software implementation of the required modules.

We evaluate SEMPER against the stateful counterpart mainly along three dimension: throughput performance, throughput distribution fairness and fault tolerance. In the following we describe the used hardware and software configuration and the employed methodology to compute the evaluation results.

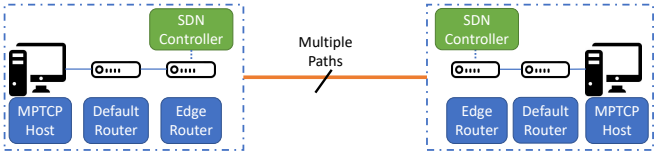


Fig. 5: Emulated topology throughout our experiments

A. Hardware and software configuration

Our virtualisation environment runs on top of desktop PC, equipped with an AMD FX(tm)-8320 8-Core Processor and 32 GB of RAM. It runs the `Mininet` network emulator [15] to create a network consisting of virtual hosts, switches, controllers, and links, where hosts run standard Linux network software. All the end hosts run a kernel implementation of Multipath TCP, while the switches run Open Virtual Switch, supporting OpenFlow . This configuration[2]. results highly flexible for both routing and forwarding, which is very convenient for the implementation of SEMPER and the automation of experiments, as we will describe in the next section. The software framework is depicted in Figure 5.

As illustrated in Fig. 5, we will focus on the classical *dumbbell* topology, where two branches are connected via multiple disjoint paths. One of this routers (the one on the left) is connected to a "default router," that serves a number of hosts, each running MP-TCP. The other border router is connected to an Iperf server running MP- TCP. All routers run also the lightweight `RYU` SDN controller, which simplifies the implementation of the different subflow forwarding techniques discussed above.

B. Setting up a experiment

We next describe the methodology that we follow to perform one experiment, which is defined by the following configuration parameters:

- Number of end hosts connecting to the server.
- Number of MP-TCP subflows generated by a host.
- Capacity of the links connecting the border routers.
- Traffic engineering solution: (stateless) random forwarding or (stateful) round robin.
- Total duration of the experiment T

For each configuration, the methodology works as follows. First, the right MP-TCP kernel, shared by all machines, is loaded. Then, `Mininet` is launched to create the environment, this including instantiating the required number of hosts, routers, controllers, etc. as well as setting up the various links between entities and the corresponding capacities (for those with finite capacity). Depending on the traffic engineering solution of choice, the SDN controllers are correspondingly configured, following the schemes described in Section IV.

[2]We used Mininet v2.3.0d1, the linux kernel 3.18.20-90-mptcp, OVS v2.0.2, RYU v4.9 and Openflow v1.3.

Once the above is set up, we launch the different MP-TCP connections between the end hosts and the server, and configure them to perform data transmission during the length indicated by the parameter T. Note that our objective is to measure performance in steady-state conditions, i.e., with all flows connected. However, given the relatively large number of subflows that we consider, we found out that depending on the schedule of the subflow starts, the bandwidth of the links and the experiment duration, it could happen that the last subflow to start may send its first SYN way after the first subflow has already finished. Because of this, we cannot rely on the statistics provided by Iperf to assess the performance of the solution, as these might not correspond to steady-state conditions. Further details on the implementation are provided in Appendix.We next describe how we compute the performance for a given experiment.

C. Computing the throughput figures

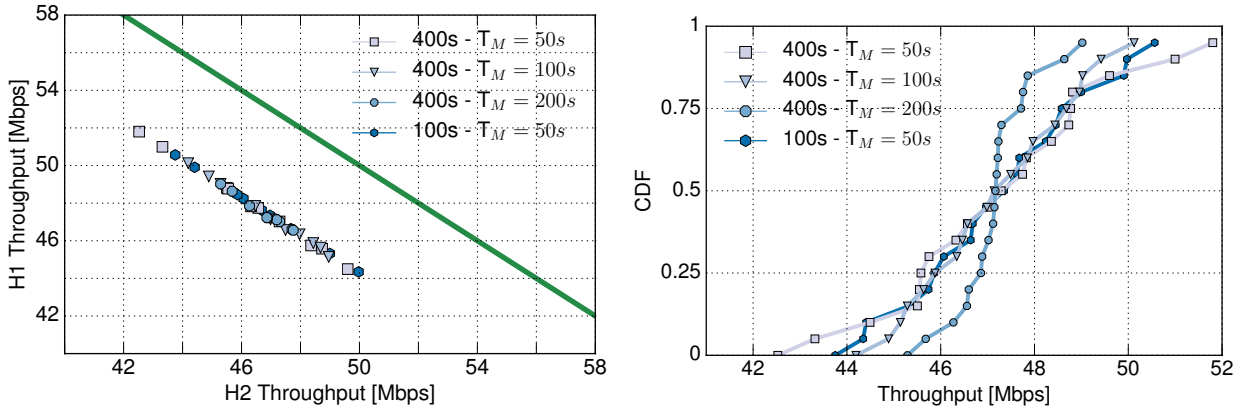
During the execution of a experiment, we store the `tcpdump` traces corresponding to the TCP segments that are sent over all the paths connecting the routers, so then we can extract the statistics corresponding to steady- state conditions. In order to find these conditions in a robust manner, we proceed as follows. We define a measurement period T_M, which is less than the total duration of the experiment T. Given that the first SYN segment is sent at time 0, our objective is to find an adequate window of time T_M where all TCP subflows are connected. This is done as follows:

- For each subflow, we iteratively search in the tracefiles for the last `ACK` segment corresponding to TCP's three-way-handshake.
- If the total number of `ACK`s found corresponds to the expected number of generated subflows, we denote the timestamp of the last `ACK` as T_1 and continue. Otherwise, we restart the experiment.
- We then search in the tracefiles for the first `FIN` of TCP's disconnecting for each subflow.
- If the total number of `FIN`s found corresponds to the expected number of generated subflows, we denote the timestamp of the first `FIN` as T_2 and continue. Otherwise, we restart the experiment.
- Finally, if $T_2 - T_1 > T_M$, we proceed to compute the throughput figures for the period $[T_2 - T_M, T_2]$.

We denote as R_i the throughput obtained by host H_i, which is obtained as the sum of the total number of bytes acknowledged for all its subflows divided by T_M (we note that this results computationally more expensive than counting on *Iperf* reports).

D. Validation

Here we analyse the impact of T and T_M on the obtained throughput figures, to guarantee that we run the experiments long enough and that statistics are gathered over the required periods of time, i.e., a steady-state where MP-TCP is efficiently using the available capacity. To this aim, we consider the dumbbell topology with $N = 2$ paths connecting the



(a) Capacity region for the validation scenario.

(b) CDF of H1 throughput for different measurement times.

Fig. 6: Validation results

two border routers. Each path consists on a 50 Mb/s link (so the total capacity connecting the routers is 100 Mb/s). There are two TCP hosts (H_1 and H_2) and each host generates two subflows, and the forwarding tables are configured such that each of the subflows from the same host goes through a different path.

We consider we following values for the total duration of the MP-TCP sessions, T = {100, 400} s, and the following values for the measurement period T_M = {50, 100, 200, 400} s. For each valid configuration, we repeat the experiment 10 times, and compute the throughput obtained by both hosts, namely, R_1 and R_2. To illustrate the differences between the obtained result and the nominal capacity of both links, we plot the resulting (R_1, R_2) points for all experiments and configurations, which results in the Fig. 6a. We also represent in the figure, with a green continuous line, the limit of the *capacity region* of this scenario, which corresponds to those x and y values such that $x + y = 100$ Mb/s.

As the figure illustrates, for all combinations of T and T_M, the total throughput obtained (R_1+R_2) is very close to the boundary of the capacity region, with an average total throughput of approx. 94 Mb/s in all cases. While the figure confirms the efficiency of MP-TCP when using the available resources, it also illustrates that there is some variability in the resource distribution between H_1 and H_2 across experiments. To investigate the impact of T and $_M$ on this variability, we compute the Cumulative Distribution Function (CDF) of the TCP throughput obtained by one host[3] during 20 repetitions. The results are depicted in Fig. 6b.

According to the results, the use of small values for T_M results in a notable variability across experiments, with throughput differences that span approx. $\pm$ 20%. In contrast, longer T_M values practically halves these differences. For these reasons, during our performance evaluation we will fix $T = 400$ s and $T_M = 200$ s.

[3]We only consider the results for H_1, given that we have seen, in the previous results, that H_2 will receive practically all the remaining capacity.

VI. Performance evaluation

Building on the methodology described above, we next evaluate the performance achieved by SEMPER, and compare it vs. the stateful solution. We start by validating the chosen configuration for the number of subflows per host of the stateful TE solution. After this validation, we first compare the performance in *static scenarios*, analysing the impact of the number of paths connecting the routers, and then in *dynamic scenarios*, where one of the links becomes unavailable at some point in time.

In our performance evaluation, we follow the methodology described in the previous section to compute set of the throughputs obtained by each host, $\{R_i\}$. Based on this set of values, we evaluate the performance based on two variables:

- Total throughput: defined as the sum of the throughput obtained by each of the hosts, which will serve as a measurement of efficiency (the closer to the maximum capacity, the better).

$$\text{Total throughput} \triangleq \sum_i R_i \tag{1}$$

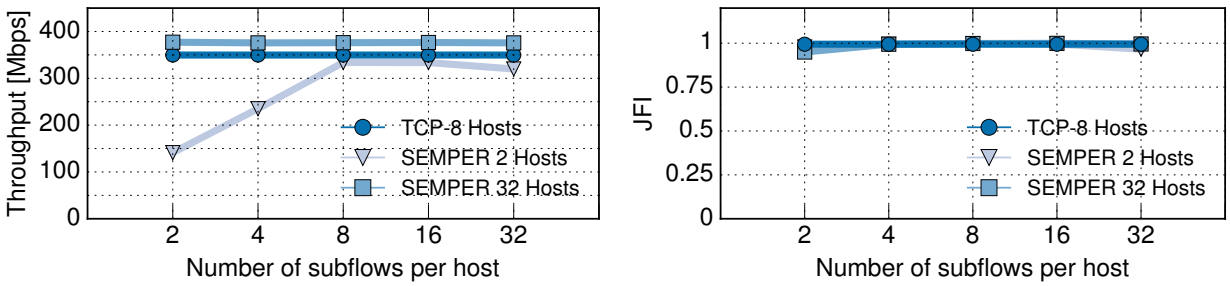
- Fairness: which serves to quantify the evenness in the distribution of the resources, and is computed following Jain's definition [16], i.e.,

$$\text{Fairness} \triangleq \frac{(\sum_i R_i)^2}{N \sum_i R_i^2} \tag{2}$$

  which ranges from one (perfect fairness) to $1/N$.

A. Validation of SEMPER configuration

We first evaluate the performance of SEMPER but using a varying number of generated subflows per host (K), to gather insight on the impact of this parameter and to confirm that our $K = 32$ setting is appropriate. We set-up a dumbbell topology with 8 links of 50 Mb/s connecting the two routers. We consider two different configurations for the number of hosts connecting to the MP-TCP server, namely, $N = \{2, 32\}$, and the following set of subflows per host $K = \{2, 4, 8, 16, 32\}$.



(a) Total throughput (Median)

(b) Jain Fairness Index.

Fig. 7: Validation of SEMPER configuration*: Experiments increasing the number of subflows*

We also compute the performance of scenario in which 8 hosts use a single TCP flow connected to the server, where each flow uses a different path, which will serve for reference purposes (i.e., a baseline).

For each scenario, we repeat each measurement 10 times, and compute the median of the results. We depict these in Fig. 7, illustrating the total throughput (Fig. 7a) and fairness (Fig. 7b) for an increasing number of subflows per host K.

Concerning the *total throughput*, the results show that when the number of hosts is small ($N = 2$) and SEMPER does no use enough subflows ($K < 8$), the performance is notably worse than the baseline. This is caused, of course, by the random assignment of subflows, which fails to occupy all the available paths (and therefore, the capacity). In contrast, when the number of hosts is large ($N = 32$) or the number of subflows is large ($K \geq 8$), the performance is closer to the baseline.

Considering *fairness* results, the performance is remarkably fair for all the considered scenarios, as for all configurations of the experiments the median is well above 0.9. There is a small drop for the case of $N = 32$ hosts and only 2 subflows per host, which is caused by the very few case in which a host has its two subflows sent over the same path (and therefore a relative lower throughput than others). Furthermore, this drop is "corresponded" for similar reasons by another small drop for $N = 2$ hosts and 32 subflows per host.

Given that the number of subflows has an impact on performance, but only if there are too few (and not too many), we conclude that our SEMPER configuration with $K = 32$ sub-flows per host is a suitable candidate to implement a TE solution, given its good performance in terms of efficiency and fairness. Additional experiments are available in the Appendix.

In the following section, we evaluate its performance in a number of scenarios, comparing the results obtained vs. the state-full solution.

B. Comparison in static scenarios

To compare the performance of the two TE solutions, we consider different dumbbell topologies in terms of the number of 50 Mb/s paths between the two routers. For each configuration, we compute the total throughput and fairness (as defined before) of the stateless and the stateful solutions. It is worth remarking here that the stateless solution does not require any knowledge of the topology, while the stateful is based on generating as many sub-flows per host as disjoint paths are available. We repeat each experiment 10 times, and compute the efficiency and fairness of each configuration, providing the median across experiments. We perform our evaluation for the case of $N = 4$ hosts and an increasing number of paths between the routers, from 2 to 8 paths. The results are depicted in Fig. 8 for the total throughput (Fig. 8a) and fairness (Fig. 8b).

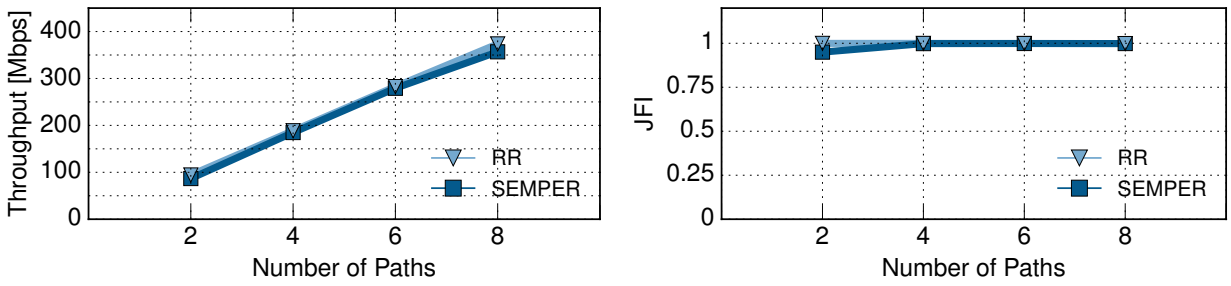
Concerning the *total throughput* figures, the results show that both TE solutions perform very efficiently, as in all cases the achieved results are very close to the total capacity between the links. Only for the case of 8 paths between the routers there are some differences between this maximum capacity (namely, 400 Mb/s) and the achieved capacity of both TE solutions, with the stateless version slightly underperforming the stateful. The reason for this small drop in performance for the proposed solution is, like in the previous section, the randomness of the assignment of subflows to paths, that in very few cases fails to occupy all available links.

For the case of *fairness*, the figure illustrates that both solutions perform very close to one (note that for the stateful solution, this is guaranteed by design). There is only one small drop in fairness for the case of two paths between hosts, which is caused when the total number of subflows is relatively way larger than the total number of paths (note that we had a similar behaviour in the previous section).

Following these results, we conclude that the performance of both TE solutions is almost optimal. Furthermore, given that our TE proposal does not require a prior knowledge of the topology (to configure the number of subflows per host to generate), nor the use of state in the routers, it results a more deployable solution to efficiently use all resources available.

C. Comparison in dynamic scenarios

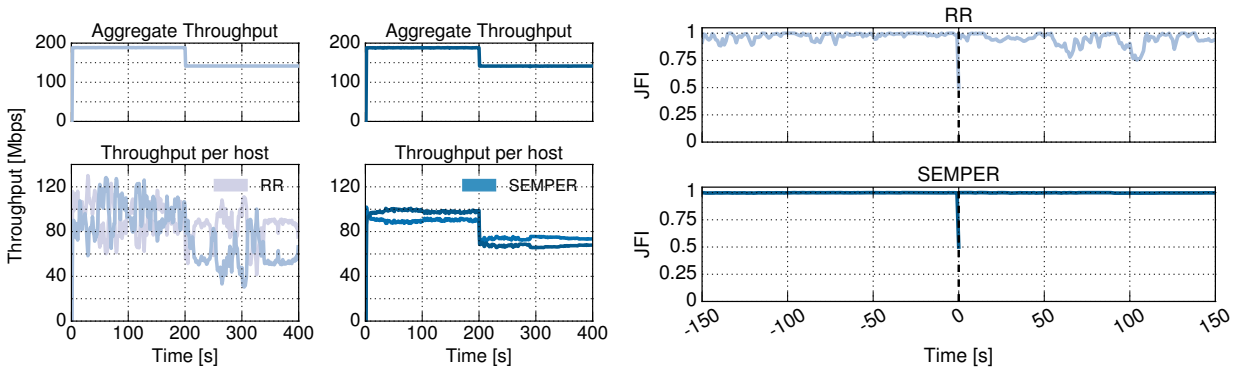
In the previous section, we consider static scenarios in terms of the available resources, i.e., a fixed number of paths between the hosts. As the results show, the designed TE solution makes an efficient and fair use of the available resources. Next, we



(a) Total throughput (Median)

(b) Jain Fairness Index.

Fig. 8: Performance evaluation in static scenarios, $N = 4$ hosts



(a) Instantaneous throughput over time.

(b) Instantaneous fairness over time.

Fig. 9: Performance evaluation in dynamic scenarios, $N = 2$ hosts

address an scenario in which the amount of available resources are reduced at a given point in time, i.e., a link is down, to assess the performance in these circumstances.

For ease of visualisation, now we consider the same dumbbell topology as before, but with only $N = 2$ hosts, and 4 paths between the routers. We run the experiments during T=400 s, and we remove one of the links at t=200 s. The resulting instantaneous throughput figures, averaged over 1 s, are depicted in Fig. 9, including the case of the total throughput (Fig. 9a, top) and the per-host throughput (Fig. 9a, bottom), for the two TE solutions considered.

According to the figure, both schemes immediately adapt to the new circumstances, with practically no loss of efficiency due to some "transient" conditions when adapting from a 4 paths to a 3 paths scenario.[4] We acknowledge that this result is somehow expected, as all involved transmission windows are operating at around the required values, and therefore the sudden unavailability of a link does not harm their operation –in terms of total throughput performance. However, when considering the per-host performance, it is worth remarking the differences between the two TE solutions: while SEMPER provides a very smooth operation over time, the stateful solution solution exhibits the "usual" variability of TCP throughput, which is caused by the fast reaction of one flow to losses experienced by the other flow. Indeed, it can be seen the "symmetry" across the average throughput per flow between the two lines. We argue that this is an additional benefit of the proposed TE algorithm, namely, better fairness properties over time.

To look further into the fairness properties of SEMPER, we compute the fairness figures over the same 1 s time windows, and represent them in Fig. 9b. As expected, the smooth behaviour from the use of our TE solution results in very stable behaviour of fairness over time, while the use of the stateful approach results in less predictable figures, due to the throughput variability described above.

VII. Conclusions

Current Enterprise networks are moving from monolithic infrastructures based on dedicated links and costly components to newer approaches that reduce costs, as well as improving resources utilization, reliability and resiliency. Costly dedicated links are replaced by multiple public/shared internet connections, increasing the availability of multiple paths. Traffic engineering solutions are very used in practise to exploit multiple paths, but they exhaust the available resource at the edge routers because of the computational time and memory they require. In this paper we proposed SEMPER, a

[4]We omit the results corresponding to the re-activation of the link, as recovery is practically immediate.

stateless solution for traffic engineering, that natively exploits path redundancy by efficiently matching MP-TCP subflows to paths, avoiding thus monitoring and fault avoidance techniques. Moreover, SEMPER moves the complexity to the end hosts by relying load balancing on the MP-TCP congestion control. Our results show that SEMPER is as effective as a stateful solution in terms of aggregating throughput and fairness, but with a practically negligible implementation cost.

REFERENCES

[1] O. Michel, E. Keller, "SDN in wide-area networks: A survey," *IEEE Fourth International Conference*, 2017.

[2] B.S. Davie, Y. Rekhter, "MPLS: technology and applications," *Morgan Kaufmann Publishers*, 2000.

[3] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "RFC 6824: TCP extensions for multipath operation with multiple addresses," *IETF*, 2013.

[4] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, X. Xiao, "RFC 3272: Overview and principles of Internet traffic engineering," *IETF*, 2002.

[5] D.O. Awduche, "MPLS and traffic engineering in IP networks," *IEEE communications Magazine*, 1999.

[6] M. Chiesa, G. Kindler, M. Schapira, "Traffic engineering with equal-cost-multipath: An algorithmic perspective," *IEEE/ACM Transactions on Networking*, 2017.

[7] E. Haleplidis, K. Pentikousis, S. Denazis, J.H. Salim, D. Meyer, O. Koufopavlou, "RFC 7426: Software-defined networking (SDN): Layers and architecture terminology," *IRTF*, 2015.

[8] C. Hopps, "RFC 2992: Analysis of an equal-cost multi-path algorithm," *IETF*, 2000.

[9] C. Raiciu, M. Handley, and D. Wischik, "RFC 6536: Coupled congestion control for multipath transport protocols," *IETF*, 2011.

[10] O. Bonaventure and S. Seo, "Multipath TCP deployments," *IETF Journal 12*, 2016.

[11] X. Wei, "MPTCP proxy mechanisms, draft-wei-mptcp-proxy-mechanism-00," *IETF draft*, 2014.

[12] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," *ACM CoNEXT*, 2012.

[13] M. Larsen and F. Gont, "RFC 6056: Recommendations for Transport-Protocol Port Randomization," *IETF*, 2011.

[14] J. Kristoff, "Ephemeral Source Port Selection Strategies," Available Online https://www.cymru.com/jtk/misc/ephemeralports.html

[15] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible Network Experiments using Container-Based Emulation," *CoNEXT*, 2012.

[16] R. Jain, D. Chiu, and W. R. Hawe. "A quantitative measure of fairness and discrimination for resource allocation in shared computer system," *Eastern Research Laboratory, Digital Equipment Corporation*, 1984.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, 2008.

[18] G. Combs, "TSharkDump and Analyze Network Traffic. Wireshark" ,2012.

APPENDIX

IMPLEMENTATION DETAILS

The implementation consists of two different software components, the *MP-TCP framework* that guides the execution of the experiments and the *data treatment tool* that extracts information from the packet dump files gathered during the experiments.

The *MP-TCP framework* (see Alg. 1) is an experiment manager that controls each component of the framework. The framework is composed by three main elements: *MP-TCP kernel*, *Mininet* and *SDN Controllers*. First of all, the framework sets up the *MP-TCP kernel* specifying the number of subflows to be generated per host, the congestion control algorithm and the path manager (*ndiffports* in our case). Then, the framework creates the topology through *Mininet's* python API, setting up hosts, servers and configuring links between edge routers with a certain bandwidth, queue size and delay. Finally, the framework starts all the component instantiated in *Mininet's* environment to be able to perform the experiment. The key component here are the *SDN Controllers* because they have the implementation of the different strategies to map subflows to paths (SEMPER and Round-Robin). More in detail, the edge routers have been deployed as a combination of an SDN controller and an OpenFlow virtual switch [17] to take advantage of the benefits of SDN.

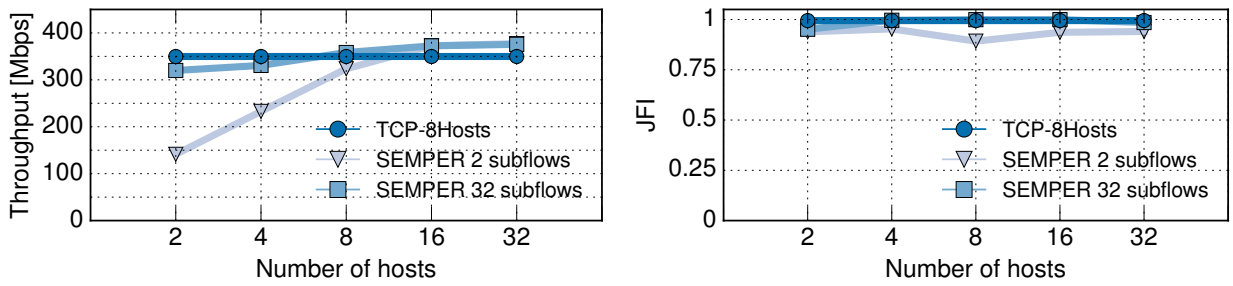
During the experiments, there are multiple *Tshark* instances

Algorithm 1 MP-TCP framework

for $iter = 0; iter < NUM\_ITERATIONS; iter++$ **do**
  $start\_flow\_balancer(edge\_routers)$
  `create_topology(NUM_HOSTS, NUM_LINKS, LINK_BW)`
  for $link = 0; link < NUM\_LINKS; link++$ **do**
    $new\ \ thread('tshark - ilink')$
  end for
  for $server = 0; link < NUM\_SERVERS; server++$ **do**
    $start\_iperf\_server(server)$
  end for
  for $host = 0; host < NUM\_HOSTS; host++$ **do**
    $start\_iperf\_client(host)$
  end for
  `# Wait experiment end`
  $stop\_servers()$
  $stop\_flow\_balancer(edge\_router)$
end for

Algorithm 2 Sampling algorithm: samples every second

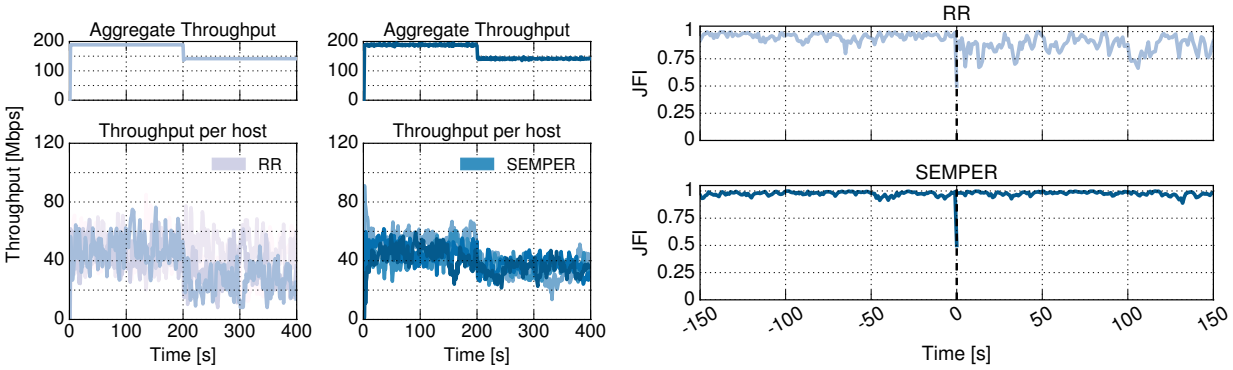
$data = sortByTimeStamp(subflow_data)$
$EXP\_DURATION = 400$
$sharedQueue = Queue$
for $check = 0; check < EXP\_DURATION; check++$ **do**
  $packet\_below = data.select(packet.time <= check)$
  $packet\_above = data.select(packet.time >= check)$
  if packet_below and packet_above NOT empty **then**
    $below\_absVal = abs(below.time - check)$
    $above\_absVal = abs(above.time - check)$
    $selected = min(below\_absVal, above\_absVal)$
  else if packet_below NOT empty **then**
    $selected = below$
  else if packet_above NOT empty **then**
    $selected = above$
  end if
  sharedQueue.write(selected)
end for



(a) Total throughput (Median)

(b) Jain Fairness Index.

Fig. 10: Validation of SEMPER configuration*: Experiments increasing the number of hosts*



(a) Instantaneous throughput over time.

(b) Instantaneous fairness over time.

Fig. 11: Performance evaluation in dynamic scenarios, $N = 4$ hosts

[18] gathering packets from all the available paths in the experiment and dumping all the information into files. The resulting files are the input of the second software component, the *data treatment tool*.

The *data treatment tool* extracts general information about each MP-TCP connection that took place during the experiment. The main idea is to extract representative information of each connection in order to have lightweight files that are used to compute concrete results (throughput or fairness). The process of extracting general information from the generated files could be challenging, because if the links bandwidth is high, huge packet traces files are generated.

The first step to optimize this process, is to define a sampling methodology to be able to handle these large dump files. The idea is to extract only representative packets of each MP-TCP connection in concrete points of the experiment, taking into account that every MP-TCP connection consists on more than one subflow (Algorithm.2). In other words, we select the packet that is close to a specific time stamp. For example, if the duration of the experiment is 5 seconds samples of the dump files should be collected every second, the selected packets will be those whose time stamp is the closest to {0s, 1s, 2s, 3s, 4s, 5s} for each subflow of each connection. By specifying the interval of time between samples, we can extract information with different levels of granularity.

Once the sampling process is defined, we speed up the extraction process by using a producer-consumer approach. With this definition, we improve the process by using multiple producers that extract the information exploiting parallelization, being the extracted information stored in a shared buffer. The consumers extract data from the shared buffer in order to generate a plain text file with the results.

Additional Results

In addition to the experiments performed in Sec. VI, we have performed further experiments regarding validation of SEMPER in Subsec. VI-A, the comparison of SEMPER with a stateful solution in a dynamic scenario (Subsec. VI-C).

Concerning the validation, we define a set of experiments where we vary the number of hosts (N) involved in the experiment , to assess how this parameter might affect to the resources usage. We also run the experiment using the dumbbell topology with 8 links of 50 Mb/s connecting the edge routers. In this case, we set 2 SEMPER configuration using $K = 2$ and $K = 32$ (SEMPER generating 2 and 32 subflows respectively), a set of hosts $N = \{2, 4, 8, 16, 32\}$ and maintaining as baseline the experiments with 8 single TCP flows using each flow a different path.

For each scenario (SEMPER configuration), we use the same methodology of 10 repetitions computing the median of

the results. Fig. 10 show both *total throughput* (Fig. 10a) and *fairness* (Fig. 10b) when increasing number of hosts involved in the experiment form $N = 2$ to $N = 32$.

Regarding the *total throughput*, the result show that SEMPER with $K = 2$ does not exploit all the available resources if $N < 8$, leading to a poor performance as we pointed out in the previous results. On the other hand, SEMPER with $K = 32$ performs well regardless of the number of hosts involved, outperforming the TCP baseline with $N = \{16, 32\}$. Considering *fairness*, all the configurations performs close to 1, which means fairness in resources sharing. As in the previous experiments, SEMPER with $K = 2$ suffers small drops caused by the cases where both subflows of a host were sent over the same path.

The number of hosts involved in the experiments has an impact on the performance, but it appears when there are too few hosts using SEMPER $K = 2$, similar to the results depicted in 7. With this information we can confirm that SEMEPER with $K = 32$ is a suitable candidate also when a larger number of hosts are involved.

Together with the experiments to compare SEMPER with a stateful solution in a dynamic scenario, we have performed a similar experiment but with two hosts ($N = 4$) and the results are characterized in in Fig. 11. As we identify in Subsec. VI-C ($N = 4$), both schemes immediately adapt to the new circumstances, in terms of throughput Fig. 11a, with practically no loss of efficiency. However, considering the per-host performance, SEMPER still operates smoothly over time compared with the stateful solution. Regarding fairness (Fig. 11b), SEMPER presents a smooth behaviour while in the stateful approach, the results are less predictive due to the fluctuation of the per-host throughput mentioned before.