Improving SD-WAN Resilience: From Vertical Handoff to WAN-Aware MPTCP

Yang Zhang, Jean Tourrilhes, Zhi-Li Zhang, *Fellow, IEEE*, and Puneet Sharma, *Fellow, IEEE*

***Abstract*—Demands for wide-area connectivity between enterprise site-edge networks and central office core networks/cloud data centers have grown rapidly. Various software defined wide area network (SD-WAN) solutions have been developed with the primary aim of improving WAN link utilization. However, mechanisms used by existing SD-WAN solutions fail to provide high reliability and performance required by today's edge to cloud applications. In this article, we present WAN-aware MPTCP which seamlessly aggregates multiple WAN links into a "big pipe" for better WAN resilience thus minimizing application performance degradation under WAN link failures. We leverage the congestion control of MPTCP to balance traffic across multiple WAN links. The key innovation is to combine LAN virtualization at end systems with WAN virtualization at SD-WAN gateways. Through evaluation in both emulated testbeds and real-world deployment, we demonstrate the performance gain of WAN-aware MPTCP in terms of resilience and throughput over existing SD-WAN solutions.**

***Index Terms*—Software defined networks, wide area networks, edge networks.**

I. Introduction

MANY modern enterprises are geographically dispersed across multiple sites over a wide area network (WAN). Typically, branch office site networks are connected to a central office core network or a core data center (private cloud) via "dedicated" WAN links provisioned by one or more service providers. For security and privacy, WAN gateways at each site route enterprise traffic over VPN tunnels connecting edge networks with core/cloud networks. However, WAN link failures happen more frequently than expected [50]. Even with a dedicated WAN, dealing with WAN failures is a key consideration in Google's B4&after systems [24], [26]. With increasing complexity in WAN (e.g., WAN managed by different ISPs; emerging 5G links adopted in WAN), such failures are likely to occur more frequently than before. More importantly, WAN failures have a significant impact on enterprises, and thus dealing with them is a major practical challenge.

Manuscript received June 2, 2020; revised October 24, 2020, December 24, 2020, and January 8, 2021; accepted January 13, 2021. Date of publication January 19, 2021; date of current version March 11, 2021. This work was supported in part by DoD ARO MURI under Award W911NF-12-1-0385; in part by DTRA under Grant HDTRA1-09-1-0050; and in part by NSF under Grant Grant CNS-1411636, Grant CNS-1618339, Grant CNS-1617729, Grant CNS 1814322, and Grant CNS183677. The associate editor coordinating the review of this article and approving it for publication was J. M. Kang. *(Corresponding author: Yang Zhang.)*

Yang Zhang and Zhi-Li Zhang are with the Department of Computer Science, University of Minnesota–Twin Cities, Minneapolis, MN 55455 USA (e-mail: yazhang@cs.umn.edu).

Jean Tourrilhes and Puneet Sharma are with the Networking & Mobility Lab, Hewlett Packard Labs, Palo Alto, CA 94304 USA.

Digital Object Identifier 10.1109/TNSM.2021.3052471

Compared to local area networks (LAN), WAN connectivity has become prohibitively expensive to meet the growing demands required by applications. This has led to a shift towards novel WAN solutions using software-defined networking (SDN) to better manage bandwidth intensive traffic traversing private WANs and increase the utilization of expensive WAN links. SD-WAN solutions [12] allow multiple WAN links to be logically combined for higher capacity. When WAN link failures are detected, traffic is re-distributed from failed links to other available links for resilience.

In this article, we present a novel scalable SD-WAN solution, *WAN-aware MPTCP* (*WaMPTCP*), which not only can seamlessly aggregate multiple (heterogeneous) WAN paths into a "big pipe", but is also capable of adapting to network failures or congestion to provide fast failure recovery to applications. The goal is to decrease the impact of WAN failures and bottlenecks on each client application by allowing it to take maximum advantage of the diversity of WAN paths provided by the SDN gateway. It is achieved by fusing WAN virtualization with an innovative *LAN virtualization* idea: we recognized that many OSs in end systems such as servers, desktops, laptops and mobile devices have built-in support for MPTCP; thus we utilize this fact by creating multiple virtual interfaces at an end system and enabling applications on the end system to create multiple MPTCP sub-flows, even though the physical interface of the end system may be attached to a single LAN that is connected to a SDN gateway with multiple WAN links. A combination of LAN virtualization and DHCP solves the challenge of making MPTCP on the end systems generate a number of subflows equal to the number of WAN paths at the gateway. LAN virtualization also solves the challenge of distributing and mapping those MPTCP subflows to WAN paths evenly and scalably: the gateway does not need to maintain flow-level state or perform MPTCP subflow association [57], but only rely on a few simple subnet routes. The gateway leverages MPTCP running on the end systems to achieve load balancing and congestion control across the MPTCP subflows and the WAN paths. By dividing a single application flow into multiple MPTCP subflows, WaMPTCP ensures that the application is not turned down when a WAN link is highly congested or fails, as such congestion or failure only affects a small portion of MPTCP subflows. WaMPTCP employs a fast recovery mechanism at gateways to distribute affected MPTCP subflows to other available WAN links to minimize the impact of WAN link failures on application performance. In addition to the support of performance critical applications running on hosts with MPTCP kernel modules, it also incorporates MPTCP proxies for legacy TCP initiated by

hosts with no MPTCP support as well as the default vertical handoff mechanism for UDP traffic. In summary, we make the following contributions:

- We motivate WaMPTCP which fuses *LAN virtualization* with *WAN virtualization* for resilience across WAN links (Section II).
- We present the detailed implementation of WaMPTCP (Section III), and propose two new metrics (Section IV) to better capture application performance under link failures.
- Through evaluation (Section V) in both emulated testbed and real-world deployment, we show the performance gain of WaMPTCP over existing SD-WAN solutions.

II. Motivation

Consider a simple enterprise branch site network consisting of two subnets (one wired and one wireless) as depicted in Figure 1. The subnets are connected to a campus core network with an SDN gateway connecting to a central office site or a corporate private cloud via three separate WAN links (each one to a different WAN provider). Client A has only one network interface connected to a WiFi subnet, whereas client B has two network interfaces, one connected to a WiFi subnet and the other to a wired subnet. Neither clients have awareness of multiple WAN links available at the SDN gateway. Traditional SD-WAN solutions also use the SDN gateways for load balancing or performing vertical handoff in event of WAN link failures or policy change [12]. They assume that the single LAN path to the gateway is not the bottleneck or main source of failure, and therefore the use of multiple WAN paths will improve end user experience. The key question this article aims to answer is: what novel mechanisms can we develop to utilize multiple available WAN links by taking advantage of software-define control of WAN links? We believe that a comprehensive solution with modifications to end-systems and SDN gateways (connecting to multiple WAN links) is required.

A. Fully Utilizing Multiple WAN Links

A common solution to fully utilize multiple WAN links is equal cost multiple path (ECMP). ECMP implementation uses TCP 5-tuple to assign packets to paths. ECMP is flow aware and ensures that packets that belong to the same TCP session use the same path over the Internet and are never spread across multiple paths. The limitations of any flow aware load balancing is that a single flow cannot aggregate the bandwidth of all paths (Section V-A) and the traffic distribution can be unbalanced and unfair when flows are not uniform, causing artificial congestion [43].

Per-packet load balancing is offered in certain SD-WAN solutions [12]. TCP packets of each flow are buffered and reordered at the merge point of the paths. This is expensive in terms of memory and computation, and tricky to get right in the presence of packet losses and varying delays. Since standard TCP congestion control algorithms assume that packets follow a single network path, they may get "confused" by different RTTs on diverse paths [14]. Packets traversing multiple paths would arrive at the destination *out-of-order*, triggering

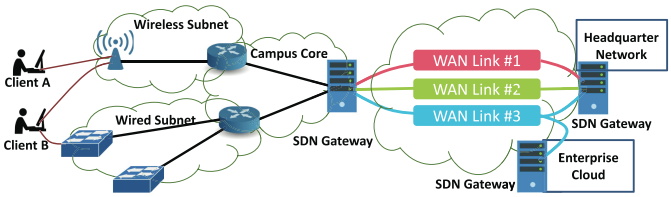


Fig. 1. Motivating scenarios.

duplicate acknowledgments, which is interpreted by the source as packet losses, thereby reducing the sending rate.

To overcome those issues, MPTCP [8], [36] is developed to enable a single TCP session running on a host with multiple interfaces to exploit multiple network paths. MPTCP manages multiple subflows (each operating as a separate TCP connection with its own congestion control), and dynamically assigns data to each TCP subflow based on available capacity, thus load-balancing among multiple paths. If one path fails (or is highly congested), MPTCP automatically routes data away from the path, thereby alleviating the impact of network failures/congestion on application performance. In addition, MPTCP shares various network paths fairly with regular TCP sessions and other MPTCP sessions [19]. The benefits of using MPTCP for load balancing, wireless handoff and coping with network failures have been widely studied [20], [37], [38], [46], [54]. In contrast, applying MPTCP in SD-WAN scenario has limited investigation.

Then the question is where we should deploy MPTCP protocol stacks. A natural way of doing so is to deploy MPTCP proxies co-located at (or implemented as a module within) SDN gateways [5], [11], [22]. This requires a pair of MPTCP proxies, e.g., one at the branch site and the other at the central office/private cloud site: a TCP connection from a client machine to a remote server is split by the MPTCP proxy at the local SDN gateway to generate multiple MPTCP subflows, and the gateway routes them across different WAN links to the remote SDN gateway on the other side of the WAN. These MPTCP subflows are then merged by the remote MPTCP proxy before routed to the server. Clearly, this approach incurs overhead, because it requires the gateways to keep track of every single TCP connection over the enterprise network to the WAN. Similarly, MPTCP overlay network [31] serves the same purpose. Instead of splitting TCP flows, it encapsulates TCP flows with MPTCP header for fully utilizing multiple WAN paths. However, the congestion control operated by the TCP connection of an application and the MPTCP congestion control at the gateways can interact negatively, creating performance penalties for high-speed WAN links (Section V-C). Since end systems (for those with MPTCP support in OS) have already deployed MPTCP stacks, this triggers us to propose utilizing the MPTCP stacks on end systems rather than duplicating them at both end systems and gateways to avoid the negative impact of MPTCP proxy or overlay network.

B. End System MPTCP to the Rescue?

Is deploying MPTCP at end systems alone sufficient? Again considering the network in Figure 1, we assume that the OSs

on both machines support MPTCP. Client A cannot avail itself of the three available WAN links using MPTCP because it is only aware of one directly connected (physical) network interface. Thus, traffic from an application running on client A can only traverse one of the three available WAN links, failure of which will affect application performance. In the case of client B, it can deploy MPTCP to generate two subflows, one over the WiFi subnet and the other one over the wired subnet. However, the two subflows will converge at the SDN gateway, and both may be routed along the same WAN link to their common destination. In such case, if there is a heavy congestion or failure on this path, both subflows will experience it, MPTCP cannot overcome the path condition using multipathing, and the application will suffer. The server could have multiple network interfaces, which would enable the client to generate multiple subflows with different destination addresses. However, the same issues persist, as the number of subflow would not always match the number of WAN links, and the gateway would not know how to route them. Ideally, the number of subflows should be the cross product of the number of WAN links and number of server interfaces to maximise network diversity. In general, we note that there can be a mismatch between the number of physical interfaces at an end system and the number of available WAN links at a SDN gateway. In addition, the gateway must distribute evenly the subflows of an MPTCP session across the WAN links to avoid any two of them being mapped to the same WAN link.[1] Making end systems aware of the availability of WAN links at the gateway, enabling them to generate appropriate number of MPTCP subflows, and routing these subflows across different WAN links in a *scalable* manner are challenging.

The above scenarios raise the following questions: i) Is it possible to enable an MPTCP-compatible end system with only one physical network interface to fully exploit *multiple* WAN links? ii) Given an end system running MPTCP with multiple subflows, is there a *scalable* way to ensure that the SDN gateway always routes subflows of an MPTCP session across different WAN links? An astute reader may suggest that one can use the "ndiffports" path manager option [8] in MPTCP to create multiple subflows across the same pair of IP addresses. However, source ports are randomly generated, so it is infeasible for a gateway to provision routing rules based on the source ports assigned during run-time for each single MPTCP subflow. Moreover, associating MPTCP subflows belonging to the same session is nontrivial and requires maintaining states at gateways [57]. In contrast, virtual interfaces assigned with different subnets can be easily provisioned, because routing rules can be statically provisioned based on pre-defined subnets. Additionally, no per-flow states are required for routing based on subnets at the gateway.

[1]If the gateway maps flows randomly to WAN links, or using traditional load balancing solutions such as hashing, there is a fairly high probability that two subflows of a session are mapped to the same WAN path, and no subflow is mapped to another WAN path (Section V-B).

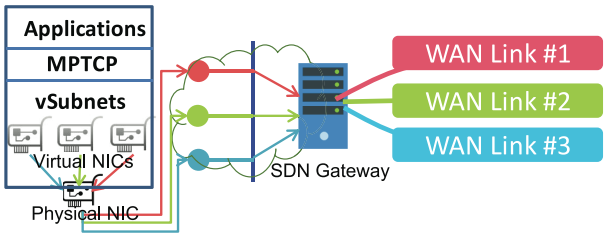


Fig. 2. Virtualizing NIC for MPTCP stack to generate multiple subflows and for SDN gateway to forward subflows in a stateless manner.

III. WaMPTCP Overview and Implementation

WaMPTCP is designed specifically to address the requirements in SD-WAN for connecting branch office enterprise networks to a central office and/or private cloud over the Internet with multiple WAN links provisioned by several WAN service providers. The goals of WaMPTCP are multi-fold:

- Mitigating the impact of failure/congestion on applications;
- Aggregating WAN links capacity to deliver higher bandwidth;
- Supporting diverse types of traffic (TCP, UDP, MPTCP);
- Maintaining fewer states at gateways for scalability and reliability, and minimizing software changes at end systems.

WaMPTCP is a novel scalable solution which is schematically depicted in Figure 2. It combines a low-overhead mechanism at the end system side to create multiple *virtual* network interfaces, and a *scalable* mechanism at the SDN gateway side that routes flows generated from the same application across different WAN links, detects WAN link failures, and adaptively reroutes flows from failed WAN links to healthy ones with minimized performance impact.

The virtual network interfaces are *WAN-aware*, as each corresponds to one WAN link, and each is assigned with a subnet address mapped to a WAN link at the SDN gateway (and remapped to another available link if the current WAN link fails). Hence the SDN gateway can route flows to the corresponding WAN link via a simple source-destination address-based flow lookup. The benefits of doing so is to fully utilize multiple available WAN links without stateful operations on the gateway. The details regarding how the virtual subnet addresses are created and advertised to end systems, and how the whole system is implemented are presented next.

A. WAN-Aware MPTCP

As is well known, an MPTCP proxy has the limitation of breaking TCP end-to-end semantics [28], and its scalability is problematic, as it needs to keep track of every pair of flow connections, and maps them back to back. Moreover, with MPTCP proxy implementations based on HPSockd [1] and Dante [2], respectively, we observed performance penalties for high speed WAN links (Section V-A). Due to those limitations, we invented WaMPTCP, a new mechanism to address the lack of WAN link awareness at end systems.

We want an end system to initiate the same number of MPTCP subflows as the number of available WAN links at an

SDN gateway (or multiple of that number if the server is multi-homed), despite the end system only having a single physical interface. Further, the gateway should map those MPTCP subflows to WAN links in a scalable manner. Thus, WaMPTCP is composed of two major components: an IP subnet provisioning in LAN and a flow routing on the gateway.

IP Subnet Provisioning: Instead of allocating a single IP subnet, we allocate multiple subnets in LAN, each one associated with one WAN link. Such provisioning uses more subnets, and thus more IP address space. However, most campus networks nowadays use private IPv4 and IPv6 address space for clients, which means that IP address space is relatively plentiful. End systems are informed of multiple IP subnets by DHCP protocol. The way of handling IPv4 is different from handling IPv6 networks. For IPv4, we modified DHCPv4 protocol [16] by inserting a DHCP option into DHCP response for informing end systems the number of available subnets (WAN links). A modified DHCP client on the end system uses this DHCP option to create multiple virtual network interfaces via IP aliases, and sends a DHCPv4 request over each virtual interface to get each of them configured with an IP address. If the number of WAN links changes (not frequent in enterprise networks), DHCP TTL is used to refresh the number of subnets on end systems.

IPv6 makes the above process simpler. Since assigning multiple IP addresses per interface is a part of the IPv6 standard, there is no modification required at end systems. The DHCPv6 server is configured to assign multiple IP addresses to the end systems, one per each subnet, and DHCPv6 client automatically configures those IP addresses on the virtual network interfaces. ISC DHCP server does not support multiple subnets per response, and therefore we use the dhcpy6d server with an appropriate configuration [3].

Gateway Routing: Routing on the SDN gateway forwards packets originated in a subnet to the associated WAN link. This relies on source-specific routing which is supported by most OSs. Only subnet-based static rules are required for routing instead of generating a rule for each flow. Return packets coming from WAN links are routed based on destination subnet (classical routing). Our routing technique is implemented to support both VxLAN tunneling to a remote gateway and direct routing to the Internet. The number of rules in both directions only depends on the number of subnets, independently of the number of clients. If the subnets are optimally allocated, subnet routing requires only a number of rules equal to the number of WAN links. This very low number of rules offers great scalability and avoids control plane churn.

Here is an example on how IP subnet provisioning and gateway routing work. When a host enters a LAN, it requests an IP address from a DHCP server. Multiple IP addresses are allocated to the host, and correspondingly, virtual interfaces are created at the host, one per WAN link. After that, the MPTCP stack on the host will "see" multiple (virtual) interfaces, and natively generates corresponding number of MPTCP subflows for delivering application traffic. Generated MPTCP subflows are then forwarded to WAN links respectively based on static (subnet-based) rules pre-defined on the gateway. When a WAN link fails, MPTCP congestion control algorithm dynamically adjusts the sending rate of each subflow at host, and thus the gateway gets relieved from stateful operations.

B. MPTCP Fast Recovery

During a WAN link failure, any MPTCP subflow using the link is affected: packets cannot reach the receiver, and the TCP congestion control algorithm triggers retransmission and eventually timeouts, causing the subflow to get stalled. When a subflow is stalled, it does not send packets to probe the path and can suffer from an outage much longer than the actual L3 outage (Section V-C). In certain circumstance, the subflow cannot resume at all, due to the TCP exponential backoff algorithm. We propose MPTCP Fast Recovery (FR) to address the issue of stalled MPTCP subflows on failed path, and it can be optionally applied to both WaMPTCP and MPTCP proxy.

One obvious solution would be to crank up retransmission frequency of all the MPTCP subflows, so that the failed subflow does not wait too long before probing the path again. However, this has a number of downsides. First, this is not a standard configuration of an MPTCP stack, so we would need a new mechanism for the gateway to communicate with end systems to use more aggressive retransmission. Second, if subflows probe failed path too often, probing packets consume extra resources on the local network and on the gateway, possibly reducing the performance of subflows using healthy paths. Third, probing packets carry MPTCP session data, and this data must first be sent over the failed subflow, and wait for the failure timer, before being resent on the healthy subflow. However, this can add latency to the data (over 300ms in Section V-D).

Our solution is to allow the SDN gateway to probe for failed subflows. When the gateway detects that a path is down, it reroutes MPTCP subflows on that path to an alternative healthy path. When those MPTCP subflows are on a healthy path, they can resume progress and it prevents them to stall. At this point, the redistributed subflow interferes with what was already on that path, but it does not break fairness in the shared WAN link due to the design principle of MPTCP congestion control algorithms [8]. Once the gateway detects that the failed path is recovered, it reroutes all the affected MPTCP subflows back to their original path. Without stalled subflows, the recovery time of MPTCP sessions can be improved (Figure 10).

The rerouting of all affected subflows on the gateway can be done with a single rule matching the destination port, by using an OpenFlow failover or select group [49], and no per-flow state needs to be tracked. The gateway must monitor the WAN links and update rules accordingly. These are the same requirements as most other L3 handoff mechanisms and are fairly standard [34]. Only one rule is needed per WAN link for FR, making it highly scalable.

C. Handling Diverse Traffic Types

An additional challenge was building a practical system on a gateway for properly handling diverse traffic types. We rely on Netfilter (serving as protocol classifier) [9] and OpenvSwitch (OVS, serving as path scheduler) [34]. The processing pipelines are illustrated in Figure 3. Netfilter marks

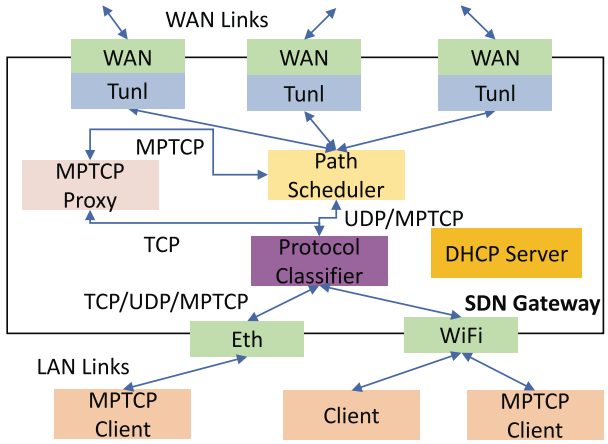


Fig. 3. Handling diverse traffic types in an SDN gateway.

packet types (TCP, MPTCP, and UDP), and then they are processed differently in the SDN gateway. TCP sessions generated by hosts without an MPTCP stack are detoured to MPTCP proxy, and encapsulated into MPTCP sessions for fully utilizing WAN links; MPTCP sessions are directly processed by OVS, and forwarded to WAN links based on source-specific routing; UDP sessions are forwarded to WAN links based on destination routing. In OVS, there are two sets of OpenFlow rules, one for directly mapping MPTCP flows to uplinks, and the other one mapping other traffic to a group selecting an uplink. The OVS failover or select group [34] is configured for switching flows to healthy links during failure. Bidirectional Forwarding Detection (BFD) integrated in OVS is used for detecting WAN link failures. Overall, WaMPTCP only requires one-time configuration in installing (virtual) subnets in DHCP servers. At end-systems, no changes are needed for IPv6 hosts, while DHCP client in IPv4 host only needs to execute a script for creating virtual interfaces once during IP provisioning.

MPTCP is designed to make sure that MPTCP sessions fairly share the network path they are using with other TCP sessions and MPTCP sessions on that path [19]. Therefore no extra mechanism is needed at the gateway to enforce fairness. WAN links are the most likely bottlenecks, where contention will happen. In the absence of failure, for every MPTCP session, exactly one MPTCP sublow is mapped on each WAN link, making achieving fairness easier. With failure and FR, multiple flows of the same session may be mapped to the same WAN link, various studies show that MPTCP can also handle this more complicated case fairly in practice [18].

D. Interaction With Middleboxes

The actual paths from client to the WAN are not always as simple as direct links shown in Figure 2. Middleboxes that transform, inspect, filter, or manipulate traffic are widely deployed in the network for the purpose of improving performance and security. We analyzed whether common middelboxes such as NAT or firewall would affect the functionality of WaMPTCP.

In most cases, middlebox functionality is placed before tunneling, so routing and failover would happen on the external side of middleboxes. Thus, the gateway would receive packets modified by middleboxes. Since the gateway relies on source-specific routing, middleboxes without modifying source IP do not affect the functionality of WaMPTCP. In the case that source IP is modified, the gateway still distributes (NATed) IPs in different subnets to different WAN links, and thus the functionality of WaMPTCP does not get affected. Even though middleboxes may convert source IPs from different subnets to the same outgoing IP, it falls back to the scenario with no WaMPTCP, because the virtualized IP addresses on the same host are unified back.

In the cases that middlebox function is placed after tunneling, WaMPTCP functionality are not affected, since traffic goes through WaMPTCP first. Middleboxes only see the external tunnel with encapsulated headers. Unless middleboxes need to perform statistics based on host (per-host metrics), the functionality of middleboxes are not affected as well.

IV. Performance Metrics for Vertical Handoff

MPTCP implements vertical handoff at client side, while a traditional SD-WAN handoff happens at the gateway. Since they use different mechanisms, we propose two new metrics for evaluating different SD-WAN techniques on application and network session performance. The metrics cannot only capture SD-WAN performance, but also drive the design of WaMPTCP. We first present the problem of existing Layer 3 metrics for measuring vertical handoff, and then describe two new metrics taking L4 information into account.

A. Layer 3 Metrics

Vertical handoff is well characterized at layer 2 (link) and layer 3 (rerouting) [45], [47]. There are two types of handoff events: 1) proactive handoff, usually the result of a policy decision or a link being activated; 2) reactive handoff, usually the result of an Internet path outage. Up to layer 3, a handoff is commonly decomposed into four phases:

- *Detect:* A mechanism must detect that the Internet path has failed or a policy has changed;
- *Compute:* An entity decides what routing changes to make;
- *Notify:* The entity that computed the new routes must notify the entities that perform rerouting;
- *Switchover:* A mechanism must change the routing of impacted traffic to the alternate Internet path.

Consequently, the time of performing a handoff is the sum of time spent over four phases, and it applies to both proactive and reactive handoff. Prior studies have defined Layer 3 metrics for measuring handoff performance under various network settings, e.g., [21], [33], [53]. However, those common Layer 3 metrics do not show the true cost of vertical handoff, because TCP congestion control is also impacted. Moreover, those Layer 3 metrics cannot apply to MPTCP, because MPTCP handoff does not happen at Layer 3.

B. Metrics Taking Layer 4 Into Account

Since rerouting events adversely affect TCP performance [39], several TCP optimization schemes

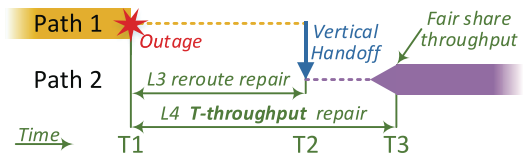


Fig. 4. T-throughput Metric.

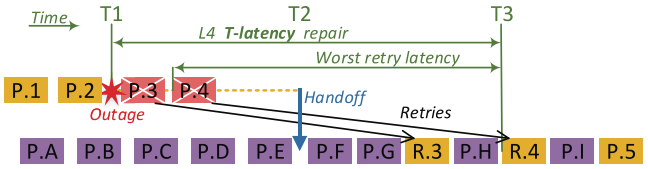


Fig. 5. T-latency Metric.

have also been proposed, e.g., [21], [29]. These studies examine the impact of link failures on TCP performance by observing congestion window reduction, instant TCP throughput degradation, or packet losses. These metrics are too specific and are not directly related to the application experience. Therefore, we propose two new metrics to evaluate the impact of TCP connection handoff. We require the new metrics to have the following features: 1) a direct measurement of the handoff effect; 2) applicable of measuring both reactive and proactive handoff, due to path outages or policy changes; 3) related directly to the application performance.

The first metric is *T-throughput*: time for throughput repair. It measures the time elapsed from the start of the outage or policy change, i.e., as soon as the link becomes undesirable, to when the throughput of an affected TCP connection is fully restored. As shown in Figure 4, there are two WAN links. Before outage, a flow is traversing over WAN link 1. The link fails at time T1, and then it takes time for the gateway to detect the outage. After that, the handoff mechanism at the gateway migrates the flow from link 1 to link 2 at T2. After the flow is placed on link 2, it takes additional time (T3 - T2) for throughput to fully recover.

As the conditions on the old path and the new path may be different, the expected throughput on the new path is likely different from that on the old path. Therefore, we consider that the TCP throughput has been fully restored when it has obtained fair share of the new path bandwidth. In other words, fair share throughput indicates that the TCP slow start phase has ended and the throughput becomes relatively stable. This metric better measures the impact of handoff on an application that is *bandwidth-bound* (e.g., a file download), as it quantifies how long an application suffers from performance degradation before it gets fully recovered. This metric is a function of how efficient a TCP congestion control algorithm operates after the handoff.

The second metric is *T-latency*: time for latency repair. It measures the time elapsed from the start of the outage or policy change, to when all packets lost during handoff are retransmitted on a healthy path. Figure 5 show an outage happening at T1. Before T1, packets P.1 and P.2 are successfully delivered over Path 1. After T1, packets P.3 and P.4 fail to be delivered. At time T2, handoff happens and those failed packets are queued to be resent over a healthy path. At time T3, the last failed packet R.4 gets delivered, and thus we measure T3 - T1 as the time for latency repair. This metric provides a better measure of impact of handoff on an application that is *latency-bound*, e.g., interactive gaming over TCP. It also better describes the impact of handoff on any application suffering from stragglers (late packets). In particular, the MPTCP receiver needs to reserialize the incoming data prior to delivering it to the application, so any stragglers will block the receiver and increase receive buffer consumption (receive buffer blocking [27]). This metric is a function on how efficient the TCP retransmission mechanism operates after the handoff, and how aggressive the MPTCP scheduler is in switching MPTCP subflow.

We verified that these metrics are able to properly explain the performance of applications during handoff (Section V-F). A key benefit of these two metrics lies in that they allow us to evaluate how TCP configurations and network conditions affect the network failure recovery time (Section V-D). In particular, they enable us to design better failover and recovery mechanisms under MPTCP and evaluate them fairly against existing (single-path) TCP mechanisms.

V. Evaluation of WaMPTCP

We evaluated WaMPTCP in both controlled testbeds and real-world deployment. This allowed us to evaluate performance across a wide range of workloads and network conditions.

We compared the following mechanisms in the experiments:

- *Tunnel Handoff* is a baseline. It uses OVS version 2.4.2 [34] to route flows over VxLAN tunnels. Unless specified, application data is carried in plain TCP, while some tests do use plain MPTCP without WAN awareness;
- *MPTCP Proxy* is a transparent proxy solution. By default, MPTCP Proxy in our experiments is our proxy implementation based on Dante Socks proxy version 1.4.1 [2]. For performance comparison, we also used our modified version of HPsockd v0.17 [1];
- *MPTCP Tunnel* [31] is an overlay solution which tunnels TCP over MPTCP. All TCP flows are encapsulated in a single MPTCP flow between gateways. We used the implementation available online [4];
- *WaMPTCP* is implemented in end systems and SDN gateways (Section III-A). End systems are provided with one IP address per WAN link. *Fullmesh*, the default MPTCP path manager, is used, and SDN gateways route MPTCP subflows based on source subnets;
- *WaMPTCP+FR* is WaMPTCP equipped with FR (Section III-B). When a gateway detects a path failure, it routes infected MPTCP subflows away from the failed path, and routes them back when the path is recovered.

The testbed for the Aggregation and Fairness experiments uses six Debian version 9 servers and its topology is broadly similar to Figure 1. There are 2 clients connected via 10 Gb/s links to the local gateway. The local gateway is connected to a remote gateway via five 1 Gbps WAN links. Two servers are connected to the remote gateway via 10 Gb/s links. The

TABLE I
THROUGHPUT AGGREGATION COMPARISON

Mb/s	Direct Routing			VxLAN tunnels		
	IPv4	IPv6	10 IPv4	IPv4	IPv6	10 IPv4
Plain TCP	941	928	944	908	893	4550
Plain MPTCP	928	915	931	893	888	4480
MPTCP Tunnel	120	N/A	119	114	N/A	112
HPsock Proxy	740	N/A	1050	752	N/A	876
Dante Proxy	4285	3925	4550	3233	2763	3610
WaMPTCP	4606	4531	4640	4433	4221	4490

local gateway implements source subnet routing, whereas the remote gateway use standard destination subnet routing. When VxLAN tunnels are used, there is one on each of the 5 link. The local and remote gateways are tunnel endpoints and they are implemented using OVS. MPTCP version 0.93 is used. MPTCP CRC are disabled. The scheduler is fullmesh and Cubic is the default congestion control. Packet MTU is 1500B on the links and 1450B on the VxLAN tunnel, and all other network parameters and drivers are using out-of-the-box defaults.

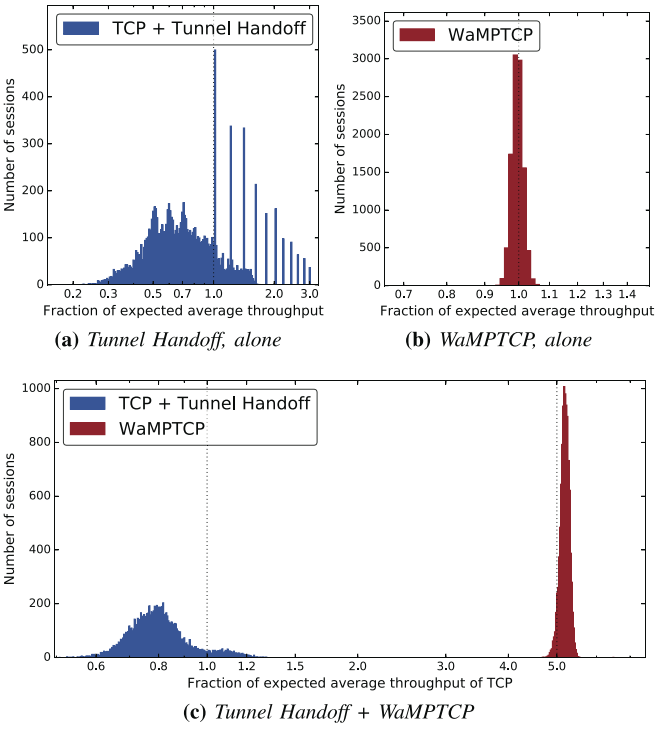
The testbed for handoff and metric experiments is similar. It uses six Ubuntu 16.04 servers. Two links between the gateways are 100 Mb/s, while the links to clients and servers are 1 Gb/s. All other configuration parameters are the same.

A. Evaluation of Bandwidth Aggregation

Table I shows the overall throughput of the 6 scenarios with a single IPv4 connection, a single IPv6 connection, or 10 IPv4 connections. Direct routing (i.e., no VxLan tunnels) can only use the default path for plain TCP connections due to routing constraints, while VxLAN tunnels can use per-flow load balancing to aggregate link capacity. Since there is only one physical NIC on each host, there is only one subflow generated for each connection when using plain MPTCP, so MPTCP does not show any improvements over using plain TCP. Even though plain TCP and plain MPTCP achieved similar throughput as WaMPTCP when there are 10 flows, plain TCP or plain MPTCP may not be overall fair between the 10 flows (Section V-B). MPTCP tunnel is far from saturating the links even when there are 10 flows, and further experiments indicated that this is a control loop issue (Section V-C). Dante proxy is a much better implementation than HPSock proxy, and therefore shows much better performance. Dante proxy offers lower throughput than WaMPTCP due to the smaller MTU and the higher CPU load. WaMPTCP performs the best because it can saturate all available bandwidth in all confitions tested.

B. Evaluation of Multipath Fairness

Previous studies have shown that TCP and MPTCP sessions share fairly the network path they are using with other TCP and MPTCP sessions on the same path [18]. On the other hand, the SDN gateway provides multiple paths, so it also needs to make sure the multiple flows are placed on the various paths fairly. When direct routing is used, the gateway cannot provide multipath for TCP flows, so we consider only the case where VxLAN tunnels are used. For TCP flows, Tunnel Handoff uses



(a) *Tunnel Handoff, alone* **(b)** *WaMPTCP, alone*

(c) *Tunnel Handoff + WaMPTCP*

Fig. 6. Multipath fairness of MPTCP and Tunnel-Handoff.

a flow based hash to map TCP flows to WAN path, which is implemented using a OpenFlow select group in OVS [34]. For MPTCP sessions, WaMPTCP maps one subflow of each MPTCP session on each WAN path.

Our fairness experiments involve generating simultaneously a random number of TCP or MPTCP parallel sessions contending for the 5 VxLAN tunnel over 5 WAN paths, each offering 1Gb/s. To normalise for the different number of sessions in each experiment, we compare the throughput of each session to the expected throughput, 4.5 Gb/s divided by the number of sessions (Section V-A). Each experiment has a random number of TCP or MPTCP session, between 5 and 15, lasts 60s and the experiments are repeated to collect over 10000 sessions. Figure 6(a) shows that flow hashing is spatially unfair due to hash collisions. Flow hashing cannot guarantee that exactly the same number of TCP flows are mapped on each WAN path. Further, for low number of flows, the fact that flows cannot be split prevent reaching a fair solution, for example there cannot be a fair distribution of 6 flows on 5 paths. Many flows get half of the expected average throughput, a very few flows can get up to 3 times of the expected average throughput. There are experiments where no flow is mapped to one of the WAN paths, and as a result the average aggregate throughput of all experiments is only 3776 Mb/s (84%). Those are known issues of ECMP style load balancing, and ways to overcome those are usually complex, and can usually only apply to a few long lived flows [15], and are never perfect as flows cannot be split. Figure 6(b) shows that WaMPTCP is very fair, with very few exceptions the throughput of MPTCP sessions falls within $[-6\%; +6\%]$ of the expected. Average aggregate throughput is 4478 Mb/s. MPTCP increases the overall number of subflows in the system by a factor five, so increases

TABLE II
VERTICAL HANDOFF PERFORMANCE OF DIFFERENT MECHANISMS ON THE TESTBED

seconds	**T-throughput-failure**		**T-latency-failure**		**T-throughput-recovery**		**T-latency-recovery**
BFD timer	*1s*	*100ms*	*1s*	*100ms*	*1s*	*100ms*	
Tunnel Handoff	3.77	0.63	3.52	0.56	1.33	0.42	0
MPTCP Tunnel	0		0.34		6.57		0
MPTCP Proxy	0		0.37		6.41		0
WaMPTCP	0		0.32		6.86		0
WaMPTCP+FR	0		0.32		1.28	0.44	0

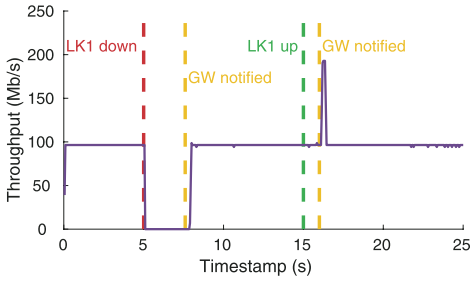


Fig. 7. One TCP Flow over Tunnel Handoff mechanism.

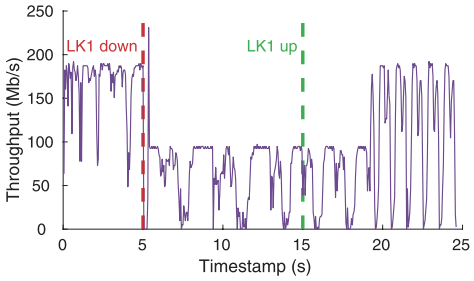


Fig. 8. One TCP Flow over MPTCP Tunnel mechanism.

the contention on each WAN path, however each MPTCP session has exactly one subflow on each WAN path so multipath fairness is easy to achieve, without any impact on aggregate throughput.

In Figure 6(c), simultaneous TCP and MPTCP traffic is generated, each managed by Tunnel-Handoff and WaMPTCP. The number of TCP and MPTCP sessions is always the same in each experiment and is randomly chosen between 5 and 15. MPTCP sessions contend on 5 paths, whereas TCP sessions contend 1 path, therefore we expect the aggregate throughput of TCP sessions to be $1/6^{\text{th}}$ of 4.5 Gb/s, and we normalise all sessions to 750 Mb/s divided by the number of sessions in the experiment. The average aggregate throughput of TCP sessions is 620 Mb/s (83% - same as alone), and the spread is lower than when alone. Thus, the presence of MPTCP traffic is not hurting the TCP sessions and fairness is preserved. MPTCP traffic average aggregate throughput is slightly higher than expected, 3874 Mb/s, because it can take advantage of cases where Tunnel Handoff maps too few sessions to a path.

C. Analysis of Handoff Trace

We next zoom in different mechanisms to observe their handoff behaviors. We initiated one TCP flow, and broke the link it is using (labelled as "LK1" in the figure) after 5 seconds. After keeping the link down for 10 seconds, we recovered it. Breaking down a link is performed by installing Netfilter rule [9], which is almost instantaneous. We decompose an outage into two phases: failure (which typically harms throughput) and recovery (which typically benefits throughput). We did not evaluate policy changes, since its effect is identical to a recovery from an outage. MPTCP proxy behaved almost identically to WaMPTCP, so it was not shown.

Figure 7 shows that tunnel handoff does not aggregate available bandwidth on the two WAN links together, and it experiences more than 3 seconds down-time during failure. Approximately 3 seconds after the link is blocked, BFD considered the path as failed (3 BFD intervals), and tunnel handoff

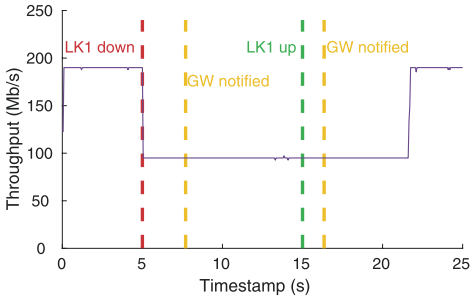


Fig. 9. One MPTCP Session over WaMPTCP mechanism.

rerouted the affected flow to the healthy link, and it resumed progress. After the failed WAN link is unblocked, BFD eventually discovers that the path is healthy, and notifies OVS to reroute the flow to its original path. In our experiment, there is throughput spikes shortly after recovery. This is because prior to handoff, the flow had filled the send buffers of the NIC and the tunnel endpoint on the gateway before the WAN link (which is the bottleneck). After handoff, the flow moved to an empty link, and could therefore transmit immediately, while the other link was still draining its send buffers. This caused out-of-order packets [39]. Since the Linux TCP/IP stack has optimization to overcome out-of-order packets [21], it did not hurt throughput.

MPTCP Tunnel in Figure 8 does not exhibit stable throughput even without any background traffic. It is mostly caused by the interaction between the outer TCP control loop (end-to-end) and inner TCP control loop (tunnels established between gateways). This is a well-known issue of any TCP in TCP encapsulation [10].

Since MPTCP natively supports resilience by maintaining multiple subflows, there is no down-time observed during failure in Figure 9 and Figure 10. Conceptually, suspension of one subflow does not affect the other one.

Figure 9 shows that when FR is not used, the subflow get stalled (Section III-B). When the failed link is recovered, it

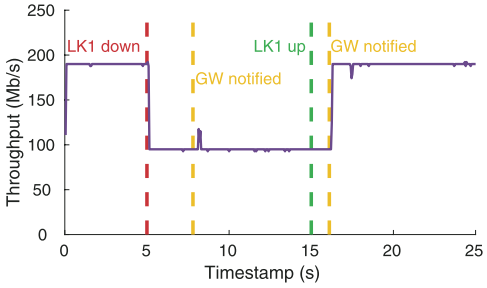


Fig. 10. One MPTCP Session over over WaMPTCP + FR mechanism.

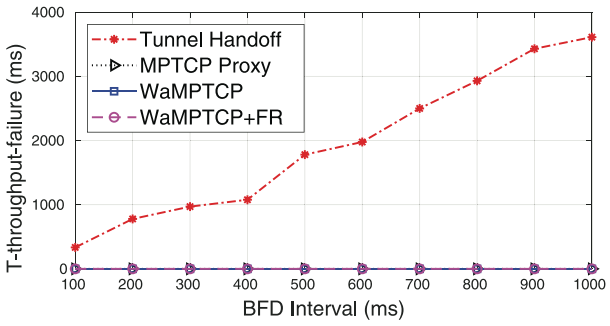


Fig. 11. BFD impact over T-throughput-failure.

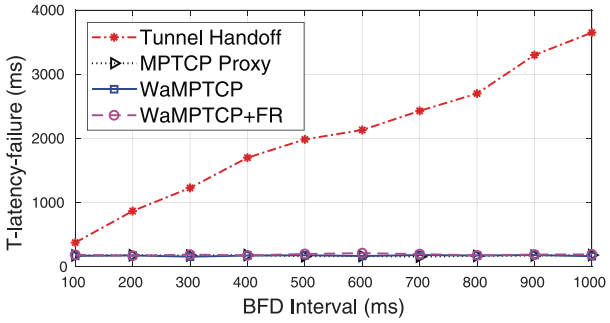


Fig. 12. BFD impact over T-latency-failure.

takes less than 1 second for the gateway to get notified, but throughput needs further 5 seconds to recover fully. Further experimentation shows that this L4 throughput repair time depends on the length of the outage and is due to TCP RTO backoff (Figure 14). In contrast, Figure 10 shows that FR prevented the subflow from getting stalled, and that it recovered the bandwidth of the second path very quickly after being moved to the recovered path.

D. Evaluation Using Handoff Metrics

In order to quantitatively study handoff impact on transport layer, we need to use the two new metrics, *T-throughput* and *T-latency*, proposed in Section IV-B. We evaluated the metrics for both *failure* phase (when flows are switched to healthy links from failed links) and *recovery* phase (when flows are switched back from healthy links to recovered links).

The speed at which gateway gets notified of the failure is the main factor in the performance of any layer 3 handoff. The BFD protocol is used by the gateway to evaluate the health of the WAN link and the Internet path. The period of

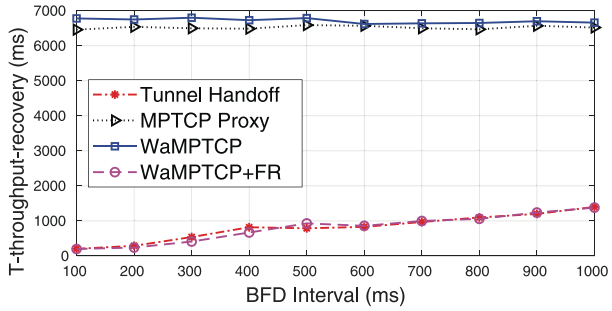


Fig. 13. BFD impact over T-throughput-recovery.

BFD handshake is equal to the BFD timer. By default BFD declares path failure after 3 failed handshakes, and declares a path recovered after a single successful handshake.

Figures 11, 12, 13, and Table II study the impact of BFD timer on our metrics for each mechanism. Those experiments confirmed that performance of tunnel handoff and FR at TCP level is proportional to the setting of the BFD timer, both for the failure and recovery phase. The actual time for tunnel handoff is larger than the time of the Layer 3 handoff. MPTCP without FR are not using BFD, and therefore is not impacted by BDF settings. As shown in Figure 11, tunnel handoff would need to use a very small BFD timer to compete with how MPTCP handles failure. However, this increases overhead and the risks of false positive.

The metrics do confirm many of our earlier findings. MPTCP provides a much better failure performance, and MPTCP without FR suffers from worse recovery performance due to stalled flows. Figure 11 also shows that *T-throughput-failure* is zero with MPTCP. This confirms that the throughput of the subflow on the healthy link is not impacted. *T-latency-recovery* is always zero across all mechanisms, because there is no retransmissions.[2]

The traffic traces in Section V-C show near instant handoff for MPTCP, the metrics are useful to show that this is not the case. *T-latency-failure* in Table II and Figure 12 shows that for all MPTCP mechanisms, some data gets stuck for around 320ms on the failed subflow before being retransmitted on the healthy subflow. This level of delay could impact latency sensitive applications. This measurement can also be use to properly dimension MPTCP receive buffers to avoid packet discards due to receiver blocking [27]. Such outages are highly unpredictable and infrequent, so MPTCP mechanisms that schedule packets based on their predicted latency will not be able to compensate for it. We believe this level of delay is way to large for high speed networks, and the MPTCP scheduler should be improved to reduce it when RTT is low.

In Figure 13, *T-throughput-recovery* of MPTCP proxy or WaMPTCP are much higher than that of tunnel handoff or WaMPTCP with FR. This is because even though a failed link is recovered, and the gateway knows about it, the gateway has no way to "ask" end systems to send packets over it immediately. The application traffic has to wait until end systems issue

[2]Even though packet retransmissions may be triggered due to congestion, we did not take it as caused by failure.

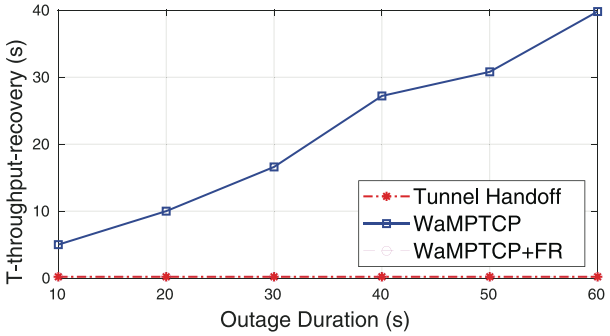


Fig. 14. Outage impact over T-throughput-recovery.

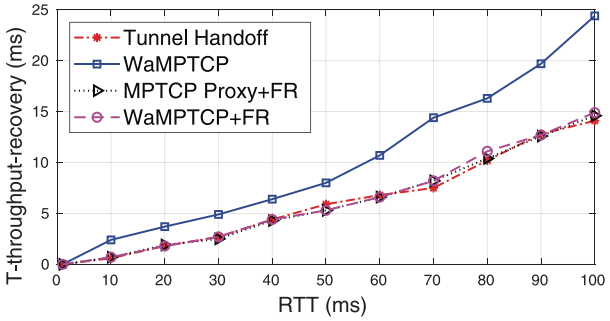


Fig. 15. RTT impact over T-throughput-recovery.

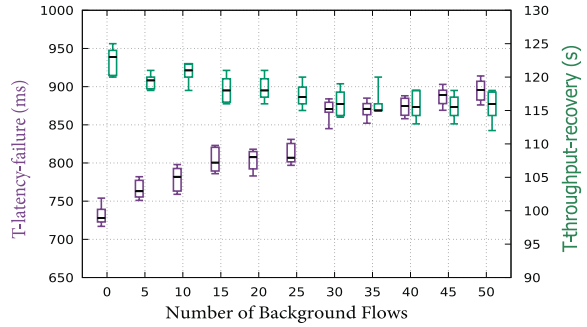


Fig. 16. Impact of TCP contention.

packet retries over suspended connection, and get response. The retry rate is determined by RTO. RTO in TCP grows in an exponential way during an outage. Thus, when an outage last longer, TCP probes the link less often at recovery time, and the probability of a quick recovery decreases.

Figure 14 shows the impact of the outage duration on the recovery phase and the *T-throughput-recovery* metric. Without FR, MPTCP recovery is very slow, and grows with outage duration. During outage, TCP resends SYN packet after 0.2 seconds by default in Linux, and the interval grows exponentially with a default maximum as 120 seconds. The default number of retry is 15, and thus the maximum outage that can be tolerated is 924.6 seconds. For high-availability server, the suggested the number of retries is 3 [13]. In this case, if the outage duration is larger than around one second, a stalled subflow will never resume.

In contrast, tunnel handoff and FR utilize BFD for link status detection, and therefore are independent of outage duration

TABLE III
IMPACT OF TCP CONGESTION CONTROL (BFD=1S, TPUT IS THROUGHPUT)

	T-latency-fail (s)		T-tput-recover (s)		Tput (mbps)	
RTT	*1ms*	*100ms*	*1ms*	*100ms*	*1ms*	*100ms*
cubic	0.43	0.71	1.26	12.3	179	126.0
reno	0.41	1.00	1.65	38.4	179	125.0
vegas	0.22	0.65	1.92	34.9	175	125.0
illinois	0.44	0.86	1.25	23.2	179	124.0
veno	0.33	0.74	1.31	26.8	179	125.0
westwood	0.27	0.55	1.43	14.9	179	126.0
olia	0.38	0.90	1.10	>120	179	83.9
balia	0.62	0.72	1.06	82.7	179	93.5
wvegas	0.64	0.59	1.46	>120	179	54.6

(Figure 14). If an MPTCP subflow is not stalled, *T-throughput-recovery* is approximately layer 3 recovery time plus the delay for congestion window (cwnd) growth. Congestion window growth closely depends on round trip time, bandwidth, and congestion on the path (Section V-E).

E. TCP Congestion Control

Figure 15 shows that *T-throughput-recovery* grows as RTT of the path grows for all mechanisms. A higher RTT forces TCP to use a larger congestion window (cwnd), and it also makes opportunities for cwnd adjustments less frequent, and thus it takes more time to recover the full cwnd. Without FR, the cwnd of subflow routed back to the recovered link has to grow to reach the entire tunnel bandwidth from TCP slow start. Mechanisms with FR do not have to perform an entire cwnd recovery, since subflows are not stalled (Section III-B), giving smaller *T-throughput-recovery*. Further, the cwnd of subflow on healthy link and the cwnd of subflow on just recovered link can grow in parallel.

Table III shows the impact of TCP congestion control algorithms over the performance of WaMPTCP with FR. In this set of experiments, six TCP congestion control algorithms are uncoupled, while the last three - olia, balia, and wvegas - are coupled and specifically designed for MPTCP [8]. We artificially added 1ms or 100ms latency to both WAN links using Linux Traffic Control [7] to emulate a large Bandwidth $\times$ Delay Product (BDP) network. Different congestion control algorithms suit different network characteristics. For example, Reno underutilizes "long fat" paths, and after packet losses its cwnd grows by one every RTT, which explains its slower recovery. We verified that for the coupled congestion control algorithms, fairness is proportional to the number of MPTCP sessions, while for uncoupled congestion control algorithms, the fairness is proportional to the number of subflows. Higher RTT reduces achievable throughput in all cases. Congestion windows of three coupled congestion control algorithms grow very slowly. Two of them do not even get recovered after 120 seconds. They suffer from lower throughput at 100ms RTT, which indicates the need of further improvements before they can be used for SD-WAN scenario.

Above tests explore only a single MPTCP session in idle links. Figure 16 shows an increasing number of TCP background flows competing for the WAN links. Contention does not impact *T-throughput-recovery* much. *T-latency-failure*

TABLE IV
HTTP DOWNLOAD DURATION WITH INTERMITTENT LINK IN CONTROLLED TESTBED (SECONDS FOR 2GB FILE)

BFD timer	*1s*					*100ms*				
Time between link events (on or off)	*1s*	*2s*	*3s*	*4s*	*none*	*1s*	*2s*	*3s*	*4s*	*none*
Tunnel Handoff	>3600	>3600	>3600	680	177	305	235	205	197	177
MPTCP Tunnel	stall	stall	stall	stall	94	stall	stall	stall	stall	94
MPTCP Proxy	176	175	175	173	91	176	175	175	173	91
WaMPTCP	176	175	175	173	91	176	175	175	173	91
WaMPTCP+FR	176	175	175	168	91	162	155	146	131	91

TABLE V
TCP TRANSACTION RATE WITH INTERMITTENT LINK IN CONTROLLED TESTBED (TRANSACTIONS/SECONDS)

BFD timer	*1s*					*100ms*				
Time between link events (on or off)	*1s*	*2s*	*3s*	*4s*	*none*	*1s*	*2s*	*3s*	*4s*	*none*
Tunnel Handoff	754.1	839.1	2021.8	2700.1	3854.0	1989.4	3437.9	3737.7	3692.8	3911.9
MPTCP Tunnel	stall	stall	stall	stall	3923.5	stall	stall	stall	stall	3943.2
MPTCP Proxy	3854.02	3883.3	3807.7	3871.9	3946.4	3847.6	3842.7	3820.9	3845.7	3876.3
WaMPTCP	3835.7	3822.7	3807.2	3884.0	4061.8	3835.1	3864.2	3886.6	3804.5	4048.5
WaMPTCP+FR	3879.7	3854.6	3873.9	3869.5	4036.8	3828.0	3855.3	3889.3	3818.6	4054.5

increases with contention, because the congestion window size shrinks and causes more delay for retransmitted packets.

F. Handoff Impact on Applications

WAN link failures (e.g., link flapping) happen more frequently than expected [50]. Therefore, we evaluate the impact of intermittent links on applications [40], [58]. For example, it may occur on a wireless link with slow fading [48]. In our tests, the backup link is always healthy, and the primary link alternates between on and off state at periodic interval. 50% of the time is blocked using Netfilter. Half of the handoff are reactive and the other half is proactive, which is more strenuous than frequent policy changes [21] (only proactive handoff).

Table IV explores the impact of intermittent links on download performance. The client downloads a 2GB file from a Web server using wget. This represents throughput-sensitive applications, and the *T-throughput-failure* and *T-throughput-recovery* metrics explain the non obvious application performance. With BFD timer of 1s, performance is low when using Tunnel handoff, especially for frequent outages. *T-throughput-failure* is greater than 3s (Figure 11), so TCP does not have enough time to fully recover either on the primary link or the backup link before the next outage. A shorter BFD timer of 100ms has a shorted *T-throughput-failure*, enabling TCP to recover between outages, thus improving performance. The metrics are independent of the outage duration (Section V-D), and therefore the performance is mostly proportional to the outage frequency (for the same overall link duty cycle).

MPTCP based mechanisms without FR use only a single link after the first outage, and thus the download time is almost twice the time larger than those without outage. *T-throughput-recovery* is greater than the outage duration (Fig. 13), effectively preventing usage of the intermittent link even when it is healthy. *T-throughput-recovery* increases with the outage duration (Fig. 14), so only an increase in the duty cycle of the bad link would allow the affected subflow to recover. Despite the large T-latency-failure (Figure 12), the throughput of MPTCP with outages is better than the performance of a single TCP flow without outages, which indicates that the impact of buffer blocking and stragglers is fairly minimal in those tests. FR with a short BFD timer has a much lower *T-throughput-recovery* (Fig. 13), this enables the affected subflow to exploit the intermittent link during the time between outages, with performance mostly proportional to the outage frequency. *T-throughput-recovery* is independent of the outage duration (Fig. 14), thus the performance is simply proportional to the outage frequency. MPTCP Tunnel is completely stalled in presence of frequent outages and needs to be restarted.

Table V explores the impact of intermittent links on transaction rate by using Netperf TCP_RR test. This measures the number of bidirectional transactions on a TCP connection. This represents latency-sensitive applications: higher transaction rate can only be achieved with shorter round trip latency.

With BFD timer of 1s, Tunnel handoff makes slow progress for frequent outages, *T-latency-failure* is greater than 3s (Figure 12), so TCP does not have enough time to fully recover either on primary link or backup link before the next outage. A shorter BFD timer of 100ms has a shorted *T-latency-failure*, enabling TCP to recover between outages, thus increasing performance. MPTCP tunnel is stalled when dealing with intermittent link. Without outages, the performance of MPTCP mechanisms is just slightly higher than tunnel handoff. This means that parallelism does not help in this experiment when there is no failure, because there is only one tiny transaction at a time for this latency test. In the presence of outages, MPTCP based mechanisms is unaffected by the outage frequency because they can switch to the healthy link. The MPTCP scheduler prefers the link with the lowest RTT, therefore after the first few outages, it permanently avoids the intermittent link. In contrast, BFD can only do binary evaluation of the link (on or off status). *T-latency-failure* explains the slight performance degradation of MPTCP

TABLE VI
HTTP DOWNLOAD DURATION WITH INTERMITTENT LINK IN GENI TESTBED, 100MS BFD (SECONDS FOR 20MB FILE)

Intermittent link	*low-latency link*					*high-latency link*				
Time between link events (on or off)	*1s*	*2s*	*3s*	*4s*	*none*	*1s*	*2s*	*3s*	*4s*	*none*
Tunnel Handoff	350	218	137	125	49	167	113	80	72	49
MPTCP Tunnel	stall	stall	stall	stall	32	stall	stall	stall	stall	32
MPTCP Proxy	90	77	68	61	32	62	58	50	49	32
WaMPTCP	92	78	67	63	32	62	57	51	47	32
WaMPTCP+FR	62	71	62	56	32	58	52	49	42	32

TABLE VII
TCP TRANSACTION RATE WITH INTERMITTENT LINK IN GENI TESTBED, 100MS BFD (TRANSACTIONS / SECONDS)

Intermittent link	*low-latency link*					*high-latency link*				
Time between link events (on or off)	*1s*	*2s*	*3s*	*4s*	*none*	*1s*	*2s*	*3s*	*4s*	*none*
Tunnel Handoff	7.72	9.67	12.48	14.09	19.55	11.73	14.11	14.78	13.53	9.83
MPTCP Tunnel	stall	stall	stall	stall	19.45	stall	stall	stall	stall	19.36
MPTCP Proxy	10.98	11.22	12.45	13.98	19.56	19.34	19.31	19.33	19.33	19.34
WaMPTCP	11.19	11.65	12.33	14.19	19.32	19.34	19.34	19.31	19.33	19.34
WaMPTCP+FR	11.13	11.52	12.35	14.78	19.34	19.33	19.38	19.33	19.34	19.37

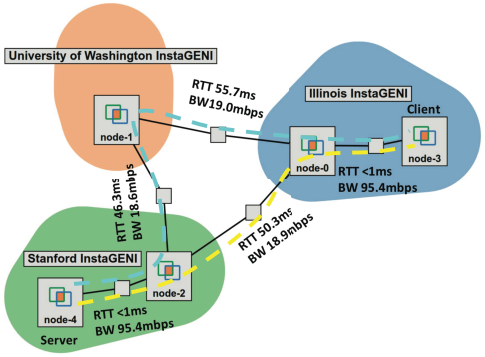


Fig. 17. GENI testbed setup.

schemes in the presence of outages. A few transactions are stuck on the failed link until they are retransmitted on the healthy link, blocking overall progress. We only evaluated the default MPTCP scheduler, leaving evaluation on other advanced MPTCP schedulers [27], [30] to be a future work.

G. IP Provisioning (DHCP)

WaMPTCP provisions multiple IP addresses to end systems rather than a single one (Section III-A). The latency of regular IP provisioning via DHCPv4 and DHCPv6 is around 1.7 seconds. DHCPv4 provisioning over WaMPTCP takes around 2.6 seconds, whereas DHCPv6 provisioning over WaMPTCP takes around 1.7 seconds. WaMPTCP with DHCPv6 can get all IP addresses provisioned in a single response, whereas WaMPTCP with DHCPv4 requires additional requests, each for one subnet.

H. Evaluation in the Wild

We deployed WaMPTCP in the real world to confirm the findings of our testbed. We performed experiments over the Internet in both GENI testbed [6] and Amazon AWS.

In GENI testbed, we created a 5-node topology as shown in Figure 17. We tested handoff performance by performing the experiments with intermittent links (Section V-F), using 100ms BFD timer. The major difference is that the overall latency of those WAN paths are much higher, and one path has twice as much latency as the other path. Therefore, we ran the experiment with the primary and intermittent link being either the low-latency link or the high-latency link, respectively.

Table VI shows the impact of intermittent links on download performance, and confirms the results from the controlled testbed. For tunnel handoff, the penalty of handoff is comparatively higher, this is due to the higher RTT causing larger *T-throughput-failure* and *T-throughput-recovery* (Figure 15). Without outages, MPTCP does not have twice the performance of Tunnel Handoff, it cannot exploit both links fully due to the higher and differing RTTs. For MPTCP, outages on the low latency link produce more performance degradation than on the high latency link, which is expected as the MPTCP scheduler by default prefers to send packets over low-latency path. The performance of MPTCP with outages is worse than the performance of a single TCP flow without outages. For those RTTs, the *T-throughput-recovery* can be as high as 20s (Figure 15), MPTCP is suffering from a straggler issue where a few packets are getting stuck on the stalled MPTCP subflow. FR reduces *T-throughput-recovery* (Figure 15), which explain why FR outperforms regular MPTCP.

Table VII explores the impact of intermittent links on transaction rate by using Netperf TCP_RR test. The TCP transaction rate depends on latency, so tunnel handoff performs as well as MPTCP based solutions if the primary link is the low-latency link with no failure. Interestingly, when tunnel handoff uses the high latency link as primary, the occurrence of outages actually increases transaction rate (from 9.8 to 13.5), despite the cost of handoffs, because the outages force the traffic to use the low-latency link which has higher performance. With MPTCP, when the intermittent path is the high-latency path, there is no performance loss: the MPTCP scheduler by default prefers to send packets over low-latency path, so it avoids the intermittent link entirely. When the intermittent path is the low-latency path, the scheduler sends transaction roughly in

TABLE VIII
NETWORK PARAMETERS FOR EC2 ACCESS

	RTT	Throughput
Cellular	138~162 ms	1.2 Mb/s
WiFi	181~242 ms	1.64 Mb/s
Ethernet	100~120 ms	6.4 Mb/s

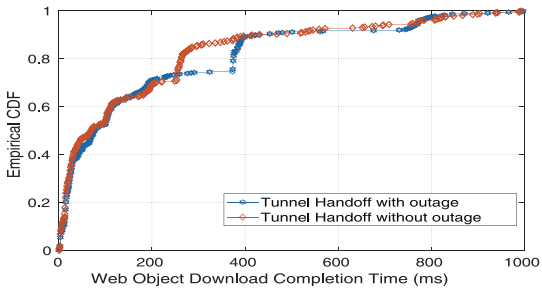


Fig. 18. Web browsing via Handoff.

equal measure one each path. Thus for infrequent outages, the end result is the average of the performance of each path. With increased outage frequency, more transactions are delayed due to the high *T-latency-failure*, decreasing performance.

In the next experiment, we instantiated a HTTP server (Apache 2.4.18) in Amazon Elastic Compute Cloud (EC2) in eu-west-1 zone. The local gateway uses three types of access networks: a cellular network by using the hotspot feature on a mobile phone (serving as a 2.4Ghz wireless routers), university WiFi (connecting it with a PAU06 300Mbps Wireless N USB Adapter), and university Ethernet (1 Gbps NIC). Those WAN links offer different raw download performance (Table VIII). A CNN's homepage (consisting of 216 objects) is downloaded and stored in an EC2 server. A security group is properly configured to make tunnel connections and server access feasible. Apache 2.4.18 is used as the HTTP server, and HTTP persistent connections are enabled with default 5-second Keep Alive Timeout. We choose Ethernet as primary link because it is the most reliable one among the three.

We manually created a single 1-second outage on the Ethernet path and investigate the cumulative distribution function (CDF) of Web object loading time. Figure 18 shows that for tunnel handoff with 100ms BFD timer, the application suffers from the outage (separation of blue and red curves). Figure 19 shows that WaMPTCP with FR can successfully hide the outage from the application.

VI. RELATED WORK

The innovation of SDN technology brings new possibilities for WAN. It drives network optimization by relying on logically centralized control of network resources. Our work builds on related research as categorized below.

A. Link Utilization in SD-WAN

SD-WAN solutions [25], [52], [55] have been extensively studied in both enterprise and literature. Existing SD-WAN productions [12] use techniques similar to Tunnel Handoff for reliability, and certain solutions provide more features such as aggregated link utilization and fast failover.

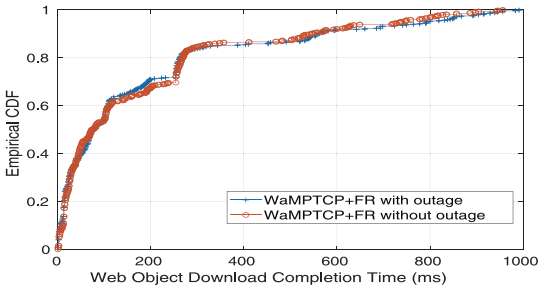


Fig. 19. Web browsing via WaMPTCP.

However, implementation details of these solutions are rarely open-sourced. In the literature, the benefits of using MPTCP for throughput and failover are studied in various scenarios [20], [32], [38], [46], [54], [56]. Moreover, several MPTCP proxies [5], [11] are implemented for the purpose of link utilization. In contrast, we focus on applying MPTCP to an SD-WAN scenario which has not been fully explored by taking practical campus network constraints (e.g., end-host only uses one physical NIC to connect gateway) into account, and studying the metrics directly related to application performance during vertical handoff.

B. Fast Failure Recovery

Traditional network relies on fast reroute mechanism when there is an edge cut. For example, failure recovery time gets remarkably reduced when fast reroute mechanism is applied in IP and MPLS networks. In commodity devices, loop-free alternate [44] is a common method to support fast reroute. RSVP-TE [17] is typically applied to manage recovery path in MPLS fast reroute.

In SDN, shortest path algorithms [35], [41] are used to compute recovery paths. SDN controllers such as ONOS and Onix take Dijkstra algorithm for shortest path computation. There are many novel fast recovery mechanisms proposed in the recent years. For example, Shared Queue Ring [42] is an on-switch mechanism that completely eliminates packet loss during link failures by diverting the affected flows seamlessly to alternative paths. Switches managed by Blink [23] maintain backup next hops by analyzing TCP flows at line rate. Sahri and Okamura [41] propose a novel fast failover architecture by having a central controller to compute backup path so as to reduce switching delay. Ranadive and Medhi [39] explore the effect of route fluctuation on TCP. Carpa *et al.* [21] explore the effect of frequent SDN route changes on TCP.

This article takes one step further. We not only detected failure and rerouted flows [51], but also identified the gap when BFD integrates with MPTCP. For better observation of network performance, we designed two metrics to measure the L4 performance, and design fast failover mechanism to settle link failure and achieve restoration.

VII. CONCLUSION

We have presented WaMPTCP, a novel SD-WAN solution for providing better resilience under WAN link failures. We motivated the design for WaMPTCP by illustrating

the problems associated with *ECMP* and *Tunnel Handoff* commonly employed by existing SD-WAN solutions to handle WAN link failures, and showed that it can lead to significant application performance degradation in the evaluation. WaMPTCP enables applications to generate multiple MPTCP flows even with a single physical interface. This is further augmented with an MPTCP proxy to accommodate end systems without native MPTCP support and integrated with tunnel handoff to support UDP traffic. We also introduced two new metrics to better capture and quantify the impact of WAN link failures on application performance. Through extensive evaluation in emulated testbed and real-world deployment, we demonstrated the performance gain of WaMPTCP over existing SD-WAN solutions. WAN awareness enables MPTCP on the client to optimally aggregate the bandwidth of all the WAN links available on the gateway, in a scalable way. MPTCP drastically reduces the impact of path failures, and Fast Recovery drastically reduces the time take advantage of a recovered path. While the focus of this article is on aggregating multiple WAN links for providing better resilience under failures to minimize their impact on application performance, WaMPTCP can also be used to further enhance WAN link utilization by intelligently distributing MPTCP subflows to WAN links and support application-aware SD-WAN traffic engineering – an in-depth exploration of these topics will be left to a future paper.

REFERENCES

[1] (2002). *HPsockd*. [Online]. Available: https://tracker.debian.org/pkg/hpsockd

[2] (2006). *Dante*. [Online]. Available: https://www.inet.no/dante/

[3] (2006). *Dhcpy6d*. [Online]. Available: https://dhcpy6d.ifw-dresden.de/

[4] (2017). *MPTCP Tunnel*. [Online]. Available: https://github.com/dfshan/mptcp-tunnel

[5] (2018). *OpenMPTCProuter—Internet Connection Bonding*. [Online]. Available: https://www.openmptcprouter.com/

[6] (2018). *GENI Network*. [Online]. Available: http://www.geni.net/

[7] (2018). *Linux Traffic Control*. [Online]. Available: https://linux.die.net/man/8/tc

[8] (2018). *MultiPath TCP—Linux Kernel Implementation*. [Online]. Available: https://www.multipath-tcp.org/

[9] (2018). *Netfilter*. [Online]. Available: https://netfilter.org/documentation

[10] (2018). *TCP over TCP*. [Online]. Available: http://sites.inka.de/bigred/devel/tcp-tcp.html

[11] (2019). *MPTCP Proxy—OverTheBox Github Page*. [Online]. Available: https://github.com/ovh/overthebox

[12] (2020). *Comparison of the SD-WAN Vendor Solutions*. [Online]. Available: https://www.netmanias.com/en/post/oneshot/12481/sd-wan-sdn-nfv/comparison-of-the-sd-wan-vendor-solutions

[13] (2020). *TCP Retries2 Parameter in Linux*. [Online]. Available: https://access.redhat.com/solutions/726753

[14] A. A. Abouzeid and S. Roy, "TCP in networks with abrupt delay variations and random loss," in *Proc. MILCOM Commun. Netw. Centric Oper. Creating Inf. Force*, vol. 1. McLean, VA, USA, Oct. 2001, pp. 726–730.

[15] 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.*, 2010, p. 19.

[16] S. Alexander and R. Droms, "DHCP options and BOOTP vendor extensions," IETF, RFC 2132, 1997.

[17] A. Atlas, G. Swallow, and P. Pan, "Fast reroute extensions to RSVP-TE for LSP tunnels," IETF, RFC 4090, May 2005.

[18] S. Barré, C. Paasch, and O. Bonaventure, "MultiPath TCP: From theory to practice," in *Proc. 10th Int. IFIP TC 6 Conf. Netw.*, 2011, pp. 444–457.

[19] M. H. C. Raiciu and D. Wischik, "Coupled congestion control for multipath transport protocols," IETF, RFC 6356, Oct. 2011.

[20] A. Croitoru, D. Niculescu, and C. Raiciu, "Towards WiFi mobility without fast handover," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Oakland, CA, USA, 2015, pp. 219–234.

[21] R. Cârpa, M. Dias de Assunção, O. Glück, L. Lefèvre, and J.-C. Mignot, "Evaluating the impact of SDN-induced frequent route changes on TCP flows," in *Proc. 13th Int. Conf. Netw. Serv. Manag. (CNSM)*, Tokyo, Japan, Nov. 2017, pp. 1–9.

[22] G. Detal, C. Paasch, and O. Bonaventure, "Multipath in the middle(box)," in *Proc. Workshop Hot Topics Middleboxes Netw. Funct. Virtualization*, 2013, pp. 1–6.

[23] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *Proc. 16th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, 2019, pp. 161–176.

[24] C.-Y. Hong *et al*. "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proc. Conf. ACM Spec. Interest Group Data Commun.*, 2018, pp. 74–87.

[25] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 15–26.

[26] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[27] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *Proc. Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, Jun. 2014, pp. 1222–1227.

[28] F. Le, E. Nahum, V. Pappas, M. Touma, and D. Verma, "Experiences deploying a transparent split TCP middlebox and the implications for NFV," in *Proc. ACM SIGCOMM Workshop Hot Topics Middleboxes Netw. Funct. Virtualization*, 2015, pp. 31–36.

[29] G. Li and P. Jin, "A dynamically adjusted congestion control algorithm for TCP," *J. Inf. Comput. Sci.*, vol. 9, pp. 4691–4697, Dec. 2012.

[30] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. 13th Int. Conf. Emerg. Netw. EXp. Technol.*, 2017, pp. 147–159.

[31] X. Liu, D. Shan, R. Shu, and T. Zhang, "MPTCP tunnel: An architecture for aggregating bandwidth of heterogeneous access networks," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–11, Mar. 2018.

[32] H. Nam, D. Calin, and H. Schulzrinne, "Towards dynamic MPTCP path control using SDN," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Seoul, South Korea, Jun. 2016, pp. 286–294.

[33] B.-H. Oh and J. Lee, "Feedback-based path failure detection and buffer blocking protection for MPTCP," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3450–3461, Dec. 2016.

[34] B. Pfaff *et al.*, "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Oakland, CA, USA, 2015, pp. 117–130.

[35] T. Qu, R. Joshi, M. C. Chan, B. Leong, D. Guo, and Z. Liu, "SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–12.

[36] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, Aug. 2011.

[37] C. Raiciu, D. Niculescu, M. Bagnulo, and M. J. Handley, "Opportunistic mobility with multipath TCP," in *Proc. 6th Int. Workshop MobiArch*, 2011, pp. 7–12.

[38] C. Raiciu *et al.*, "How hard can it be? designing and implementing a deployable multipath TCP," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, San Jose, CA, USA, Apr. 2012, pp. 399–412.

[39] U. Ranadive and D. Medhi, "Some observations on the effect of route fluctuation and network link failure on TCP," in *Proc. 11th Int. Conf. Comput. Commun. Netw.*, Scottsdale, AZ, USA, Oct. 2001, pp. 460–467.

[40] M. B. Reynolds, "Mitigating TCP degradation over intermittent link failures using intermediate buffers," Dept. Elect. Comput. Eng., Air Force Inst. Technol., Wright-Patterson AFB, OH, USA, 2017.

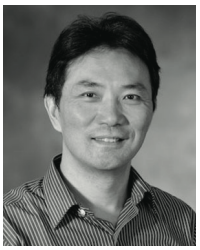
[41] N. M. Sahri and K. Okamura, "Fast failover mechanism for software defined networking: OpenFlow based," in *Proc. 9th Int. Conf. Future Internet Technol. (CFI)*, 2014, pp. 1–2.

[42] E. Sakic, M. Avdic, A. Van Bemten, and W. Kellerer, "Automated bootstrapping of a fault-resilient in-band control plane," in *Proc. Symp. SDN Res.*, 2020, pp. 1–13.

[43] B. Schlinker *et al.*, "Engineering egress with edge fabric: Steering oceans of content to the world," in *Proc. Conf. ACM Spec. Interest Group Data Commun.*, 2017, pp. 418–431.
[44] M. Shand and S. Bryant, "IP fast reroute framework," IETF, RFC 5714, 2010.
[45] V. Sharma and F. Hellstrand, "Framework for multi-protocol label switching (MPLS)-based recovery," IETF, RFC 3469, Feb. 2003.
[46] H. Shi *et al.*, "STMS: Improving MPTCP throughput under heterogeneous networks," in *Proc. USENIX Conf. Usenix Annu. Techn. Conf. (ATC)*, Boston, MA, USA, Jul. 2018, pp. 719–730.
[47] W. Silva, "Make flows great again: A hybrid resilience mechanism for openflow networks," *Information*, vol. 9, no. 6, p. 146, Jun. 2018.
[48] J. Tourrilhes, "Fragment adaptive reduction: Coping with various interferers in radio unlicensed bands," in *Proc. Int. Conf. Commun. (ICC)*, vol. 1. Helsinki, Finland, Jun. 2001, pp. 239–244.
[49] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "SDN and OpenFlow evolution: A standards perspective," *Computer*, vol. 47, no. 11, pp. 22–29, Nov. 2014.
[50] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 315–326, Aug. 2010.
[51] N. L. M. V. Adrichem, B. J. V. Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, London, U.K., 2014, pp. 61–66.
[52] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: WAN-aware optimization for analytics queries," in *Proc. 12th USENIX Conf. Oper. Syst. Design Implement. (OSDI)*, Savannah, GA, USA, Nov. 2016. pp. 435–450.
[53] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A measurement study on the impact of routing events on end-to-end Internet path performance," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2006, pp. 375–386.
[54] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implement.*, 2011, pp. 99–112.
[55] D. Z. Tootaghaj, F. Ahmed, P. Sharma, and M. Yannakakis, "Homa: An efficient topology and route management approach in SD-WAN overlays," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Toronto, ON, Canada, Jul. 2020, pp. 2351–2360.
[56] S. Zannettou, M. Sirivianos, and F. Papadopoulos, "Exploiting path diversity in datacenters using MPTCP-aware SDN," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Messina, Italy, 2016, pp. 539–546.
[57] Y. Zhang, H. Mekky, Z.-L. Zhang, F. Hao, S. Mukherjee, and T. V. Lakshman, "SAMPO: Online subflow association for multipath TCP with partial flow records," in *Proc. IEEE INFOCOM 35th Annu. Int. Conf. Comput. Commun.*, San Francisco, CA, USA, 2016, pp. 1–9.
[58] A. Zimmermann and A. Hannemann, "Making TCP more robust to long connectivity disruptions (TCP-LCD)," IETF, RFC 6069, 2010.

Yang Zhang received the B.S. and M.S. degrees in software engineering and computer science from Southeast University, Nanjing, China, in 2011 and 2013, respectively, and the Ph.D. degree in computer science from the University of Minnesota–Twin Cities in 2019, advised by Prof. Z.-L. Zhang. His research interests are computer networks and distributed systems.

Jean Tourrilhes received the M.S. degree in computer science from Ecole Nationale Superieure des Telecommunications, Paris. He is a Researcher with the Networking and Communication Lab, Hewlett Packard Labs, Palo Alto, CA, USA. His research interests over the years include wireless LANs, embedded linux, SDN, openflow, QoS, datacenter networking, NFV, SD-WAN, and bandwidth measurement.



Zhi-Li Zhang (Fellow, IEEE) received the B.S. degree in computer science from Nanjing University, Nanjing, China, in 1986, and the M.S. and Ph.D. degrees in computer science from the University of Massachusetts, Amherst, MA, USA, in 1992 and 1997, respectively. He joined the Computer Science and Engineering Faculty, University of Minnesota, where he is currently a Qwest Chair Professor and a Distinguished McKnight University Professor. His research interests include computer communication and networks, Internet technology, content distribution systems, cloud computing, and emerging IoT applications. He is a co-recipient of several best paper awards and has received a number of other awards. He has Co-Chaired several conferences/workshops including IEEE Conference on Computer Communications, IEEE International Conference on Network Protocols, ACM International Conference on Emerging Networking Experiments and Technologies, ACM CoNext, IFIP Networking, and has been a Member of the TPC of numerous conferences/workshops. He is a Member of the Association for Computing Machinery.

Puneet Sharma (Fellow, IEEE) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology Delhi, New Delhi, India, and the Ph.D. degree in computer science from the University of Southern California. He is the Head of the Networked Systems Group and a Distinguished Technologist with Hewlett Packard Labs where he leads Edge to Cloud Infrastructure Research Portfolio. His work on mobile collaborative communities was featured in the New Scientist Magazine. His research focuses on edge computing, SDWAN, virtualization, IoT, NFV, SDN, applied machine learning, network monitoring, wireless networks, and data center networks.