Endpoint-transparent Multipath Transport with Software-defined Networks

Dario Banfi*‡, Olivier Mehani*, Guillaume Jourjon†, Lukas Schwaighofer‡, Ralph Holz§

* NICTA, Sydney, Australia; Email: olivier@mehani.name
† Data61, CSIRO, Sydney, Australia; Email: {first.last}@data61.csiro.au
‡ Faculty of Informatics, Technical University of Munich, Germany; Email: {first.last}@tum.de
§ School of IT, University of Sydney, NSW, Australia; Email: ralph.holz@sydney.edu.au

***Abstract*—Multipath forwarding consists of using multiple paths simultaneously to transport data over the network. While most such techniques require endpoint modifications, we investigate how multipath forwarding can be done inside the network, transparently to endpoint hosts. With such a network-centric approach, packet reordering becomes a critical issue as it may cause critical performance degradation. We present a Software Defined Network architecture which automatically sets up multipath forwarding, including solutions for reordering and performance improvement, both at the sending side through multipath scheduling algorithms, and the receiver side, by resequencing out-of-order packets in a dedicated in-network buffer. We implemented a prototype with commonly available technology and evaluated it in both emulated and real networks. Our results show consistent throughput improvements, thanks to the use of aggregated path capacity. We give comparisons to Multipath TCP, where we show our approach can achieve a similar performance while offering the advantage of endpoint transparency.**

***Index Terms*—multipath transport, SDN, OpenFlow, Open vSwitch**

I. Introduction

IP networks are inherently multipath. Yet, the existence of multiple paths between two endpoints is rarely leveraged. This issue can be ascribed to the fact that only lower layers can establish an accurate view of the network topology, while only upper layers are able to control transmission rate and end-to-end connectivity.

Nonetheless, solutions have been proposed at various layers to enable specific use-cases and improve performance. Examples are given at layers 2–3 for data-centres with, *e.g.*, BCube [1] or DCell [2], or at layer 4 for multi-homed devices with Multipath TCP (MPTCP) [3] or Concurrent Multipath Transfer for SCTP (CMT-SCTP) [4].

The most prominently quoted motivations for multipath are the potential for continuity of connectivity in case of path failure or congestion (*i.e.*, fail-over or load-balancing), or capacity aggregation to speed up high volume transfers between endpoints [*e.g.*, 5, for MPTCP].

Layer-2 multipath topologies [*e.g.*, 6], have been successfully deployed and used within fully-controlled data-centre networks. End-to-end multipath support throughout the public Internet is however limited [7] due to the requirement to modify end-hosts. Heterogeneous network paths also worsen the issue of packet reordering, creating head-of-line blocking delays, and sometimes leading to worse performance than single-path transfers [8].

In this paper,[1] we attempt to join both lower- and upper-layer approaches and merge their successes through the use of SDN. We aim to satisfy the following goals: capacity aggregation, ease of end-to-end deployment, adaptivity to failures, and automatic path computation. To this end, we introduce the MPSDN architecture, comprising an SDN controller with better knowledge and control of available paths than endpoint-only layer-4 solutions, as well as modifications of the Open vSwitch implementation and OpenFlow protocol to enable finer packet scheduling and reordering within the network, without need for explicit end-host support.

The solution can be deployed with either layer-2 forwarding or layer-3 routing or tunnelling, and the controller does not require full control of the network hops. We show that this approach enables performance similar to MPTCP's while lifting the requirement for end-host modifications. The focus of this paper is on TCP, but we note that our proposal can handle other transport protocol in a similar fashion [9]. Our work also allows us to identify some non-trivial issues when implementing layer-4 switching and scheduling with SDN solutions.

The proposed mechanism can offer benefits in several scenarios where additional bandwidth would enhance the Quality of Experience for users. A typical scenario is high-definition video streaming where the bit-rate is higher than the capacity of a single path.[2] Another use-case for this proposal is that of multi-cloud overlay networks between virtualised environments.[3] In this scenario, a user controls the edges of the network and deploys the proposed mechanism to maximise bandwidth utilisation between clouds.

The remainder of this paper is organised as follows: The next section reviews state of the art of multipath approaches in line with the goals of our research. We present the proposed architecture and its implementation in Section III and provide a performance evaluation in both emulated conditions and

[1]This paper improves on the first author's MSc thesis but focuses on TCP only; please refer to [9] for more details and other transport protocols.

[2]A video demonstration of this use-case can be found at https://www.youtube.com/watch?v=hkgf7l9Lshw

[3]See, for example, Docker's overlays https://docs.docker.com/engine/userguide/networking/get-started-overlay/.

Table I: Comparison of characteristics and fulfilment of our goals of state-of-the-art multipath proposals and MPSDN.

	TRILL	**SPB**	**FLARE**	**Harp**	**MPTCP**	**CMT-SCTP**	**AMR**	**OLiMPS**	**MPSDN**
Layer 2	✓	✓	✓		†		✓	✓	✓
Layer 3			✓	✓					✓
Layer 4					✓	✓			‡
SDN-based solution							✓	✓	✓
Bandwidth Aggregation				✓	✓	✓	✓	✓	✓
Easy Deployability	✓	✓					✓	✓	✓
Adaptivity to Failures	✓	✓		✓		✓	✓		✓
Load-Balancing	✓	✓	✓	✓		✓	✓	✓	✓
Automatic Path Computation	✓	✓					✓	✓	✓

† MPTCP was used for aggregation on top of a multihomed L2 network
‡ MPSDN uses L4 knowledge, *e.g.*, sequence numbers, to reorder packets

in a real multi-homed testbed in Section IV. We give some insight and lessons learnt about mixing SDN and multipath in Section V and offer concluding remarks in Section VI.

II. Related Work

Multipath topologies in both layer 2 and layer 3 networks are common, offering multiple communication options for capacity aggregation, load-balancing, and congestion avoidance. This section reviews the state-of-the-art of solutions proposed to leverage those capabilities. We do this layer by layer, from 2 to 4, and offer some insight about previous uses of SDN for this purpose.

Table I summarizes the discussed work in light of our design goals. With "easy deployability" we denote the use of software/hardware that can be incrementally deployed and used on real networks and that is not only an experimental proof-of-concept.

A. Link-layer Multipath

The spanning tree (STP) protocol is extensively used on L2 networks to ensure loop-free forwarding in Ethernet networks. It has the downside of actively pruning paths from the networks which could be utilized for increased bandwidth. Cisco's layer-2 multipath [10] attempts to remediate this by enabling the use of alternate paths, while the IEEE 802.3ad amendment introduces provisions for link aggregation [11]. Neither solution however offers full multipath support across complex topologies.

TRILL (Transparent Interconnection of Lots of Links) [12] uses IS-IS routing to ensure that every bridge has full knowledge of the network, allowing for the creating of an optimal forwarding tree with support for Equal-Cost Multipath (ECMP) [13]. 802.1aq SPB (Shortest Path Bridging) [14] also leverages IS-IS to compute a shortest path through the network. A designated MAC address (used with SPB-MAC) or VLAN ID (SPB-VID) is assigned for each switch, and used as label on each received frames. Packets travel on the shortest path to the edge switch, which again de-encapsulates the frame and sends it to the end device. Neither of these techniques allows aggregated bandwidth because of their use of ECMP-like hashing.

MPTCP, discussed in more details below, has also been suggested as a way to leverage multiple layer-2 paths in data-centres and improve performance and robustness [15]. It has been shown that, with a sufficiently high number of subflows, it is possible to aggregate capacity and increase load-sharing. The downsides of this approach are the necessary end-host support, the lack of multipath capability for other protocols such as UDP or SCTP, and its limitation to data-centres.

B. Network-layer multipath

Flowlet Aware Routing Engine (FLARE) [16] is a dynamic multipath load balancing technique. It uses time delays between packets of the same flow to split them into *flowlets* that may be distributed on different paths. This allows to distribute the traffic between available paths more accurately, as compared to flow-based distribution, while maintaining in-order arrival at the receiver. FLARE has shown, through trace-driven simulations of tier-1 and regional ISPs, that highly accurate traffic splitting can be implemented with very low state overhead and negligible impact on packet reordering. However its focus is on load-balancing and does not offer capacity aggregation.

The Harp network architecture prioritizes foreground traffic and uses multipath to dissipate background transfers [17]. It can leverage path diversity and load imbalance in the Internet to tailor network resource allocation to human needs (foreground vs. background traffic). It also provides better fairness and utilization compared to single-path end-host protocols. Moreover, it can be deployed at either end-hosts or enterprise gateways, thereby aligning the incentive for deployment with the goals of network customers. Packet reordering is performed at the exit gateways to cope with different path latencies. Its focus on background traffic at the exception of all other traffic, however, makes it ill-fitted for our goals.

C. Transport-layer multipath

Extensions to two main transport protocols have been proposed to support multipath. MPTCP [3] introduced a new set of TCP options to enable negotiation between multipath-capable hosts while using backward-compatible TCP packets on each path. SCTP's fail-over supports load-balancing [18] and has been extended to support concurrent multipath transfer [19]. Despite their intrinsic limitation to a single transport protocol, those approaches have seen reasonable success in the lab, with their main barrier to deployment being the need for end-host support.

A very active area of research with transport-layer multipath is enabling packet schedulers to deal with path asymmetry without introducing head-of-line blocking [8]. Most schedulers attempt to distribute packets unevenly or out-of-order across available paths, so they arrive in order at the destination [20]–[26]. An adequate scheduling policy is important to enable the benefits of capacity aggregation in heterogeneous scenarios.

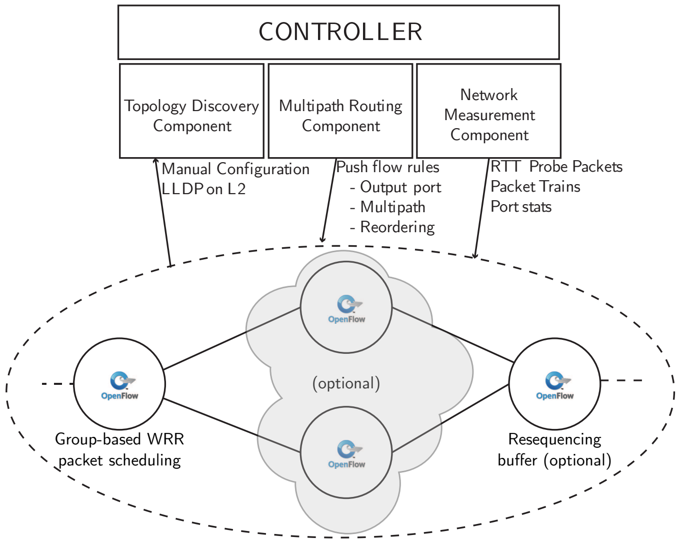


Figure 1: Multipath SDN Architecture

D. *SDN-based multipath solutions*

Multipath in OpenFlow has been proposed back in 2010[4] and later implemented through *Groups* (such as Select or All) to enable L2/L3 multipath forwarding for load-balancing purposes. It has since then been researched extensively [27]–[29], but none of these approaches allows for aggregated bandwidth as they all rely on flow hashing (as does OpenFlow at its core).

Adaptive Multipath Routing (AMR) has been used to perform layer-2 aggregation in data-centres [30]. It splits flows over multiple paths and introduces an architecture which adapts dynamically to network congestion and link failures. An interesting aspect of this approach is its computation of max-flow paths throughout the network to determine the best combination to use. An analogous technique has also been used in *OLiMPS* (OpenFlow Link-layer Multipath Switching) to utilize robust inter-domain connectivity over multiple physical links.

Overall, existing proposals can either not provide aggregated path capacity or are limited to layer-2 forwarding. Layer-4 solutions, while supporting aggregation as their main advantage, lack in deployability as they require end-host support. AMR comes closest to our goals, but is an L2-only solution. Moving forwards, we propose an architecture able to handle both layer-2 and -3 multipath scenarios, while accounting for the scheduling and reordering requirements of heterogeneous paths.

III. Architecture

Our proposed architecture for an endpoint-transparent multipath network consists of a centralized controller with knowledge of the network topology which dynamically sets up loop-less forwarding rules on SDN switches under its control (Figure 1). For the presented proof-of-concept, we focus on two path scenarios only.

[4]http://archive.openflow.org/wk/index.php/Multipath_Proposal

The controller has some knowledge of the network state and views the underlying infrastructure as a directed graph, where costs between switches are given by the latency and capacity of the paths. With this knowledge it computes the optimal multipath forwarding table to send data from one node to the other, maximizing the capacity usage with an algorithm based on the maximum-flow problem. This is similar to AMR [30], but we extend it to layer-3 infrastructures. In case of failure or heavy congestion, the controller will compute an updated forwarding table and push it to the SDN switches.

In the remainder of this section, we present the key concepts of our architecture: the topology discovery and path selection, as well as the packet scheduler and reordering buffer. We also describe how we implemented this architecture in the Ryu OpenFlow controller[5] and how we modified Open vSwitch to support packet reordering on edge switches.

A. *Topology Discovery*

In order to discover the network topology, we both query the forwarding devices using the Link Layer Discovery Protocol (LLDP) when available (*i.e.*, layer 2) or deploy ad hoc mechanisms to estimate end-to-end latency and throughput (*i.e.*, layer 3). In particular, we estimate path latency with a slightly modified Bouet's algorithm [31], which yields high accuracy and has a low network footprint. Unlike NetFlow or measurements using ICMP echo requests, this does not require additional servers or components. The algorithm is run using controller-to-switch messages only.

We use port statistics counters for bandwidth estimation. As shown in OpenNetMon [32], we can accurately monitor a flow's throughput by probing flow statistics periodically. The controller uses a similar approach by periodically requesting port statistic messages from its switches (every 2 seconds in the current implementation). The per-port available capacity is determined by subtracting the maximum capacity with the utilization from the last period of observation.

B. *Path Selection*

In order to maximize the aggregated capacity of multiple paths, the controller uses an algorithm similar to the Edmonds-Karp algorithm to solve the *maximum flow* problem, with a Breadth First Search to find the augmenting paths. It uses the Dijkstra algorithm with min-priority queue to find the shortest paths from source to destination. The estimated available bandwidth between the nodes is used to maximize the overall throughput between the sender and the receiver.

Pilot experiments showed that, in a similar manner as for layer-4 multipath, not all paths are compatible and a very high delay imbalance was detrimental. To select compatible paths, we introduce the concept of *maximum delay imbalance*,

$$\mathrm{MDI} = \frac{d_{max}}{d_{max} + d_{min}} - 0.5\,, \qquad (1)$$

where d_{min} and d_{max} denote the minimal and maximal delays from the candidate paths, and 0.5 a rescaling factor. Its range

[5]https://osrg.github.io/ryu/

is $[0, 0.5]$, where 0 represents completely balanced paths and 0.5 is the limit of imbalance.

This metric is used for different purposes in our solution. If the computed MDI among the selected paths is higher than a reordering threshold, a flow reordering rule is set up at the receiving edge router. Similarly, if the MDI is above another threshold, the delay difference is considered too high to provide any aggregated capacity advantage. We determine those thresholds in Section IV

C. Packet Scheduler

The common challenge for every multipath protocol is deciding how to send data over the available paths. The task is usually done by a *scheduling algorithm*. This scheduler can rarely work in isolation as it needs to adapt to changing path characteristics, mainly in terms of delays and congestion. There are many approaches to multipath scheduling [20], ranging from simple information agnostic round-robin approaches to omniscient algorithms.

To maximize the performance, a multipath scheduler should push the right amount of data over different paths, without overloading already congested ones and ensuring full utilization of the available capacity. MPTCP uses subflows with independent congestion windows [3], [5] and can buffer some packets before sending them on the desired path [24], [26].

In the case of in-network multipath, however, neither the per-path window information nor the advance buffering option are readily available. To maximize application throughput, we use a Weighted Round-Robin (WRR) scheduler which sends bursts of packets along the paths, weighted according to their capacity as $w_j = c_j / \sum_i^n c_i$, where w_j is the weight associated with path j, and c_j its estimated capacity. While not as fine-grained as layer-4 scheduling, this approach maps well to OpenFlow's Groups approach and our measurements, presented in Section IV, show the performance difference is acceptable.

D. Reordering Mechanism

By selecting multiple paths with potentially different characteristics, our mechanism introduces packet reordering. To avoid a performance impact due to out-of-order packets, we implemented a corrective mechanism that can be deployed on the edge switches.

Layer-4 multipath algorithms (Section II) solve this problem by using out-of-order queues at the receiver, which resequence packets in the desired order prior to passing them to the application.

We introduce a *resequencing buffer* at the receiving edge switch in order to address this problem in a similar fashion, albeit without the receiver node's involvement. The buffer temporarily stores packets received ahead of time. It does so by maintaining a record of the next expected sequence number for each flow, in a similar fashion as TCP, and only forwards packets if the sequence numbers match. This is show in Algorithm 1.

This can cause a problem in case packets are lost prior to reaching the resequencing buffer. To avoid timeouts at the TCP sender, our proposed solution implements dynamic buffer sizes based on a *buffering threshold* T, sized as a function of the MDI and the bandwidths of the selected paths for the flow. If the number of packets buffered for a flow exceeds its threshold, the buffer releases them all in order, ignoring gaps. This may trigger some unnecessary retransmissions, but endpoints supporting SACK should see only minimal impact.

Additionally, to protect against bursts of losses in the network, the next expected sequence is further increased by a loss-recovery factor LRF after a threshold-triggered release. This causes the buffer to forward packets with lower sequence numbers in their order of arrival, ignoring other lost packets of the burst, until the new expected value is reached, thereby ignoring any other missing packets from the loss burst. Experimental tests showed that a value of 20 allowed the buffer to recover from bursty losses while limiting the amount of out-of-order packets during this recovery period.

Algorithm 1 Resequencing for each flow

```
Require: buffer $B$ of size $S$
Require: buffering threshold $T$
Require: loss-recovery factor $LRF$
while $pkt \leftarrow$ receive packet do
  if $pkt$ is SYN then
    $expected \leftarrow pkt.seq + pkt.size$
    forward $pkt$
  else if $pkt.seq < expected$ then
    forward $pkt$ {immediately forward duplicates}
  else if $pkt.seq = expected$ then
    for all $p \in B | p.seq < expected$ do
      forward $p$ {send all delayed packets in order}
    end for
    forward $pkt$
    $expected \leftarrow expected + pkt.size$
  else if $B.use > T$ then
    store $pkt$ in $B$
    for all $p \in B$ do
      forward $p$ {send all packets in order, ignoring gaps}
      $lastp < -p$
    end for
    $expected \leftarrow lastp.seq + lastp.size \cdot LRF$ {account for bursty losses}
  else if $B.use < S$ then
    store $pkt$ in $B$
  else
    $spkt \leftarrow p \in B | p.seq = \min_{p \in B}(p.seq)$
    if $spkt.seq \leq pkt.seq$ then
      send $spkt$ {send the packet with the lowest sequence number}
      store $pkt$ in $B$
    else
      forward $pkt$
    end if
  end if
end while
```

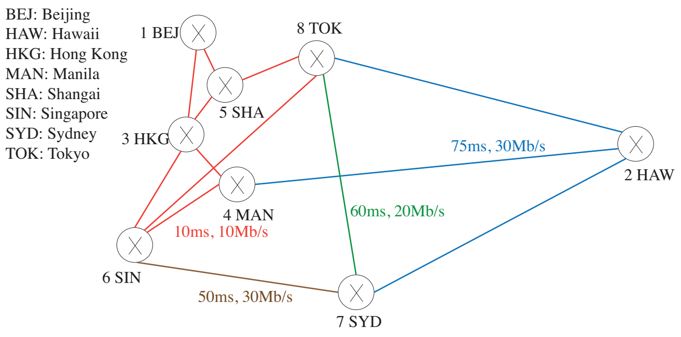


Figure 2: Topology model in our emulation (East Asia Internet Backbone).

E. Implementation Considerations

We implemented the WRR scheduler using the existing *Select group* in Open vSwitch. The resequencing buffer required the addition of a new group in Open vSwitch, as well as a new OpenFlow message to configure it. The code for these modifications is available online[6], as is that of our Ryu-based controller.[7]

While layer-2 path capacity was estimated using port statistics, a dynamic layer-3 equivalent was not fully implemented—the controller currently needs manual configuration of path capacities. We expect the switches could use client traffic to implement methods such as packet dispersion [33]. Such an approach is, however, beyond the scope of this paper.

IV. Performance Evaluation

We evaluated MPSDN using both emulation and large-scale deployment on a multihomed testbed. We first used emulation of an L2 topology to explore the sensitivity of our approach to variations in conditions. We then performed use-case experiments in real-world L3 deployments to confirm the feasibility of our solution. In both cases, we provide comparisons with MPTCP.

All measurements were done using Linux with default TCP parameters. In particular this means that CUBIC was used as the congestion avoidance algorithm for all TCP flows throughout this section.

A. Emulation

We used Mininet [34] to create an L2 topology mirroring the East Asia Internet Backbone,[8], shown in Figure 2. As our setup could not emulate the Gigabit speeds of the backbone, we scaled the capacities down. However, we chose realistic delays between the routers, as estimated by probing their real counterparts with ICMP echo requests.

1) Throughput Measurements: In the following experiments, we used `iperf` 3,[9] `netperfmeter` [35], `netcat` and `d-itg` [36] to generate traffic. We measured flow parameters (`cwnd`, `rtt`) with `ss` and `captcp`.

[6]https://github.com/dariobanfi/ovs-multipath
[7]https://github.com/dariobanfi/multipath-sdn-controller
[8]http://maps.level3.com/default/
[9]http://software.es.net/iperf/

Table II: Evaluation of throughput with MPSDN multipath forwarding.

Path	Capacity	Latency	Throughput
BEJ–SHA–TOK–HAW BEJ–HKG–MAN–HAW	10 Mbit/s 10 Mbit/s	95 ms 95 ms	18.2 Mbit/s
TOK–SIN–SYD TOK–SYD	10 Mbit/s 20 Mbit/s	60 ms 60 ms	26.8 Mbit/s

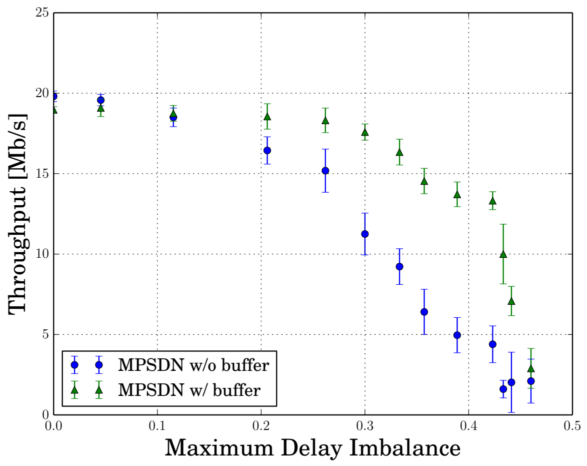


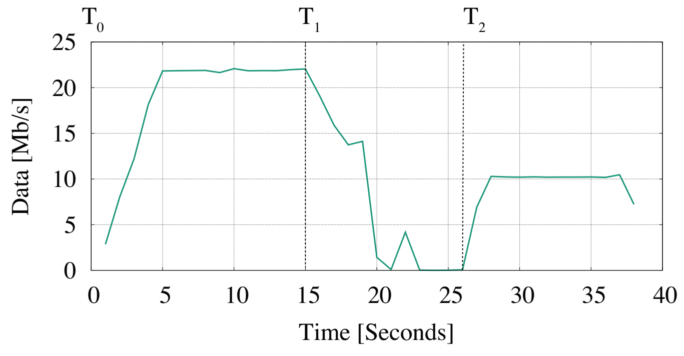
Figure 3: Impact of the maximum delay imbalance MDI on the aggregated throughput, with and without resequencing buffer (base latency 25 ms, 10 Mbit/s paths, 15-seconds `iperf`)

We measured the throughput our solution achieved without cross-traffic. The results are shown in Table II. In this first scenario, the capacity was close to the aggregated bandwidth of the single paths.

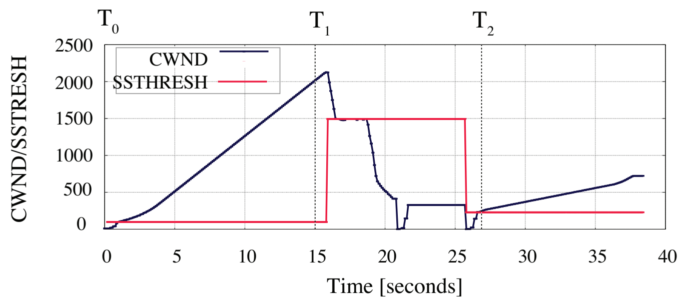
We also measured the throughput achieved with unbalanced latencies. The result of these measurements is shown in Figure 3. These measurements were performed on a simple topology with just two direct paths. One path has a 25 ms latency, the other path's latency is increased to obtain the different MDI values (x-axis). The measurements were done both with and without enabling the resequencing buffer. The effectiveness of the buffer within a certain range of MDI values can be clearly observed.

2) MDI cutoffs: Figure 3 also shows that using the resequencing buffer for MDIs beyond 0.15 improves performance quite vastly, while for path capacities beyond 0.4 the aggregated bandwidth falls below the bandwidth of a single path even when using the resequencing buffer.

3) Intra-flow fairness: We also verified that introducing MPSDN in a network does not have an adverse effect on intra-flow fairness. We started 10 30-second `iperf` transmissions over an MPSDN network and reported the flow throughput every second. We computed Jain's fairness index [37] for each period. Overall, the mean fairness was 0.81 ($\sigma = 0.042$), which



(a) Aggregated throughput



(b) Congestion window

Figure 4: Impact of transient congestion of one of the paths.

we find to be quite good.[10]

4) Impact of path congestion on transport: We then evaluated the resilience of our approach to dynamic congestion, both in terms of path recomputation and transport resilience to changes.

We used the same topology and source/destination as before. At time T_1, we start an `iperf` UDP with a target rate of $1\,\text{Gbit/s}$ to completely saturate the link between Singapore (*SIN*) and Sydney (*SYD*). At T_2, our controller measures the available path capacity and recomputes bucket weights. As one path is completely congested, it switches to forwarding on one path only.

Figure 4a shows the impact of the path congestion on the transport's throughput, where it quickly drops to 0 before our mechanism reconfigures the paths, after which the throughput slowly grows to the new one-path capacity of $10\,\text{Mbit/s}$.

While the controller adequately updated the path selection, the transport is badly impacted during the congested period—even though only one path is congested—and slow to respond after the path recomputation. This is due to the TCP sender only maintaining a single congestion window for all the paths, and reducing it drastically when losses start to occur on the congested path, as shown in Figure 4b.

5) Comparison with MPTCP: MPTCP and MPSDN differ in the requirements they impose on implementing systems: multi-homing in the case of MPTCP, and SDN support with measurement capability for MPSDN. Nonetheless, they share the same objective of capacity aggregation. We therefore compared the goodput achieved by our solution to MPTCP's, in systematic experiments varying the delay on the second path.

We set up a basic topology composed of just two hosts. For MPTCP, the hosts are multi-addressed. For MPSDN, each host has only one IP address, but there are two available paths in the network. For MPTCP, we use two subflows and the default scheduler. The sender starts a 30 seconds transmission; the application-layer goodput is measured at the receiver.

Figure 5 shows the TCP goodput for the single paths and compares it to MPTCP and MPSDN performance. In subfigures (b) and (c), which have a high delay difference ($25\,\text{ms}$ and $50\,\text{ms}$ corresponding to an MDI of 0.17; $25\,\text{ms}$ and $100\,\text{ms}$ corresponding to an MDI of 0.3) the reordering buffer is used.

Our results show that MPSDN performance remains close to that of MPTCP when the paths are balanced (although with a higher variance) and performs slightly worse, but still comparable, when the delay differences are high.

B. Real-world deployment

We now verify that our proposal is usable in real world deployments. The most notable difference is that, instead of an L2 topology, we now consider an L3 network where we only control the edge switches. Apart from quantitative measurements, our goal is also to qualitatively explore the deployability of our solution over the real Internet.

We deployed our MPSDN solution on the NorNet[11] Core testbed, which offers distributed, multihomed, and programmable nodes [39], where static IP tunnels are established to form a full mesh between nodes, and packets are routed based on their source/destination address.

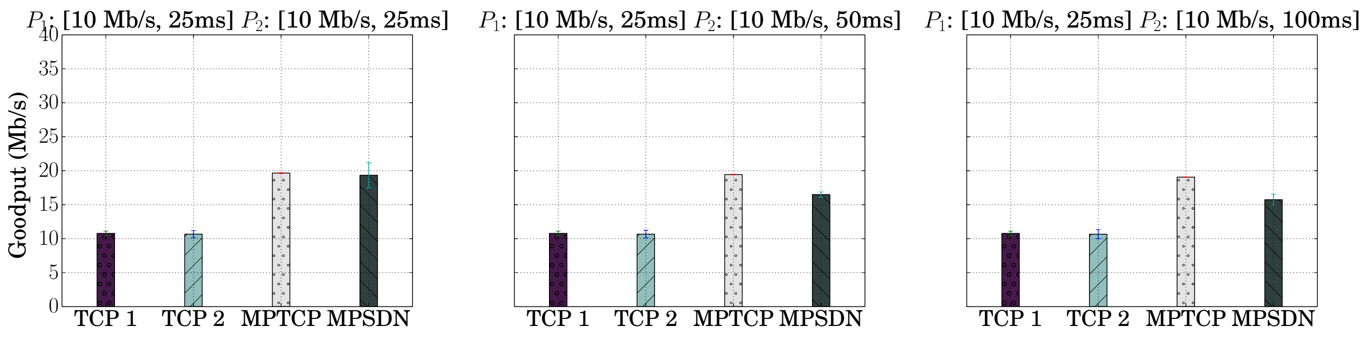
We ran our experiments on Ubuntu 14.04 LTS virtual machines with kernel 3.13.0-68-generic, 1 GB RAM, and 2.60 GHz CPUs. The VMs were multi-addressed and used the aforementioned IP tunnelling. We simply installed our modified Open vSwitch directly on the VMs and used it to route the traffic. This allows the application to create normal TCP connections and keeps the multipath splitting transparent.

We chose the sites at the Simula Research Laboratory near Oslo (NorNet's home) and one in Longyearbyen, just 1300 km from the North pole, in the Svalbard archipelago. The switch on the sender side was configured to rewrite the layer-3 source/destination addresses to trigger the Weighted Round Robin Scheduling and forward packets onto their selected path. The receiving switch performed the reverse address mapping.

1) TCP Goodput: We first tested the scheduling without any reordering buffer between two endpoints with paths of equal capacity to determine how many packets would arrive out-of-order and cause performance degradation. We used two of the multiple paths/ISP combinations between both endpoints, which had at least $10\,\text{Mbit/s}$ of capacity. As discussed in Section III-E, we manually set the weights for both paths in the scheduler. We set them to equal values. Both paths have RTTs around $40\,\text{ms}$.

[10]The best fairness index would be 1, but anything above 0.5 is considered "reasonably fair" [38].

[11]https://www.nntb.no/



(a) $c_{p_2} = 10\,\text{Mbit/s} \quad d_{p_2} = 25\,\text{ms}$ (b) $c_{p_2} = 10\,\text{Mbit/s} \quad d_{p_2} = 50\,\text{ms}$ (c) $c_{p_2} = 10\,\text{Mbit/s} \quad d_{p_2} = 100\,\text{ms}$

Figure 5: Comparison of MPSDN goodput to MPTCP with varying path delays. Error bars show the standard deviation of 10 runs.

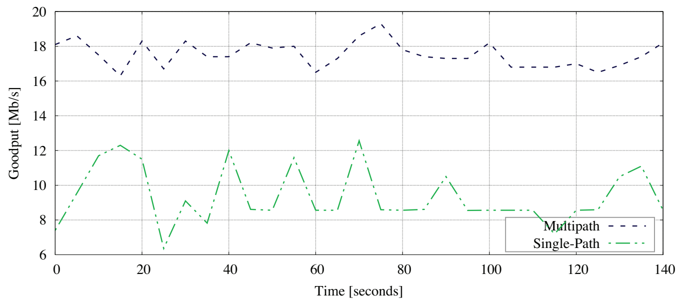


Figure 6: `iperf` throughput: single-path vs. multipath.

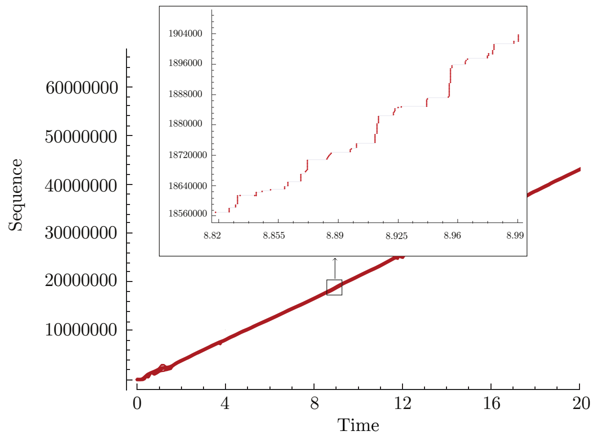


Figure 7: TCP sequence numbers at the receiver.

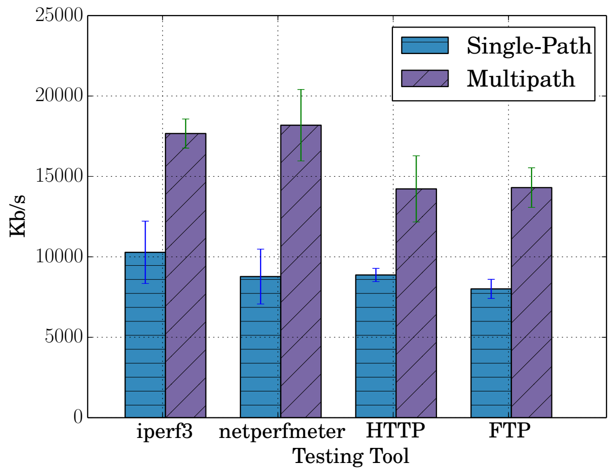


Figure 8: Multipath throughput with different traffic types.

Figure 6 shows the goodput over 140 seconds as measured by `iperf` 3 periodic reports, with the default settings. Multipath forwarding succeeds in aggregating paths capacities, resulting in a roughly doubled throughput, compared to single-path.

As Figure 7 shows, the TCP sequence numbers at the receiving end are growing almost monotonically, showing only very light packet reordering. It is interesting to note the jagged profile of the curve, where bursts of packets arrive at different rates, depending on which paths they had been forwarded on.

2) Application use-cases: We continued with the same configuration, and experimented with a number of different application workloads. We first tested with the same setup, but also launched `netperfmeter` using TCP and the default flow settings, which attempts to maximise the throughput. We then ran two other experiments. We used Python's `SimpleHTTPServer`[12] module and `wget`[13] to simulate the transfer of a 70 MB file over HTTP, and over FTP using `vsftpd`.[14]

Figure 8 shows our results. Multipath forwarding consistently delivered on its promise of capacity aggregation. This is best shown for applications which actively attempt to saturate the network capacity, but all significantly benefit from MPSDN.

V. Discussion and Lessons Learnt

In this section, we reflect on the proposed architecture, influence of the design choices, and resulting performance

[12]https://docs.python.org/2/library/simplehttpserver.html
[13]https://www.gnu.org/software/wget/
[14]https://security.appspot.com/vsftpd.html

and usability of MPSDN.

1) Impact of buffer and MDI: The MDI proved to perform quite well in our experiments. However, all our measurements have a somewhat similar latency (with the faster path usually having a latency of either $10\,\mathrm{ms}$ or $25\,\mathrm{ms}$). We later realised that using a purely relative measurement for the imbalance does not allow the MDI to work equally well for all latencies. It needs to be refined to also incorporate the absolute difference between the latencies.

2) Path selection: In the evaluated version of our proposal, we did not consider a dynamic and continuous estimation of the paths' capacity, but rather focused on the feasibility of our solution in a stable environment. We showed in this context that, in conjunction with the MDI metric, it was possible to identify suitable complementary paths. Nonetheless, as future work, we plan to investigate the possible addition of a measurement mechanism (active or passive) to estimate these capacities. In particular, we aim to determine the trade-off between adding more measurements and accidentally contributing to congestion. It might also be worthwhile to replace a currently-congested path with another, uncongested, path.

3) Impact on Vanilla TCP: Our solution successfully enables transparent capacity aggregation by scheduling bursts of packets on different paths, and prevents spurious retransmissions by reordering datagrams before delivering them to the end node. Nonetheless, TCP's control loop can become disrupted due to transient issues on any single path, leading to performance degradation for the whole transfer.

This is due to the fact that the TCP has no knowledge of the use of multiple paths. Its RTT estimate is that of the longer path and reordering buffer, while its congestion window covers the aggregated capacity. In case one path experiences a spike in delays, or a burst of losses, TCP will react by reducing its sending rate *for the whole transfer*. As a result, only paths of similar characteristics (as determined by metrics such as the MDI) will aggregate well, but the throughput will be very sensitive to the performance of the worst path.

4) Comparison to MPTCP: Even without a reordering buffer, our in-network multipath solution achieves a very good aggregated bandwidth and similar goodputs as MPTCP while not requiring end-host support.

An SDN solution, with its advantages of being network stack-agnostic, can achieve a performance that is similar to that of MPTCP. While MPTCP's challenge is endpoint support, the challenge with MPSDN lies in determining the parameters for path setup and packet reordering.

5) Ease of deployability: Our MPSDN proposal reduces the deployability issues seen with MPTCP. While each end-host needs to be separately enabled to support MPTCP, MPSDN only requires leaf networks to deploy at least one edge switch supporting our extensions to provide multipath connectivity from all hosts on that network to any other MPSDN-enabled network. Some deployment considerations were however not addressed, such as when two MPSDN networks are not under the jurisdiction of the same controller. Access control and delegation in SDN is beyond the scope of this paper, but can be adequately addressed by a broader research agenda in SDN [*e.g.*, 40].

VI. Conclusion

We have presented a solution to enable the use of multiple paths in a layer-2 or -3 topology. The main objective is to use alternate paths in parallel to aggregate capacity and provide higher goodputs. Unlike solutions such as MPTCP or CMT-SCTP, our approach leverages an SDN infrastructure to provide path selection, packet scheduling, and packet reordering in the network, without the need to modify the endpoints. We have evaluated the solution in a range of emulated scenarios and showed that it is able to adequately provide capacity-aggregation benefits that are similar to what MPTCP achieves. We have also demonstrated the deployability of the solution in a real multi-homed scenario over the Internet.

Our work highlighted the need that the various aspects of multipath transfer are addressed in the right layer—path discovery and selection belongs in the network, but the transport needs to be aware of the existence of multiple paths and manage them separately—and a richer communication between those layers to support it. Future work should study how this split can best be achieved. Unfortunately, TCP/IP networks are poorly equipped for a lightweight upgrade that could unlock the full potential of multiple paths.

References

[1] C. Guo, G. Lu, D. Li *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers", *Comput. Commun. Rev.*, vol. 39, no. 4, 2009.

[2] Q. Zhang, L. Cheng and R. Boutaba, "Cloud computing: State-of-the-art and research challenges", *J. Internet. Serv. Appl.*, vol. 1, no. 1, 2010.

[3] A. Ford, C. Raiciu, M. Handley *et al.*, "TCP extensions for multipath operation with multiple addresses", RFC 6824, 2013.

[4] Y. Yuan, Z. Zhang, J. Li *et al.*, "Extension of SCTP for concurrent multi-path transfer with parallel subflows", in *WCNC 2010*.

[5] A. Ford, C. Raiciu, M. Handley *et al.*, "Architectural guidelines for multipath TCP development", RFC 6182, 2011.

[6] M. Al-Fares, A. Loukissas and A. Vahdat, "A scalable, commodity data center network architecture", *Comput. Commun. Rev.*, vol. 38, no. 4, 2008.

[7] O. Mehani, R. Holz, S. Ferlin *et al.*, "An early look at multipath TCP deployment in the wild", in *HotPlanet 2015*.

[8] G. Sarwar, R. Boreli, E. Lochin *et al.*, "Performance evaluation of multipath transport protocol in asymmetric heterogeneous network environment", in *ISCIT 2012*.

[9] D. Banfi, "Endpoint-transparent multipath in software defined networks", Master's thesis, 2016. [Online]. Available: https://www.nicta.com.au/pub?id=9163.

[10] Cisco. (2011). Layer 2 multi-path (L2MP) overview.

[11] H. Frazier, S. Van Doorn, R. Hays *et al.* (2007). IEEE 802.3ad link aggregation (LAG) — what it is, and what it is not.

[12] J. Touch and R. Perlman, "Transparent interconnection of lots of links (TRILL): problem and applicability statement", RFC 5556, 2009.

[13] D. Thaler and C. E. Hopps, "Multipath issues in unicast and multicast Next-Hop selection", RFC 2991, 2000.

[14] IEEE 802.1 Working Group, "Standard for local and metropolitan area networks: Virtual bridged local area networks — amendment 8: Shortest path bridging", Tech. Rep., 2012.

[15] C. Raiciu, S. Barré, C. Pluntke *et al.*, "Improving datacenter performance and robustness with multipath TCP", in *SIGCOMM 2011*.

[16] S. Kandula, D. Katabi, S. Sinha *et al.*, "Dynamic load balancing without packet reordering", *Comput. Commun. Rev.*, vol. 37, no. 2, 2007.

[17] R. Kokku, A. Bohra, S. Ganguly *et al.*, "A multipath background network architecture", in *INFOCOM 2007*.

[18] Z. Yi, T. Saadawi and M. Lee, "Analytical modeling of load balancing SCTP", in *NYMAN 2004*.

[19] J. R. Iyengar, P. D. Amer and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths", *IEEE/ACM T. Network.*, vol. 14, no. 5,

[20] A. Singh, C. Goerg, A. Timm-Giel *et al.*, "Performance comparison of scheduling algorithms for multipath transfer", in *GLOBECOM 2012*.

[21] G. Sarwar, R. Boreli, E. Lochin *et al.*, "Mitigating receiver's buffer blocking by delay aware packet scheduling in multipath data transfer", in *AINA 2013*.

[22] N. Kuhn, E. Lochin, A. Mifdaoui *et al.*, "DAPS: Intelligent delay-aware packet scheduling", in *ICC 2014*.

[23] C. Paasch, S. Ferlin, O. Alay *et al.*, "Experimental evaluation of multipath TCP schedulers", in *SIGCOMM 2014*.

[24] F. Yang, Q. Wang and P. Amer, "Out-of-order transmission for in-order arrival scheduling policy for multipath TCP", in *PAMS 2014*.

[25] T.-A. Le and L. X. Bui, "Forward delay-based packet scheduling algorithm for multipath TCP", Tech. Rep., 2015. arXiv: 1501.03196.pdf.

[26] S. Ferlin, O. Alay, O. Mehani *et al.*, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks", in *IFIP Networking 2016*.

[27] Y. Li and D. Pan, "OpenFlow based load balancing for Fat-Tree networks with multipath support", in *ICC 2013*.

[28] M. Koerner and O. Kao, "Evaluating SDN based rack-to-rack multi-path switching for data-center networks", in *FNC/MobiSPC 2014*.

[29] S. Fang, Y. Yu, C. H. Foh *et al.*, "A loss-free multipathing solution for data center network using software-defined networking approach", in *APMRC 2012*.

[30] T. N. Subedi, K. K. Nguyen and M. Cheriet, "OpenFlow-based in-network layer-2 adaptive multipath aggregation in data centers", *Comput. Commun.*, vol. 61, 2015.

[31] M. Bouet, "Monitoring latency with OpenFlow", in *CNSM 2013*.

[32] N. L. M. van Adrichem, C. Doerr and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow software-defined networks", in *NOMS 2014*.

[33] C. Dovrolis, P. Ramanathan and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology", *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, 2004.

[34] B. Lantz, B. Heller and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks", in *SIGCOMM 2010*.

[35] T. Dreibholz, H. Adhari, M. Becke *et al.*, "NetPerfMeter– a versatile tool for multi-protocol network performance evaluations",

[36] S. Avallone, S. Guadagno, D. Emma *et al.*, "D-ITG distributed internet traffic generator", in *QUEST 2004*.

[37] R. K. Jain, D.-M. W. Chiu and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems", Tech. Rep. DEC-TR-301, 1984. arXiv: cs/9809099.

[38] S. Floyd, M. Handley, J. Padhye *et al.*, "TCP friendly rate control (TFRC): Protocol specification", RFC 5348, 2008.

[39] T. Dreibholz, J. Bjørgeengen and J. Werme, "Maintaining and monitoring the infrastructure of the NORNET CORE testbed for multi-homed systems", in *WAINA 2015*.

[40] I. Baldin, S. Huang and R. Gopidi, "A resource delegation framework for software defined networks", in *HotSDN 2014*.