Endpoint-transparent Multipath Transport with Software-defined Networks

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

* 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 in using multiple paths simultaneously to transport data. While most techniques in this area require endpoint modifications, this article proposes to do it inside the network, transparently to the hosts. This approach, however, is known to introduce packet reordering at the receiving end, which may cause critical performance degradation. We present a Software Defined Network architecture which automatically sets up multipath forwarding including solutions to those issues, 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. A prototype of the model is implemented with wide-spread technologies and evaluated in both emulated and real networks. In this demo, we will show how our solution improves Quality of Experience for high quality video streaming while remaining reactive to network congestion.**

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 get an accurate view of the network topology, while only upper layers are able to control the rate and end-to-end destination.

Nonetheless, solutions have been proposed at various layers to improve specific use-cases, such as layers 2–3 for data-centres with, *e.g.*, BCube [1] or DCell [2], or 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 requirements to modify end-hosts. Heterogeneous network paths also increase 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, 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,[1] 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.

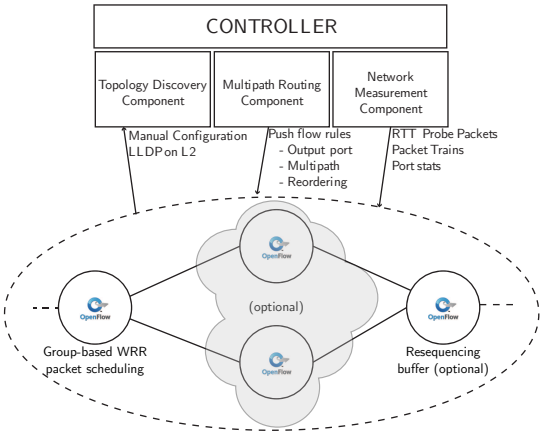

Figure 1: Multipath SDN Architecture

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 our proposal can handle other transport protocol in a similar fashion [9]. This work also allows us to identify some non-trivial issues when implementing layer-4 switching and scheduling with SDN solutions.

The remainder of this paper is organised as follows: We present the proposed architecture and its implementation in Section II and the demo's requirements in Section III.

II. Architecture

Our proposed architecture for an endpoint-transparent multipath network consists of a centralised controller with knowledge of the network topology which dynamically sets up

[1] A more detailed version of our proposal is going to be presented in the regular track of LCN

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.

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 [10], 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[2] and 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 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 [11], which yields high accuracy and has a low network footprint. Unlike NetFlow or measurements using ICMP echo requests, it does not require additional servers or components. The algorithm is run using controller-to-switch messages only. In addition, we use port statistics counters for bandwidth estimation.

B. Path Selection

In order to maximise the aggregated capacity of the 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.

To select compatible paths, we introduce the concept of *maximum delay imbalance*,

$$MDI = \frac{d_{max}}{d_{max} + d_{min}} - 0.5\,, \tag{1}$$

where d_{min} and d_{max} denote the minimal and maximal delays from the candidate paths. Its range 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 by 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.

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

C. Packet Scheduler

To maximise the performance, a multipath scheduler should push the right amount of data over different paths, without overloading already congested ones and ensuring full utilisation 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 [12], [13].

In the case of in-network multipath, however, neither the per-path window information nor the advance buffering option are readily available. To maximise 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.

D. Reordering Mechanism

We propose to introduce a *resequencing buffer* at the receiving edge switch in order to address this problem in a similar fashion, albeit without 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 sequence numbers match.

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*, 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 in order of their sequence number, ignoring gaps. This may trigger some unnecessary retransmissions, but endpoints supporting SACK should only see minimal impact.

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[3] as well as that of our Ryu-based controller.[4]

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 about path capacities. We expect the switches could use client traffic to implement methods such as packet dispersion [14]. Such an approach is, however, beyond the scope of this paper.

III. Demonstration

As detailed above, we have implemented our solution in both Open vSwitch for the various mechanisms and in RYU SDN controller for the control part. In order to demonstrate the benefit of our proposal, we offer a walk-through scenario in

[3]https://github.com/dariobanfi/ovs-multipath

[4]https://github.com/dariobanfi/multipath-sdn-controller

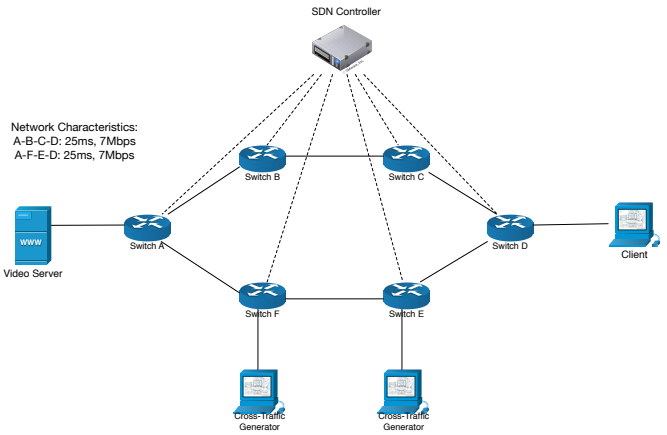

Figure 2: Demonstration Topology

which a high quality video would be streamed in an emulated network and our solution would automatically re-configure the various switches.[5]

1) Scenario: In our demonstration, a server streams HD video (10 Mb/s) over HTTP over a simple multipath topology composed of six switches (two paths) and a client as shown in Figure 2. In this topology, the nominal capacity of each path is set to 7 Mb/s, with a similar one-way latency of 25 ms.

At the beginning of the demo, the network is left un-configured where the video stream would follow a single path, resulting in video stalling and poor performances. After, this first phase, the SDN controller would reconfigure edge switches (A and D in Figure 2) to provide Multipath-SDN. Once configured, we will demonstrate how our solution leverages this addition network capacity to provide good Quality of Experience to the end-user.

Finally, in order to demonstrate the ease of reconfiguration of our solution, UDP cross traffic is generated in a last phase resulting in a congestion on one of the path (between F and E in Figure 2). Upon observation of this congestion event, the SDN controller is automatically re-configuring the network to use single path.

2) Implementation Details and Equipment: To emulate the network topology shown in Figure 2 we would use a single laptop running virtualization software deploying our modified open vSwitch application. The server and client will also run in separate virtual machines.

The exact demonstration requirements in terms of equipment are provided in the list below:

[5]A recorded video of our demo can be found at https://www.youtube.com/watch?v=hkgf7l9Lshw

- Table large enough for 1 laptop and monitor;
- HDMI Monitor/Screen (optional);
- Power supply for 1 laptop, and 1 screen;
- Ethernet cable with Internet access;
- Poster wall;

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] 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.

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

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

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

[14] 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.