Augmented Multipath TCP Communications

Matthieu Coudron, Stefano Secci, Guy Pujolle
LIP6, UPMC, 4 place Jussieu 75005, Paris, France. Email: firstname.lastname@lip6.fr

Abstract—**Cloud networking imposes new requirements in terms of connection resiliency and throughput among virtual machines, hypervisors and users. A promising direction is to resort to multipath communications, yet existing protocols still struggle to take advantage of the path diversity offered by IP networks. Multipath TCP (MPTCP) can create several TCP subflows on different interfaces and concurrently forward data on these subflows. Current MPTCP implementations create a full mesh of subflows between IP endhosts, which may be suboptimal according to the topology. We propose to rely on topology information brought by an external protocol in order to improve the MPTCP subflow management; we resort to the Locator/Identifier Separation Protocol (LISP) to retrieve IP path diversity information, to then accordingly create MPTCP subflows. We report noticeable benefits obtained using a large-scale Cloud access test bed, and we describe further work we are conducting in this sense.**

I. Introduction

From a physical network perspective, it is easy to observe the existence of a large number of paths between two end-points, by various measurement techniques. In most countries, the competition between telecommunication companies lead to the deployment of different wireline physical networks till the home: xDSL, optical fiber links, etc.. Data center networks also evolve adopting topologies with increased path diversity such as Fat-Tree [1] or VL2 [2]. On the other hand, mobile users can also rely on multiple wireless interfaces with smartphones and tablets (3G, 4G, Wifi, etc). In this context, we aim at increasing the throughput offered to applications by a light and scalable extension to the Multipath TCP protocol.

By informing MPTCP subflow management system of the available path diversity, and relying on a network infrastructure "programmable" to some extent, MPTCP subflows can be created and then routed via multiple disjoint paths. This does not need to be done pervasively, i.e., these paths do not need to be end-to-end completely disjoint, as long as sharing a link does not imply sharing a bottleneck. Once these subflows are created, the MPTCP congestion control system is in charge of spreading the load over the different subflows.

MPTCP implementations (see [3]) estimate by default the path diversity by counting the number of local and remote interfaces. This might lead to situation with more subflows than needed, or as in the example in Fig. 1 less subflows than needed. In this example, MPTCP implementations create one subflow where in fact they could use the WAN path diversity (considering there is no LAN bottleneck). In this case, in a Locator/Identifier Separation Protocol (LISP) network, MPTCP could create two subflows instead of one relying on the LISP router to attribute one subflow to each WAN path.

After a brief overview, we synthetically present a solution showing we can achieve a 100% increase in the throughput by adding a single subflow for the real scenario depicted in Fig. 1. Then we present further research on the topic.

II. Multipath TCP (MPTCP)

Multipath TCP (MPTCP) is a TCP extension [4] thought to make the best out of end-to-end path diversity. It basically tries to establish several TCP subflows - under the supervision of the main MPTCP connection - between remote and local endpoints. MPTCP then concurrently forwards data over those subflows, which is likely to increase the throughput, under appropriate congestion control. These subflows can differ by their local/remote IP addresses, or even by their port numbers, thus allowing the creation of several subflows on a single interface. MPTCP conception emphasizes backward compatibility with current networks. For instance, if an endpoint tries to establish an MPTCP connection with a non-MPTCP compliant remote host, then the communication will fall back to legacy TCP. Many mechanisms were also devised to cope with middleboxes fiddling (such as IP replacement in the NAT case, sequence number offsets in firewalls etc...).

III. The Loc/Id Separation Protocol (LISP)

IP addresses usually assume localization and identification functions, hence they need to be distributed according to the topology to allow efficient routing, which hinders the operational distribution of IP addresses. LISP solves this "conflict" between the functional and operational needs, by assigning one namespace to each function: an endhost owns an IP, which differs from the IP used for routing in the Internet transit segment. This is achieved by tunneling packets between remote and local networks. When an endhost sends a packet to a remote site, this packet passes through the border router, which is a LISP router. This LISP router requests from a mapping system (similar to DNS) the egress LISP router(s) to tunnel the packets to, in order to reach the remote endhost. It then encapsulates the original packet into a specific UDP packet and forwards to the Egress Tunnel Router (ETR). This ETR decapsulates the UDP packet and forwards the inner IP packet to its original destination. As MPTCP, LISP can balance the load over different paths, and provides backward compatibility via the deployment of Proxy Ingress Tunnel Routers and Proxy ETRs.

IV. Proposed solution

In order to achieve the MPTCP-LISP coupling, we modified a linux kernel MPTCP implementation [3] so that it queries a LISP router how to manage subflows on every new MPTCP

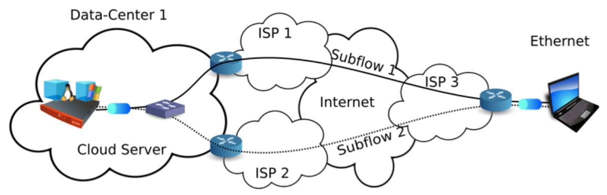

Fig. 1. Our Cloud access testbed scenario

connection via a Linux kernel module [7]; we also had to slightly modify an open source LISP router (LISPmob [5]) in order to accept these requests and perform load-balancing. Our code is open source [7].

In a nutshell, we propose that when a LISP router receives an end-point-identifier mapping request from the MPTCP endpoint, it responses with a mapping reply with the necessary information to determine the number of WAN paths towards the requested destination. Upon reception, MPTCP creates the corresponding number of subflows and attributes consecutive (modulo the number of WAN links) source port number to the subflows, so that our LISP router can achieve optimal load-balancing by looking at the TCP source port number. In fact, the LISP router forwards the subflow on the WAN link number (TCP source port number) % (number of LISP routing locators towards the packet destination). In the current implementation, the endpoint is supposed to be aware of the load-balancing capabilities of the LISP router.

V. Evaluation

We tested our solution using a server in a data center of the nu@ge project (www.nuage-france.fr) multihomed via two different Internet service providers, with a single-homed client, as in Figure 1. Both hosts are MPTCP compliant and behind a LISP router connected to the LISP Beta Network testbed (www.lisp4.net).

We made the client download 40 files from the server, each file being bigger than the previous one by an increment of 256 kb. We recorded five transfer completion times for each file, and repeated the whole procedure using three different configuration cases: (i) legacy TCP, i.e., no cooperation with LISP, single TCP flow; (ii) MPTCP only, i.e., no cooperation with LISP, and a single MPTCP subflow; (iii) LISP-MPTCP, crosslayer cooperation with as many subflows as the product of remote and local RLOCs, i.e., two subflows.

On Figure 2, we see that unassisted MPTCP (marked ‘MPTCP’) and TCP transfer times are quite close, MPTCP being a little slower than TCP; the cause should be the additional 20 bytes overhead per packet introduced by MPTCP. More importantly, we can see that our solution (marked ‘LISP+MPTCP’) performs significantly better. For almost all file sizes, we get a gain close to 100%, i.e., with the additional LISP-enabled subflow we can halve the transfer time. This very clearly shows the important benefit we can reach with the MPTCP-LISP crosslayer cooperation module we propose.

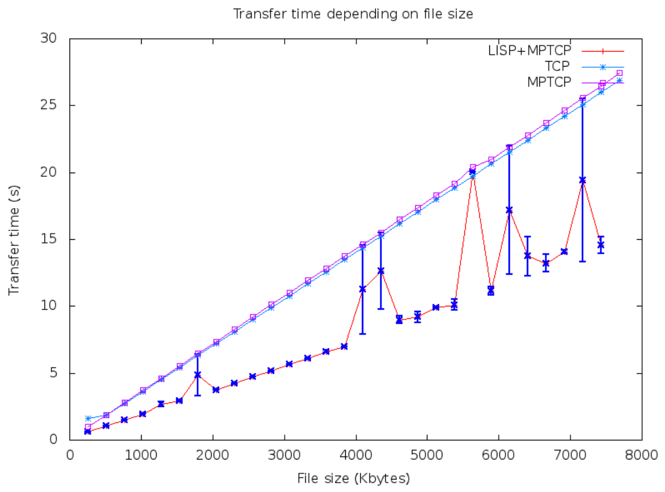

Fig. 2. Completion times for different file sizes

VI. Perspectives and Future Work

We obtain therefore promising results, yielding to an augmented MPTCP communication experience thanks to a lightweight informed IP path diversity distribution. These results can be further improved with extensions we aim to work on. Apart adding more subflows thanks to additional network multihoming’s path diversity, we can further augment the performance and reinforce the protocol architecture by leveraging on the LISP Traffic Engineering (LISP-TE [6]) capabilities to (i) ensure that subflows follow different paths in a different way instead of configuring the LISP router to balance the load according to TCP source ports; (ii) stitch additional subflows to explicite WAN overlay paths created across LISP-TE tunnelling routers. Indeed, LISP-TE allows explicit IP overlay routing on the top of LISP routers by building LISP encapsulation chains. Furthermore, subflow addition can also be supported at the LAN/DC/L2 level; an adequate integration with networks based on OpenFlow, TRILL or SPB can further augment MPTCP, as long as they can provide topology information to MPTCP. Finally, the resulting gain in path diversity management can open the way to MPTCP-based network coding techniques for challenging conditions (i.e., wireless, faulty or lossy network environments).

References

[1] M. Al-fares, A. Loukissas, A. Vadat , “A scalable Commodity Data Center Network Architecture ”, *M*ay 2002.
[2] A. Greenberg, et al. “VL2: A Scalable and Flexible Data Center Network Architecture ”, *M*ay 2002.
[3] C. Raiciu et al., “How Hard Can It Be ? Designing and Implementing a Deployable Multipath TCP ”, in *Proc. of USENIX NSDI’12*.
[4] A. Ford et al., “TCP Extensions for Multipath Operation with Multiple Addresses”, RFC 6824, Jan. 2013.
[5] “LISPmob open source LISP node ”(website): http://www.lispmob.org
[6] D. Farinacci, P. Lahiri, M. Kowal, “LISP Traffic Engineering Use-Cases ”, *draft-farinacci-lisp-te-02*, Jan. 2013.
[7] MPTCP & LISPMOB open source patches: https://github.com/teto