Cross-layer Cooperation to Boost Multipath TCP Performance in Cloud Networks

Matthieu Coudron, Stefano Secci, Guy Pujolle, Patrick Raad†, Pascal Gallard†
LIP6, UPMC, 4 place Jussieu 75005, Paris, France. Email: firstname.lastname@lip6.fr
†Non Stop Systems (NSS), 27 rue de la Maison Rouge, 77185 Lognes, France. Email: {praad, pgallard}@nss.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 exploit multipath communications, yet existing protocols have a so limited scope that performance improvements are often unreachable. Generally, multipathing adds signaling overhead and in certain conditions may in fact decrease throughput due to packet arrival disorder. At the transport layer, the most promising protocol is Multipath TCP (MPTCP), a backward compatible TCP extension allowing to balance the load on several TCP subflows, ideally following different physical paths, to maximize connection throughput. Current implementations create a full mesh between hosts IPs, which can be suboptimal. For situation when at least one end-point network is multi-homed, we propose to enhance its subflow creation mechanism so that MPTCP creates an adequate number of subflows considering the underlying path diversity offered by an IP-in-IP mapping protocol, the Location/Identifier Separation Protocol (LISP). We defined and implemented a cross-layer cooperation module between MPTCP and LISP, leading to an improved version of MPTCP we name Augmented MPTCP (A-MPTCP). We evaluated A-MPTCP for a realistic Cloud access use-case scenario involving one multi-homed data-center. Results from a large-scale test bed show us that A-MPTCP can halve the transfer times with the simple addition of one additional LISP-enabled MPTCP subflow, hence showing promising performance for Cloud communications between multi-homed users and multi-homed data-centers.**

I. Introduction

In Internet-working research, multipath communication capabilities are becoming an increasingly popular subject as Wide Area Network (WAN) multipath protocol features and implementations become more and more tangible. Indeed, to reap the benefit of multipath forwarding - that is in our perspective to gain in resilience and throughput - one needs at some point to have different data flows following physically disjoint paths.

From a physical network perspective, the competition between telecommunication operators resulted in the construction of parallel physical networks till the home, be it wired (xDSL, optical fiber links) or wireless (3G, 4G, etc...). At the same time, enduser laptops and cellphones got equipped with concurrent interfaces such as Wi-Fi, Bluetooth, 4G. On the server side, data-centers tend to connect to the Internet through several ISPs to improve their resiliency, i.e. data-centers are more often multihomed. In the end, networks may well provide physically disjoint paths between the servers and its endusers, both ends still need to be able to use these different paths concurrently to improve the performance. The exploitation by the transport layer of this path diversity available end-to-end at the IP layer is actually not easily attainable. What is left to use these paths simultaneously is an adequate protocol stack, flexible and compatible enough with the existing environment, able to stitch transport-layer multipath capabilities to different IP-layer paths.

A significant amount of work has already been done into devising the Stream Control Transmission Protocol (SCTP) [1], as transport-layer multipath protocol. Yet its later equivalent Multipath TCP (MPTCP) [2] is more retro-compatible and hence is more likely to be largely deployed (there are already many MPTCP servers connected to the Internet). Indeed MPTCP is compatible with TCP, both at the header and at the congestion mechanism levels; more precisely:

- if one of the endhost is not MPTCP compliant or if there are middleboxes preventing MPTCP use, then the TCP connection falls back to legacy TCP;
- an MPTCP connection can coexist with other TCP connections, i.e., the MPTCP connection does not get more capacity than the TCP connections.

The congestion mechanism objectives set in the specifications also state that MPTCP must get at least as much capacity its TCP equivalent would get. Thus, the congestion mechanism and the subflow management system appear tightly intertwined; when to create new subflows or to delete them become critical questions. The MPTCP stack has a poor knowledge of the underlying network to adopt efficient decisions, but lower layers may share useful knowledge to support MPTCP for an efficient subflow selection. For instance, creating several subflows for a data-center communication with a layer 2 bottleneck, e.g., due to a shared spanning tree segment, might prove counterproductive in terms of processing overhead vs increase in throughput; similarly, an Internet Cloud-access multipath communication with a server in a data-center through a single external path (single provider) could offer no performance gain.

The purpose of our study concerns the establishment of additional subflows between MPTCP servers and users in a Cloud network where, at least at one side, path diversity is available at the IP layer, but not usable with native protocols. More precisely, we propose a specific cross-layer signaling to allow a MPTCP endpoint to profit from path diversity offered by a Locator/Identifier Separation Protocol (LISP) [4] network.

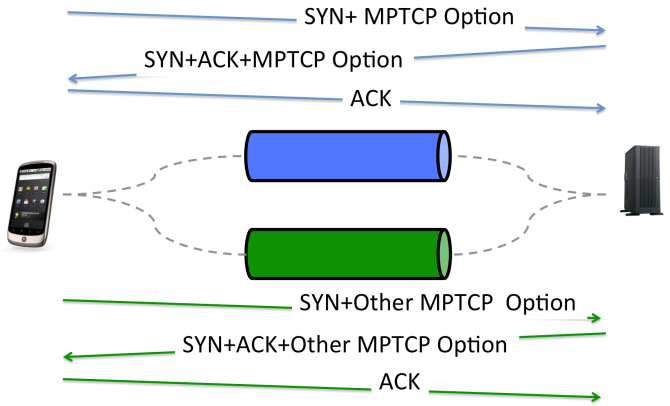

Fig. 1. Simplified representation of MPTCP signaling. Source: [5]

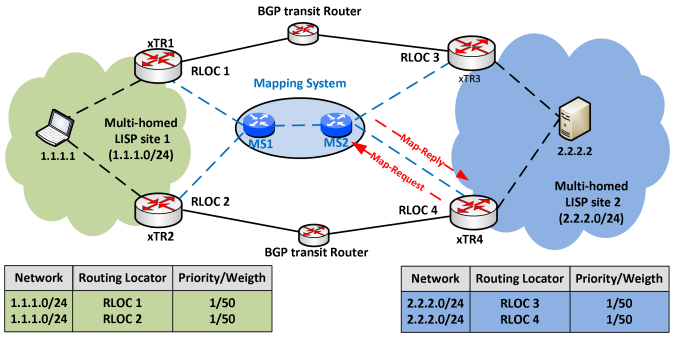

Fig. 2. LISP communications example

LISP can give hints to MPTCP about the WAN paths diversity between two MPTCP endpoints. By allowing sharing of such an information among endpoints, we influence the number of subflows to create. Our proposition leads to an improved version of MPTCP we name Augmented MPTCP (A-MPTCP). A preliminary version of this work was presented as a poster at IEEE ICNP 2013 [3].

The paper is organized as follows. In Section II we present the MPTCP and LISP protocols. Our cross-layer cooperation proposal is described in Section III. Then we present the experimentation framework and the simulation results in Section IV. Section V concludes the paper drawing future work in the area.

II. Presentation of MPTCP and LISP

We describe in this section the MPTCP and LISP protocols highlighting the basic features our proposal relies on.

A. Multipath TCP

As already mentioned in the introduction, MPTCP is a TCP extension enabling end-to-end multipath communications, with an emphasis on backward compatibility, leveraging on multiple connection subflows concurrently used for connection data forwarding. Without entering excessively into technical details, the MPTCP signaling relies on the use of one TCP option in the three-way handshake, and of another TCP option during the connection to open and close subflows as a function of connection, subflow and network states, as depicted in Figure 1.

As explained in [5], nowadays, many middleboxes (e.g., TCP Optimizers, firewalls, PAT/NAT) are able to inspect packets either to modify their content (e.g., change sequence number) or to drop them (e.g., if it detects unknown protocols), and thus can hinder TCP extensions deployement [2]. As a consequence, MPTCP refers to IPs with an identifier rather than by the IP in order to prevent the confusion induced by Address Translators (PAT/NAT). If any problem of the kind is detected by MPTCP, it falls back to legacy TCP, as it is the case if the remote endhost is not MPTCP compliant. Once an MPTCP connection is established, endhosts can advertise their IPs, add or remove MPTCP subflows anytime. These subflows, which we could define as TCP connections children of a more comprehensive parent TCP connection, can be used to achieve greater throughput or resiliency (indeed with the "backup" flag, MPTCP can create a subflow used only in case other subflows stop transmitting). It is worth noting that a host can create/advertise subflows with a same IP address, but with a different port number. The main challenge of such a protocol is the congestion control mechanism. It should not be more aggressive than MPTCP, but at the same time it should use the unused capacity; it should balance the load over different paths, but without causing too much packet disordering so that TCP buffering can reorder them. Adequate path discovery is part of the solution and that is where LISP can help.

B. Locator/Identifier Separation Protocol (LISP)

IP addresses assume today two functions: localization and identification of its owner, which induces a few problems one of which is scalability. IP addresses need to be distributed according to the network topology, which conflicts with ease of use. Provider independent addresses as well as the Internet growth tend to increase global Border Gateway Protocols forwarding information databases, thus slowing lookups and increasing device costs.

The Locator/Identifier Separation Protocol (LISP) [4] splits up localization and identification functions into two IP addresses. In this way, only the localization IP addresses, named Routing LOCators (RLOCs), depend on the topology; RLOCs typically belong to border routers of the endpoint network. The IP address used to identify an endpoint is named Endpoint Identifier (EID). LISP is a network-level solution, i.e., there is no need to change the endhost configuration in a LISP network. Upon reception of a packet from the local network to an outer EID, the border router acts as an Ingress Tunnel Router (ITR): it retrieves the EID-to-RLOC mapping from a mapping system (similar to DNS), then it prepends to the packet a LISP header and an outer IP header with the destination RLOC as destination IP address. As the outer packet is a traditional IP packet, it can be routed on the legacy Internet. One should pay attention to the Maximum Transfer Unit (MTU) though, since the LISP encapsulation adds a 36 bytes overhead in IPv4 (56 bytes with IPv6), i.e., 8 bytes for

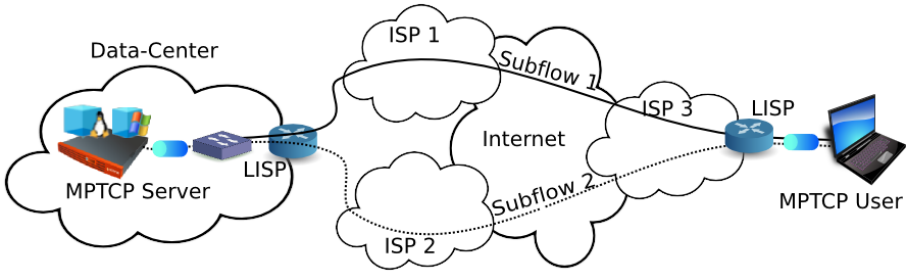

Fig. 3. Example of a multihomed data center and associated MPTCP subflows

the LISP header, 8 bytes for the UDP header and 20 (IPv4) or 40 (IPv6) bytes for the outer IP header. The packet should reach at some point its destination RLOC acting as an Egress Tunnel Router (ETR), i.e., it decapsulates the prepended LISP header and forwards the inner packet to the destination EID.

Consider the example in Figure 2: the traffic sent to the 2.2.2.2 host is encapsulated by the source's ITR toward one of the two destination's RLOCs. The one with the best (lowest) priority metric is selected, which at reception acts as ETR and decapsulates the packet, before sending it to the destination. On the way back to 1.1.1.1, RLOC4 queries a mapping system and gets two RLOCs with equal priorities, hence performs load-balancing as suggested by the weight metric (RLOC1 is selected in the example's packet). In order to guarantee EID reachability, LISP uses a mapping system that includes a Map Resolver (MR) and a Map Server (MS). Typically merged as a single MS/MR node as depicted in Figure 2, a Map Resolver holds a mapping database, accepts MAP-REQUESTs from xTRs and handles EID-to-RLOC lookups. A Map Server receives MAP-REGISTERs from ITRs and registers EID-to-RLOC in the mapping database. In practice, many MRs are geographically distributed and relay MAP-REQUEST messages using a specific protocol called LISP Delegated Database Tree (LISP-DDT) protocol. This is the case of the LISP Beta Network testbed[1].

As a backward compatibility feature, if a source (or destination) site is not yet LISP compliant, the traffic might get encapsulated (or decapsulated) by a Proxy ITR (or ETR). Furthermore, the usage of RLOC priorities and weights in the mapping system allows inbound traffic engineering, suggesting a best RLOC or an explicit load-balancing. In short, LISP routers are border nodes that advertise their RLOCs so that a remote LISP site knows it needs to tunnel packets to those routers to reach hosts in that remote site (i.e. an EID). LISP allows IP mobility (i.e., a machine keeps its EID but updates its RLOCs, for both mobile users and mobile virtual machines [6]), and also give hints if the site is multihomed or not, i.e., if the site is reachable by different providers. In the case the RLOCs of a site belong to different prefixes, they could belong to different providers, and the site is said to be multihomed. In this case, it would be interesting to have additional MPTCP subflows, whose path is splitted at the LISP site border, since they could follow different Internet paths, if not end-to-end, at least along a segment of the Internet path.

[1]LISP Beta Network testbed (website): http://www.lisp4.net

III. Proposed Cross-Layer MPTCP-LISP Cooperation

The current MPTCP path discovery does not explicitly limit the number of subflows to create, so current implementations create by default a mesh of subflows between two hosts' IPs. Most of the times, the more the subflows, the higher connection throughput, under appropriate congestion control. (note that there are also cases in which this default mechanism would prevent MPTCP from increasing the throughput, and cases where fewer subflows could provide the same gain by using fewer network resources). We target the specific case where more subflows could be created if intermediate multipath forwarding nodes (at the border between the local site and the WAN) can be used to split subflows over different WAN paths, under the hypothesis that the connection throughput is limited by WAN capacity rather than by LAN capacity.

In the following, we describe how the MPTCP path discovery can be augmented in this sense in a LISP network. Then we present the required signaling between MPTCP and LISP network elements, and the possible methods to perform the required multipath subflow forwarding.

A. Augmented MPTCP path discovery

The MTPCP specification – see the path management section in [2] – states that the path discovery mechanism should remain modular so that it can be changed with "no significant changes to the other functional components". We leverage on this specific part of the MPTCP protocol to define an augmented subflow discovery module taking profit of LISP capabilities, to augment MPTCP performance while preserving endhost resources.

Figure 3 presents a Cloud access situation where MPTCP could perform better if its path discovery module were informed of LISP path diversity and forwarding capability when creating subflows. In the example situation, there is one IP on the client host and one IP on the server; as such, the legacy MPTCP mechanism creates only one subflow. Under the assumption that commonly connection throughput is limited by WAN capacity rather than by LAN capacity, limiting to a single subflow prevents from benefiting of the WAN

path diversity and the likely greater throughput achievable if forms of multipath forwarding at intermediate nodes exist. A LISP network offers the signaling capabilities to retrieve path diversity information, and the switching mechanisms to ensure multipath forwarding. In the Fig. 3 example, this is possible establishing two subflows instead of one, assuming each subflow is forwarded to a different IP transit path (guaranteed as explained next) thanks to the LISP-capable border router. It is worth highlighting that as of the specifications - and as implemented in the MPTCP Linux implementation [9] - different MPTCP subflows can share the same source IP provided that they use different TCP ports. This subflow identification mode should be used to create the additional LISP-enabled subflows.

More generally than the Fig. 3 example, the number of MPTCP subflows can be significantly increased thanks to LISP capabilities in the case the endpoints dispose of multiple interfaces. We assume communications between hosts are strongly asymmetric, the largest volume being from server to client, so that the client-to-server flow essentially consists in acknowledgments. Let l_1 and l_2 be the number of interfaces of server endpoint (or of the hosting hypervisor in case of VM server) and client endpoints, respectively. A LISP site can be served by one or many LISP routers, each disposing of one or many RLOCs. Let L_1^r and L_2^r be the number of locators of the r^{th} LISP border router at site 1 (Server side) and 2 (Client side), respectively. Therefore, the maximum number of subflows that can be opened by legacy MPTCP is $l_1 l_2$. Following the design choice to route only one subflow over one RLOC-to-RLOC inter-site path to avoid bottlenecks and other management issue in the WAN segment, the number of maximum number of subflows that could be opened thanks to LISP multipath forwarding is $(\sum_r L_1^r)(\sum_r L_2^r)$. Then we can distinguish two cases:

1) if the LISP site network allows full reachability between endpoints and corresponding ITRs, and endpoint's traffic reaches any corresponding ITR in a non deterministic way, then the maximum number of subflows that shall be opened is:

$$N_a = \max\{l_1 l_2; (\sum_r L_1^r)(\sum_r L_2^r)\}. \quad (1)$$

LISP-based augmentation would likely be effective only if the second term is higher than the first. The non deterministic interconnection between the endpoint and its ITRs can be due, for instance, to adaptive L1/L2/L3 traffic engineering changing routes and egress points from time to time even for a same application flow, or load-balancing such as equal-cost multipath L2/L3 routing so that different subflows may exit by different ITRs. It is worth highlighting that (1) is only a suggested upper bound; in the case the right term is the maximum, in such non deterministic situations there is no guarantee the N_a WAN paths will be used, especially if $l_1 < \sum_r L_1^r$ or $l_2 < \sum_r L_2^r$.

2) if each of the endpoint's interfaces can have its traffic routed via one single RLOC each, then we can take a much larger profit from the LISP path diversity. The best situation is when $l_1 \geq \sum_r L_1^r$ and $l_2 \geq \sum_r L_2^r$. So, if the number of the interfaces is at least equal to the number of the RLOCs, then the desired feature is to explicitly link each interface to the right ITR. Such a setting is the de-facto setting when the user is a LISP mobile node MPTCP-capable user; for the Cloud server side, it is technically relatively easy to create a number of virtual interfaces and bind them to different ITRs via virtual LAN mechanisms and the like. In such configurations, the maximum number of subflows that shall be opened is:

$$N_b = (\sum_r L_1^r)(\sum_r L_2^r) \quad (2)$$

The situations described in the previous points can hold only at one site or even only partially within each site. (1) and (2) are indeed upper bounds. Nevertheless, it does not hurt creating a number of subflows higher than the number of paths than N_a or N_b, at the expense of a higher processing and signaling burden on the network, with a probably limited and in any case non-deterministic gain. Therefore, (1) or (2) can be used to compute the number of subflows for the MPTCP-LISP augmentation, depending on the situation, upon appropriate customization of the path discovery module settings.

B. Signaling requirements and implementation aspects

From a mere signaling perspective, the requirement is therefore that the MPTCP endpoint be able to query the LISP mapping system to get the RLOCs of the other endpoint (the number of interfaces being already transported via MPTCP options). Standard LISP control-plane messages can natively assume this task (i.e., MAP-REQUEST and MAP-REPLY messages). Once this information is obtained, the number of subflows to be opened can be computed, and the MPTCP endpoint can either advertise these possible subflows to the other endpoint or initiate the required subflows following a procedure such as the one we propose hereafter.

Let us describe in higher detail the different steps, depicted in Figure 4, needed by the MPTCP host to retrieve the adequate number of local and remote RLOCs, allowing to compute the number of subflows to create. In our implementation, these communications are managed by a specific module at the kernel level. In the case of servers hosted in VMs, the MPTCP logic and our path discovery module could be implemented via an MPTCP proxy (see [8]) at the hypervisor level. In the following, we also give necessary hints on implementation aspects related to each step.

1) endpoint C (Client) first establishes an MPTCP connection with endpoint S (Server). Once endpoint S is assessed as MPTCP compliant, the path discovery process can start.
2) The kernel of endpoint C calls a function of the previously loaded MPTCP kernel module, with as parameters the current MPTCP connection identifier and the EID we want to find the RLOCs associated to.

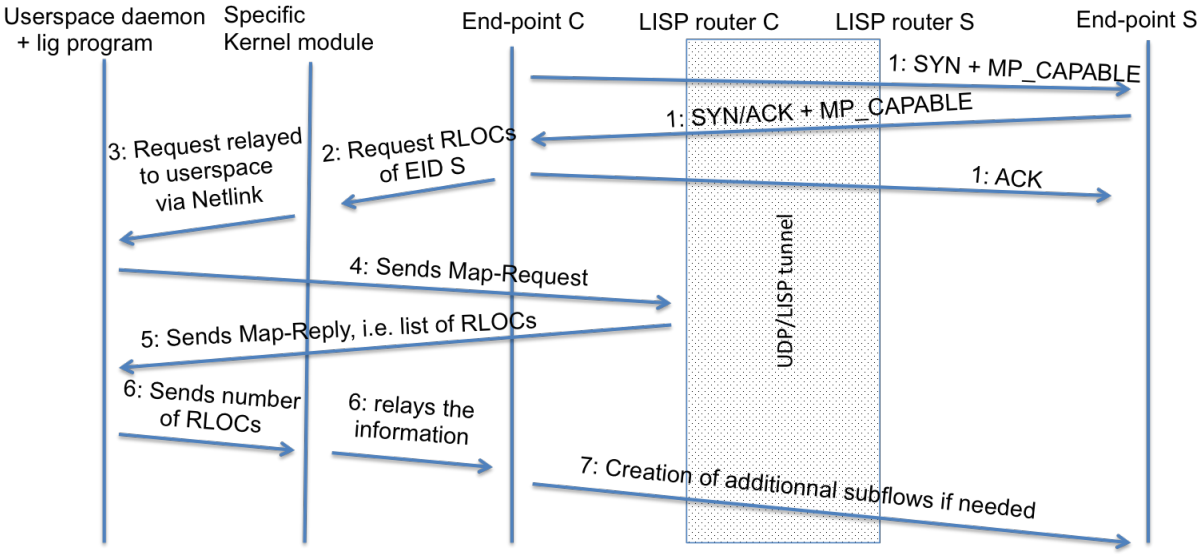

Fig. 4. Signaling process chronology

3) Upon reception, the kernel module translates it into a Netlink message and sends it to the MPTCP Netlink multicast group (Netlink is the networking protocol used by the linux kernel space to communicate with user space and vice-versa).
4) A specific daemon in the user space is registered with the MPTCP netlink group. Upon reception of the Netlink message, it sends a MAP-REQUEST to the LISP router C, i.e., the daemon asks for the RLOCs responsible for the EID (i.e., the IP of Server S). Note that normally one should send a MAP-REQUEST to a Map Resolver, which in existing vendors implementations can be colocated in a LISP router; in our implementation, we modified an open source LISP router for this purpose. Also note that an ad-hoc DC network controller could do this job too.
5) The LISP router C queries its configured map-resolver if the answer is not in its cache. It then sends to that user space daemon a MAP-REPLY listing RLOCs responsible for the requested EID, thus retrieving the number of RLOCs.
6) Upon reception of the MAP-REPLY, the userspace daemon forwards the answer via Netlink to the kernel module.
7) Finally the module relays the information to the kernel, triggering the creation of new TCP subflows.

It is worth mentioning that other strategies could be adopted according to the level of control we have on the different nodes. In case only the DC-side is under control, the server could advertise the same IP until the client creates subflows with adequate source ports. Or it could accept enough communications to ensure that some of them will go through different paths; the MPTCP congestion mechanism would then send data only on the best subflows, which will likely follow different paths.

Furthermore, it is worth noting that the described process is expected to be efficient only for long-lived flows. Indeed, on the one hand short flows may finish before the procedure; on the other hand, the different requests sent from the hosts to the routers consume bandwidth and add an avoidable load to the router. Caching associations between endpoint identifiers and the number of subflows to create mappings in the source host could alleviate signaling traffic. To avoid applying our proposed cross-layer interaction to any flow, a system that triggers the procedure only for long-lived flows would be very reasonable, for example based on a whitelist or a dynamic recognition system such as the one described in [24]. In the case of servers hosted in a VM, this could be implemented at the hypervisor level.

C. LISP multipath forwarding requirements

In order to ensure that subflows indeed follow different WAN paths, it is possible to implement both stateful and stateless load balancing mechanisms.

- A possible stateful solution would be to resort to the LISP-Traffic Engineering (LISP-TE) feature described in [10]. This feature allows enforcing an "Explicit Locator Path" via the LISP control-plane, i.e., with little or no impact on data-plane packet crafting. For a given EID, the mapping systems can build a LISP overlay using intermediate LISP routers between the ITR and the ETR, as re-encapsulating tunnel routers (RTRs), which decapsulate the incoming LISP packet and re-encapsulate it toward the next RTR (or the ETR) in a xTR path. In the context of all subflows being addressed to the same destination EID, the usage of options in the packet header (e.g., using the LISP Canonical Address Format, LCAF, features [11]) can allow routing concurrently multiple subflows via different xTR paths. It is worth mentioning

that LISP-TE extensions are not yet implemented in vendor routers (e.g., Cisco), and in the control-plane of open source routers. However, in the case LISP is implemented at the hypervisor level, hence without touching the source host (VM), the LISP-TE encapsulation with LCAF options can be defined as a switching rule in virtual switches such as OpenVSwitch – knowing that OpenVSwitch already implements basic LISP encapsulation as switching rule since a few months[2].

- A stateless solution could be to carefully choose IP or TCP header fields so that IP packets follow a foreseen IP path. The computation of these fields (for instance the TTL value in IP packets or source port in TCP packets) require the knowledge of both the topology and the load-balancing algorithms running in the nodes to be able to correctly choose the path. Nodes usually use an hash function over IP and some TCP fields to compute a unique flow identifier that decides the egress interface. Hashing functions, by construction, make it hard to foresee the egress interface of a packet. On the contrary, invertible functions can prove helpful as explained in [12].

In our implementation detailed hereafter, we opted for the second option as the first is not yet available in any LISP implementation to date. At the last step in the previous subsection, subflows source ports are chosen so that their port number modulo the number of disjoint paths are all different. The LISP router can then deterministically route each subflow to a specific path. For instance, in the case of 2 paths, we need to have one subflow whose (source + destination port number) % 2 equals 0 and the second subflow (source + destination port number) % 2 equals 1. This mechanism could be replaced with a more scalable solution as those described in [12], yet it was easier to implement. The load balancing is fully effective when installed on both remote and local sites (as in the experiment). Still, in case communications are asymmetric, it can perform well even if installed only on the side sending data (in this case the server side).

The stateless approach is not limited to L3 routing; it can also apply to L2 routing protocols disposing of additional fields (e.g., the hop-count) such as the Transparent Interconnection of a Lot of Links (TRILL) [14] and Shortest Path Bridging (SPB) [15] protocols.

In DC controlled environments, for packet-based intra-DC communications, the computation of the different fields could be left to an advanced layer-2 fabrics allowing DC traffic engineering such as TRILL, SPB or OpenFlow. As for the extra-DC segment, stateless solutions do not allow enough control to choose paths. The LISP-TE features could thus prove helpful since it explicitly encodes some LISP nodes to go through, but requires the usage of a wide enough WAN operational LISP network to prevent packets from making a prejudicial detour because they have to go through LISP routers.

[2]See: http://openvswitch.org/pipermail/git/2013-February/003666.html.

IV. Experimental results

In this section, we report and discuss the results obtained performing several MPTCP communications over the experimental test bed. First, we present the test bed setting and the developed node features we publish as open source code, then we assess the achievable gain with our Augmented MPTCP (A-MPTCP) solution. Finally we qualify the data-plane overhead of the cross-layer interaction.

A. Network test bed

Let us illustrate the experimental test bed we used for the experimentation of our solution, displayed in Fig. 5. It implements our basic reference augmented TCP scenario described in Fig. 3. We used a MPTCP-capable virtual machine hosted in the Paris-Est DC of the French NU@GE project [20], disposing of two Internet Service Providers, Neo Telecom (AS 8218) and Alphalink (AS 25540). On the other side, we use a single-homed and MPTCP-capable Cloud user. The Internet-working between the VM and user is such that two disjoint Internet paths are used down to the user's ISPs, and such that the two intra-DC paths between the VM's hypervisor and the DC border LISP router are disjoint ones.

In such a configuration, the highest improvements can be reached when the Cloud user's provider access and transit links are not congested. The test bed scenario can be considered as quite representative for real cases, where typically Cloud users are not multihomed and the DC has a few ISPs. Moreover, by targeting the more basic configuration with only two subflows we can more precisely demonstrate the benefit of our LISP-MPTCP cross-layer cooperation solution.

B. Open Source Nodes

In terms of open source software, we used the open source MPTCP Linux implementation [9] and the LISPmob [19] implementation (preferred over the faster BSD OpenLISP router [22] [23] because more easily customizable). We then applied these necessary modifications to the open source nodes. We published the patches in an open source repository [25]. More precisely,

- to the MPTCP kernel, to add our path discovery feature that retrieves the number of local and remote RLOCs for each destination it is talking to as previously described;
- to the LISPmob router so that: (i) it acts as a Mapping Resolver too; (ii) it balances the two subflows deterministically between its two RLOCs. For (ii), instead of resorting to a hash of the tuple (source IP, destination IP, Source port, Destination Port, Time To Live), we replaced the load balancing mechanism by a single modulo (number of remote RLOCs) on the TCP source port and force MPTCP to start TCP subflows with certain source port numbers.

To ease the development and to decorrelate the path discovery mechanism from other MPTCP mechanisms a stated in [13], we minimized the modifications to the kernel and instead created a kernel module called by the kernel each time MPTCP establishes a new connection. We also integrated

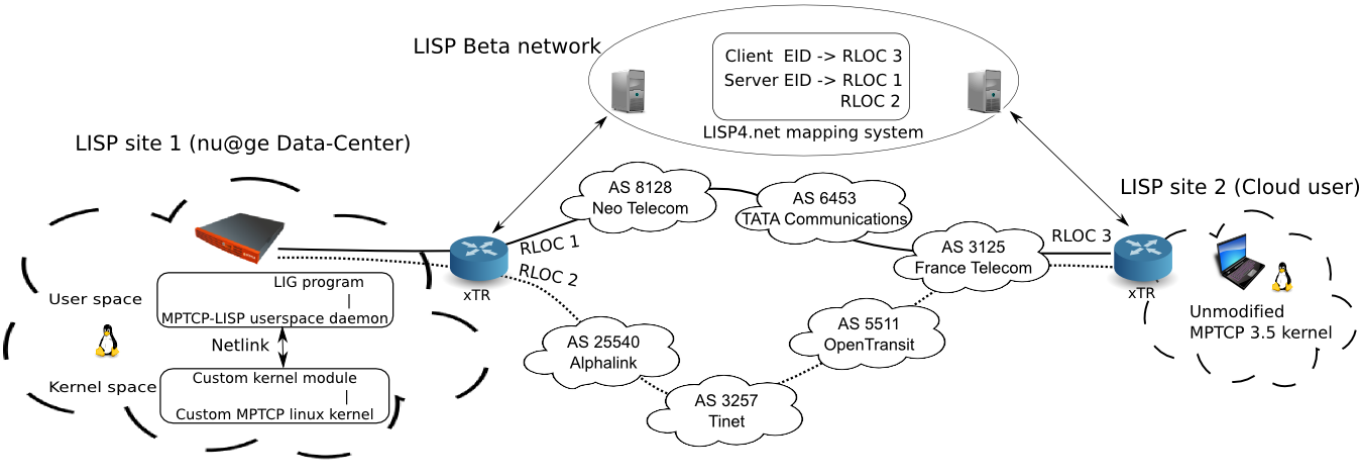

Fig. 5. Cloud network test bed scenario

an existing user space program named LIG (LISP Internet Groper, [16], [17]) to request EID-to-RLOC mappings (equivalent of the Domain Information Groper DIG but for LISP networks), using MAP-REQUEST and MAP-REPLY control-plane messages. The LISP nodes were connected to the LISP Beta Network test bed [18]. In order to retrieve a list of RLOCs responsible for an EID (and implicitly their number), we created a user space python daemon listening to requests transmitted by the kernel module, which then interacts with the LIG program. In the following, we do not differentiate between the two user space programs since it is just a matter of implementation.

C. Transfer times

In order to demonstrate the previously described architecture, we worked on the same topology as in Figure 3. A single homed client (with one link to AS3215) downloads 30 files from a single homed server (one link to a LISP software router) in a dual-homed data center (both links active, one towards AS25540, another one towards AS6453). Each transferred file is set bigger than the previous one by an increment of 256 KB to assess the performance for different volumes. We record 20 transfer completion times for each file, and plot in the following maximum-minimum error bars. We conduct the experimentations using four different configurations:

1) legacy TCP: with no cooperation with LISP, and a single TCP flow.
2) MPTCP: with no cooperation with LISP, and a single MPTCP subflow.
3) A-MPTCP: with cross-layer cooperation, creating as many subflows as the product of remote and local RLOCs, i.e., 2 subflows.
4) A-MPTCP overridden: we manually override the daemon result to create 3 subflows instead of 2

One can reasonable expect the cases 1 and 2 to be very similar in the provided configuration since MPTCP should use one flow only. During our tests, the server upload speed could fill the client download speed (8 Mbit/sec, corresponds to RLOC 3 in Figure5) with a single TCP connection. In order to exhibit an increase in throughput via the use of several subflows, we limited RLOC 1 and 2 throughput via the linux QoS utilities so that each each RLOC could send no more then 4 Mbit/sec. On Figure 6, we see that unassisted MPTCP (i.e., MPTCP without LISP support) and TCP transfer times are quite close, as expected, MPTCP being a little slower than TCP: the primary cause is the additional 20 bytes overhead per packet introduced by MPTCP. More importantly, we can see that our A-MPTCP solution combining MPTCP and LISP performs significantly better. For almost all file sizes, we get a gain close to 90%, i.e., with the additional LISP-enabled subflow we can nearly halve the transfer time. This very clearly shows the important benefit we can reach with our Augmented MPTCP - LISP cross-layer cooperation module.

We also get an interesting result by forcing the creation of three MPTCP subflows even if there are only two RLOCs at the server side. We notice that the throughput is on average worse than for the suggested A-MPTCP behavior with two subflows, but it sometimes does perform better than with two subflows only. The decreased throughput makes sense as having three subflows going out two RLOCs means that at least two of the subflows compete for bandwidth in some part of the WAN. The improvement is likely due to some further load-balancing occurring on the WAN (on the outer header) and of transient bottleneck free subpaths, hence not as deterministic as for the the two subflows case.

It is worth noting that due to the use of linux QoS utilities to cap the bandwidth, it could induce from times to times some latency (the RTT changing between 60ms to 300ms on each path) during the transfers without much impact on MPTCP.

D. Data-plane overhead

From a data-plane forwarding perspective, we recall that LISP performs an IP-in-IP encapsulation with shim UDP and LISP headers. In order to get an idea of the LISP header

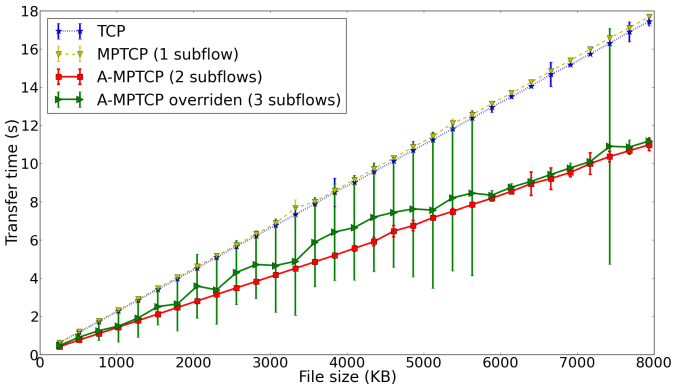

Fig. 6. Completion times for different file sizes

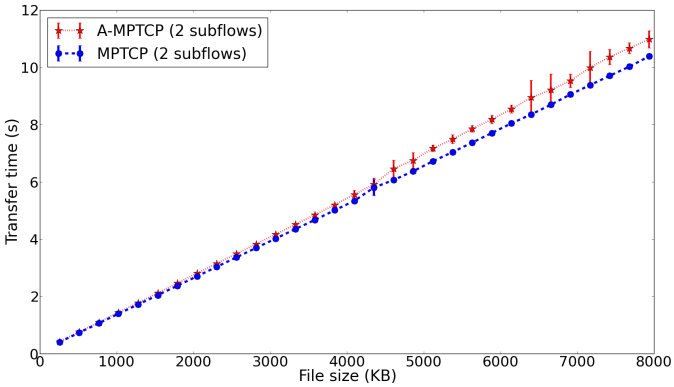

Fig. 7. Transfer times for two subflows with and without LISP

overhead, we recorded the transfer times in two cases. A first case with two MPTCP subflows enabled via LISP, and a second case with the same two MPTCP subflows, but manually configured, without LISP, hence avoiding data-plane encapsulation overheads. The results are shown in Figure 7. At a first look one can immediately notice that the overhead is most of the time negligible. It is worth noting that neither our connection or LISPmob allowed us to test higher rates. Nevertheless, we suspect that at higher rates, we might see a more important processing overhead of LISP traffic since the UDP encapsulation prevents the system from offloading TCP segmentation to the NIC.

V. Conclusions

We have shown that the current MPTCP specification can be refined, adding features to its path discovery, and that the resulting Augmented MPTCP (A-MPTCP) can achieve better performance thanks to a better knowledge of the underlying IP topology gathered via LISP protocol in a LISP-based Cloud network.

Our experimentations on a real large-scale test bed involving one data-center network show that the throughput, hence the transfer time, can be greatly improved, thanks to the cross-layer A-MPTCP protocol cooperation we propose in this paper. Our proposition consists in allowing an MPTCP endpoint to gather information about the IP Routing Locators using LISP control-plane messages to query the LISP mapping system. We show that with just one additional LISP-enabled subflow, a transport-layer connection spanning the Internet via disjoint AS paths can terminate file transfers twice faster. It is therefore reasonable to generalize these results stating that, in absence of middle-boxes filtering the MPTCP signaling, the achievable gain is directly proportional to the number of additional LISP-enabled subflows. Hence the higher the multi-homing degree of Cloud clients and data-centers, the higher the achievable gain.

It is worth stressing that one data-center does not require a worldwide pervasive LISP deployment to offer this performance to its clients, but just need to enable LISP (and MPTCP) at user's endpoints (e.g., using LISPmob [19]) for mobile users, or client's border routers (e.g., using OpenLISP [22] [23]) for enterprise/business customers. It is also worth noting that the proposed A-MPTCP solution does readily apply to inter-DC communications spanning an IP WAN and also the Internet, provided that at least one DC is multi-homed, with no modification to the signaling process between MPTCP and LISP nodes.

As a future work we plan to extend this cross-layer protocol architecture and the related prototype to take into account Local Area Network (LAN) and intra-DC path diversity information. This would allow the solution to cover intra-DC MPTCP communications as well. Indeed, instead of consulting a LISP mapping resolver, one could interrogate a Link State Protocol database or mapping directory (such as ISIS in TRILL or SPB), or ad-hoc DC controllers.

Acknowledgement

This article was partially supported by the NU@GE project [20], funded by the French "Investissement d'avenir" research programme, and the ANR LISP-Lab project (contract nb: ANR-13-INFR-0009) [21]. The authors would like to thank Nicolas Turpault and Alain Duvivier from Alphalink for their support in the Data-Center interconnect needed for the test bed.

References

[1] L. Ong, J. Yoakum, "An introduction to the stream control transmission protocol (SCTP) ", *RFC 3286*, May 2002.

[2] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, "Architectural guidelines for multipath TCP development ", *RFC 6182*, March 2011.

[3] M. Coudron, S. Secci, G. Pujolle, "Augmented Multipath TCP Communications", in *Proc. of 2013 IEEE Int. Conference on Network Protocols (IEEE ICNP 2013)*, Poster session, Oct. 7-11, 2013, Gottingen, Germany.

[4] D. Farinacci, V. Fuller, D. Meyer, D. Lewis, "The Locator/ID Separation Protocol ", *RFC 6830*, Feb. 2013.

[5] O. Bonaventure "Multipath TCP", Tutorial, IEEE CloudNet 2012. http://www-phare.lip6.fr/cloudnet12/Multipath-TCP-tutorial-cloudnet.pptx

[6] P. Raad, G. Colombo, D. Phung Chi, S. Secci, A. Cianfrani, P. Gallard, G. Pujolle, "Achieving Sub-Second Downtimes in Internet Virtual Machine Live Migrations with LISP", in *Proc. of IEEE/IFIP IM 2013*, 27-31 May 2013, Ghent, Belgium.

[7] S.C. Nguyen, T.M.T. Nguyen, G. Pujolle, S. Secci, "Strategic Evaluation of Performance-Cost Trade-offs in a Multipath TCP Multihoming Context", in *Proc. of 2012 IEEE Int. Conference on Communications (IEEE ICC 2012)*, 10-15 June 2012, Ottawa, Canada.

[8] K.Sriganesh, "Multipath Transmission Control Protocol Proxy", WIPO Patent No. 2012049631. 20 Apr. 2012.

[9] C. Raiciu, C. Paasch, S. Barr, Alan Ford, Michio Honda, F. Duchne, O. Bonaventure and Mark Handley. "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP ", *Proc. of USENIX Symposium of Networked Systems Design and Implementation (NSDI'12)*, San Jose (CA), 2012.

[10] D. Farinacci, P. Lahiri, M. Kowal, "LISP Traffic Engineering Use-Cases", *draft-farinacci-lisp-te-02*, Jan. 2013.

[11] D. Farinacci, D. Meyer, J. Snijders, "LISP Canonical Address Format (LCAF) ", draft-ietf-lisp-lcaf-02, March 2013.

[12] G. Detal et al., "Revisiting Flow-Based Load Balancing: Stateless Path Selection in Data Center Networks", *Computer Networks* (in press).

[13] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, Jan. 2013.

[14] R. Perlman, "RBridges: Transparent Routing", in *Proc. of INFOCOM 2005*, Mar. 2004.

[15] D. Allan et al., "Shortest path bridging: efficient control of larger ethernet networks", *IEEE Communications Magazine*, 48.10 (2010): 128-135.

[16] D. Farinacci, D. Meyer, "The LISP Internet Groper", RFC 6835.

[17] The LISP Internet Groper (LIG) open source version (website): https://github.com/davidmeyer/lig

[18] LISP Beta Network testbed (website): http://www.lisp4.net

[19] "LISPmob open source LISP node "(website): http://www.lispmob.org

[20] "NU@GE project "(website): http://www.nuage-france.fr

[21] ANR LISP-Lab project (website): http://lisplab.lip6.fr

[22] D. Saucez, L. Iannone, O. Bonaventure, "OpenLISP: An open source implementation of the locator/ID separation protocol ", *Proc. of ACM SIGCOMM 2009*, Demo paper.

[23] D. Phung Chi, S. Secci, G. Pujolle, P. Raad, P. Gallard, "An Open Control-Plane Implementation for LISP networks", in *Proc. of IEEE IC-NIDC 2012*, Beijing, China, 21-23 Dec., 2012.

[24] A. R. Curtis, W. Kim, P. Yalagandula, "Mahout: Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection", in *Proc. of IEEE INFOCOM 2011*.

[25] MPTCP & LISPMOB open source patches: https://github.com/teto