Towards a Multipath TCP Aware Load Balancer

Simon Liénardy, Benoit Donnet
Université de Liège
Montefiore Institute
Belgium
firstname.name@ulg.ac.be

ABSTRACT

Multipath TCP has been recently introduced in order to allow a better resource consumption and user quality-of-experience. This is achieved by allowing a connection between two hosts through multiple subflows. However, with the rise of middleboxes and inherent Internet ossification, the large-scale deployment of this TCP extension is difficult. In particular, a load balancer at the entry point of a data center may forward subflows to different servers, cancelling so the advantages of Multipath TCP.

In this paper, we introduce MpLb, a Multipath TCP aware load balancer that fixes this particular issue without any modification to the Multipath TCP protocol itself. We demonstrate advantages of MpLb through a proof-of-concept.

CCS Concepts

•**Networks** → **Middle boxes / network appliances; Transport protocols;** *Network protocol design;*

Keywords

Multipath TCP, load balancer, middlebox, MpLB

1. MPTCP AND LOAD BALANCERS

Multipath TCP (MpTcp) [1] is a new TCP extension enabling to create multiple subflows between two hosts instead of the well-known single flow characterized by its 5-tuple (Protocol, Local address, Local Port, Remote address, Remote Port). Each subflow has its own 5-tuple and thus is forwarded independently of the others. As a result, the connection will be split among different paths in the network, leading to a better resources consumption [2]. Moreover, different subflows can be transmitted on diverse physical layer technologies. As an example, a smartphone can use both Cellular connection and Wi-Fi. If one of the subflows is lost, the other pursues the data transmission in a transparent way from the user, improving so its quality-of-experience [3].

The use of MpTcp is transparent for the application layer. The TCP concept of a data flow sent from a host to another still exists: MpTcp splits the data among several subflows and ensure that is delivered to upper layer in the correct order. Each subflow consists in a regular TCP connection and thus maintains its own sequence numbers, receiver window, etc.

In order to use MpTcp, hosts have to exchange `MP_CAPABLE` option during the 3-way handshake of the first subflow. It enables to exchange Keys between the hosts. Each new added subflow will authenticate that it belongs to a MpTcp connection thanks to the `MP_JOIN` option that contains, for the `SYN` segment, a Token, that is a cryptographic hash of the receiver Key exchanged in the first subflow. The rest of the handshake pursues authentication thanks to nonces and HMAC [1]. This is partially shown on Fig. 1(b).

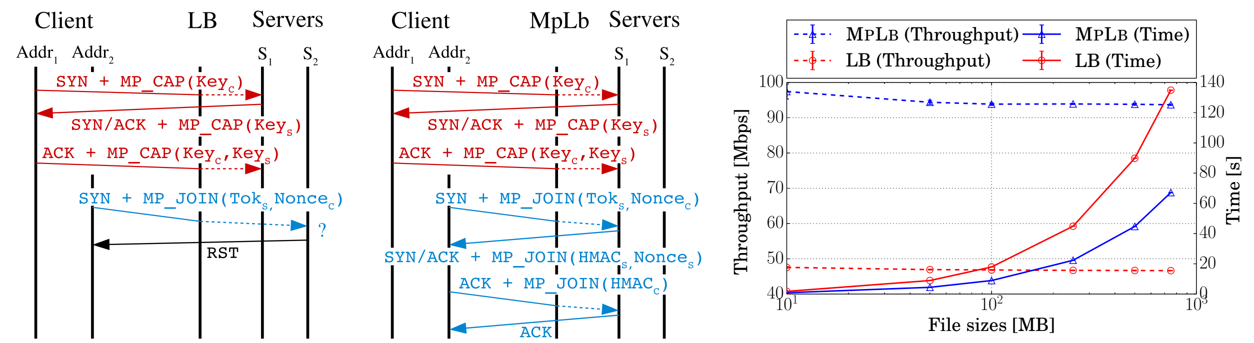
With the rise of middleboxes [4], TCP extensions and new transport protocol are difficult to deploy [5]. In particular, middleboxes have also heavily influenced the design of MpTcp [1, 5]. However, if MpTcp is able to cope with several types of middleboxes, load balancers spreading the traffic among several servers are still an issue. Indeed, if the load balancer is unaware of MpTcp, as each subflow of a given MpTcp connection has its own 5-tuples, those subflows will be balanced to distinct servers, as illustrated on Fig. 1(a). As a consequence, MpTcp falls back to standard TCP, leading to a potential loss in the user quality-of-experience.

In this paper, we tackle this particular problem by proposing MpLb, a MpTcp aware load balancer that is able to redirect all subflows of a given MpTcp connection to the same server. MpLb does not require any modification in the MpTcp protocol. In the section 2, we present the MpLb concepts and demonstrate its advantages through a proof-of-concept.

2. MPLB

First, a *client* (i.e., a connection initiator) that wants to use MpTcp with the server will add in the 3-way handshake of the first subflow the `MP_CAPABLE` option. The exchanged `MP_CAPABLE` options contain two pieces of information that the load balancer needs to correctly handle the subsequent subflows: the server Key and the cryptographic hash algorithm to compute the Token from the key. This algorithm is negotiated between the client and the server: the client proposes a list of hash algorithms in the `SYN`, the server selects one of them in the `SYN+ACK`, and the client confirms it in the `ACK`. Hence, the third packet of the 3-way handshake suffices to compute the server's Token because it contains both

ANRW '16, July 16, 2016, Berlin, Germany
© 2016 ACM. ISBN 978-1-4503-4443-2/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2959424.2959426



(a) Flows between MpTcp and unaware load-balancer

(b) Flows with MpLb

(c) MpLb performance vs. unaware load-balancer

Figure 1: MpLb principles and performance

client and server Keys and confirms the hash algorithm.

Second, MpLb uses a hash map (hereinafter referred as *<$flow_{id}$, output> map*) to keep track of the link between the flow identifier, computed on the basis of the 5-tuple, and the outgoing interface. This enables to quickly forward to the same server all the datagrams that belong to the same flow. The outgoing interface of the first subflow is decided when the SYN segment is processed. Since it is the first subflow, the MpLb can choose on which interface it will be forwarded either on a round-robin basis or on a more complex load balancing policy basis. When the ACK is received, MpLb computes the server's Token and a second hash map (hereinafter referred as *<Token, output> map*) is used to maintain state about the relationship between the Token and the outgoing port of the first flow.

Third, when a new subflow connection is detected, the Token is extracted from the MP_JOIN option. The <Token, output> map is queried. If the token is unknown, MpLb chooses an outgoing interface at random. If the token corresponds to a previously seen MpTcp connection, the subflow is forwarded on the same output interface. The mapping between the subflow id and the output is also remembered in the <flow_{id}, output> map. Then, all the datagrams of the newly added subflow can be balanced towards the same destination. All this process is illustrated in Fig. 1(b).

We implemented a proof-of-concept using the Click Modular Router [6]. Click enables to design a router by connecting pre-implemented boxes that perform basic operations. Once combined, these boxes form the processing flow of a packet in a router. We implemented a box that has one input and multiple (the number is configurable) outputs. This box balances an incoming packet among the output according to what we explain above. To respect Click philosophy, the balancing box does not tackle the forwarding decision: a previous box in the packet flow cope with that problem. In fact, the packets forwarded to the servers are given in input of the balancing box.

Based on our Click implementation, we demonstrate, as a proof-of-concept, the advantages of MpLb. The topology we used is made of four virtual machines: a client, two servers, and the load balancer (LB). The client and the servers are MpTcp capable (MpTcp linux version 0.89 [7]). The LB does not require to have a MpTcp capable kernel since it is the Click program that is in charge of routing operation.

The client is connected to the LB thanks to two links limited at 50 Mbps (this is a software limitation). The servers are located on the same collision domain as one of the output of the LB. Interfaces of both servers have the same IP address but different MAC. Once the LB has chosen the destination server, the datagram is encapsulated in a frame with the chosen server MAC address. In our proof-of-concept, the LB knows the MAC address of the servers and no ARP request is required (in LB → server direction).

We tested two LB solutions: one LB selects destination based on a hashing of IP_{src} and IP_{dest} and is MpTcp unaware. We could implement a 5-tuple hash but that was not necessary to show as, in this case, the LB cannot handle MpTcp subflows. The two IP addresses of the client were assigned in order that this LB does not choose by chance the same output for the two MpTcp subflows. The other LB solution is our MpLb implementation.

Fig. 1(c) presents the time taken and the throughput to download, from the client, files of increasing size (the X-Axis, in log-scale). Each data point on Fig. 1(c) represents the mean value over 30 runs of the experiment. We determine 95% confidence intervals for the mean based, since the sample size is relatively small, on the Student t distribution. These intervals are typically, though not in all cases, too tight to appear on the plot. We can see that there is a factor 2 between both time and throughput curves. In fact, MpLb manages to use both links between the client and the LB to transmit data. The link between the LB and the server was set up to not be the limiting factor of the exchange. We also see on Fig. 1(c) that the simple LB (MpTcp unaware – red curve) leads to a drop in performance. This is due to the fact that it balances the second subflow to the wrong server, that subsequently resets the connection (as already illustrated in Fig. 1(a)). As a result, only a single regular TCP flow can be used for the exchange.

3. CONCLUSION

This paper introduces a case of study in which MpTcp cannot work properly: a load balancer that spreads the subflows among different interfaces, making impossible for a server behind this balancer to gather all the subflows. To fix this, we proposed MpLb, a load balancer that is MpTcp aware. We demonstrated its advantage through a simple proof-of-concept. In the near future, we will improve MpLb and study its behavior on more complex topologies, in particular when several load balancers are run in parallel at a datacenter entry.

4. ACKNOWLEDGMENTS

This work was supported by the European Commission European Commission H2020-688421 MAMI project, SERI 15.0268.

5. REFERENCES

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," Internet Engineering Task Force, RFC 6824, January 2013.

[2] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, August 2011.

[3] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "A first analysis of multipath TCP on smartphones," in *Proc. Passive and Active Measurement Conference (PAM)*, March/April 2016.

[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, August 2012.

[5] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP," in *Proc. ACM Internet Measurement Conference (IMC)*, November 2011.

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, August 2000.

[7] C. Paasch, S. Barré et al., "Multipath TCP in the Linux kernel," available from http://www.multipath-tcp.org.