Making Multipath TCP friendlier to Load Balancers and Anycast

Fabien Duchene, Olivier Bonaventure
Institute of Information and Communication Technologies, Electronics and Applied Mathematics (ICTEAM)
Université catholique de Louvain
Louvain-la-Neuve, Belgium
Email: firstname.lastname@uclouvain.be

***Abstract*—Multipath TCP is a recent TCP extension that enables the utilization of different paths for a single connection. This provides various benefits including bandwidth aggregation and fast handovers on mobiles. A Multipath TCP connection starts with a single TCP connection called subflow and other subflows are added later to increase bandwidth or support failover. One drawback of Multipath TCP is that it is not currently compatible with stateless load balancers which rely on the five-tuple for their forwarding decision. This hinders the deployment of Multipath TCP.**

We show that this limitation can be circumvented with a small change to the handling of the initial subflow. Clients use this subflow to discover the load-balanced server and the additional Multipath TCP subflows are terminated at a unique address associated to each physical server. With this small change, Multipath TCP becomes compatible with existing stateless load balancers. Furthermore, we show that the same approach enables anycast Multipath TCP services, a major benefit given the difficulty of deploying anycast TCP services. We implement this modification in the Linux kernel and demonstrate its benefits with several micro benchmarks.

I. Introduction

Multipath TCP [19], [20], [35] is a recent TCP extension that enables hosts to send packets belonging to one connection over different paths. Several use cases have emerged for Multipath TCP during the last years [15]. Apple uses Multipath TCP on all its tablets, smartphones and laptops to support the Siri voice recognition application. In this use case, Multipath TCP provides very fast failovers when a smartphones leaves the coverage of a WiFi access point. A second use case for Multipath TCP is bandwidth aggregation. In Korea, high-end smartphones include Multipath TCP to bond the bandwidth of their WiFi and cellular interfaces and achieve higher throughputs [36]. Finally, network operators have started to deploy Multipath TCP proxies to bond xDSL and LTE networks in rural areas [15], [14].

Given that the Multipath TCP specification was published in 2013, the Apple deployment that began in September 2013 is the fastest deployment of a TCP extension [21]. Despite this fast start, Multipath TCP is still not widely supported by servers [27]. There are several reasons for this limited deployment. On the client side, Apple's implementation of Multipath TCP is not exposed to regular applications and the implementation in the Linux kernel [31] is not included in the official kernel. On the server side, the deployment of Multipath TCP is hindered by technical problems [32]. Many servers reside behind load balancers [4]. There are basically two families of load balancers: the stateless and the stateful load balancers. Stateful load balancers maintain state for each established TCP connection and load balance them among different servers. Some of these load balancers have been upgraded to support Multipath TCP [9], [3]. However, the state maintained by these load balancers limits their scalability. This is the reason why large content providers prefer to deploy load balancers [17] that stores as less state as possible. Those load balancers operate on a per-packet basis and take their load balancing decision based on field of the IP and TCP headers (e.g. the five tuple that uniquely identifies each TCP connection). Multipath TCP breaks this assumption and forces to rethink the operation of stateless load balancers. Two groups of researchers have proposed modifications to load balancers to support Multipath TCP [29], [26]. We discuss them in details in section II.

In this paper, we view the load balancing problem from a different angle. Instead of changing the load balancers to support Multipath TCP, we propose to slightly change Multipath TCP to be compatible with existing load balancers. This enables Multipath TCP to be used in any environment where load balancers already are and is a much simpler deployment path than changing load balancers. Our modification is simple since it relies on a single bit in the option exchanged during the three-way handshake. We implement it in the Linux kernel and demonstrate its performance based on lab measurements. Furthermore, we show that with this small modification it becomes possible to efficiently support anycast services over Multipath TCP. This opens new benefits for Multipath TCP in addition to the existing bandwidth aggregation and fast failover use cases.

II. Background and Motivation

Multipath TCP is a recent TCP extension defined in [19]. A key benefit of Multipath TCP is that a Multipath TCP connection can transport data over different paths. A typical example is a smartphone that wants to use both its WiFi and LTE interfaces to exchange data. A Multipath TCP connection always starts with a three-way handshake like regular TCP connections, except that the `SYN` and `SYN+ACK` carry the

978-1-5090-6501-1/17/$31.00

MP_CAPABLE option. This option carries keys whose usage is described later. This TCP connection is called the initial subflow in the Multipath TCP terminology. Once established, data can flow in both directions over the initial subflow. The added value of Multipath TCP comes from its ability to use different paths. This is done by creating one subflow (i.e. also a TCP connection) over each of those paths. Considering our smartphone example, if the initial subflow was created over the cellular interface, then another subflow is created over the WiFi interface. Each subflow is created by using the TCP three-way handshake with the MP_JOIN option in the SYN packets. This option contains an identifier of the Multipath TCP connection to which the subflow is attached (called token in [19]) that is derived from the information exchanged in the MP_CAPABLE option and some authentication information.

Besides MP_CAPABLE and MP_JOIN, Multipath TCP also defines other TCP Options. When data are sent over different subflows, they needs to be reordered at the destination. This is achieved by using the DSS option that provides a sequence numbering scheme at the level of the Multipath TCP connection. Thanks to this option, some data sent over one subflow can later be resubmitted over another subflow if needed. Another important Multipath TCP option is ADD_ADDR. It enables a host to announce its other addresses. For example, a dual stack server could accept a Multipath TCP connection over its IPv4 address and then use the ADD_ADDR option to announce its IPv6 address to its client that will then be able to create an IPv6 subflow towards the server.

A. Load balancing principles

A network load balancing infrastructure is typically composed of one or several load balancers located between the physical servers that host the content and the edge routers as shown by figure 1. In this paper, we focus on layer-4 load balancers[7], [17], [2] that do not terminate the TCP connection unlike some layer-7 load balancing solutions[8], [6].

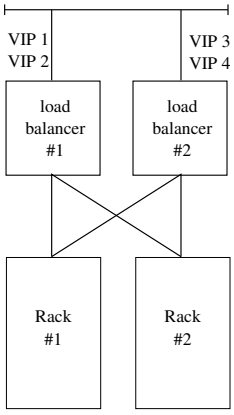

Fig. 1. Typical deployment of Layer-4 load balancers

Load balancers typically announce Virtual IP addresses (VIP) to the Internet. A VIP differs from a traditionnal IP address because it is not assigned to a single server. It usually belongs to a service whose content will be served by multiple servers located behind the load balancers.

When a client tries to connect to a service, it usually obtains the VIP via a DNS query and then sends packets to this address. When the first packet of a connection reaches the load balancer, the load balancer needs to select one of the service's servers for this particular connection. The specific algorithm used to select the best server is out of the scope of this paper, but it is important to emphasise that once that the server has been selected, all the packets belonging to this specific connection will be forwarded to that particular server. With regular TCP, load balancers usually extract the 5-tuple of the connection (protocol, source address, destination address, source port, destination port) from each received packet and assign each tuple to a specific server. Some load balancers simply forward these packets to the corresponding server [7] while others encapsulate the packet using Generic Routing Encapsulation (GRE)[17]. The main benefit of encapsulation is that the physical server does not need to be physically close to the load balancer.

This solution works perfectly with TCP and UDP because each TCP or UDP packet contains the five-tuple that identify the flow. Multipath TCP unfortunately breaks this assumption. A load balancer should send all the packets that belong to a given Multipath TCP connection to the same physical server. Since a Multipath TCP connection is composed of different TCP connections, a packet can reach the load balancer via any of these TCP connections and thus via possibly different five-tuples. A layer-4 load balancer cannot rely only on the information contained in this packet to determine the physical server that was selected for this specific Multipath TCP connection.

Several solutions have been proposed to reconcile Multipath TCP with load balancers. A first approach is to use stateful load balancers. Some commercial products [9], [3] already support Multipath TCP and researchers have proposed stateful Multipath TCP capable load balancers [26]. Olteanu and Raiciu propose in [29] a modification to Multipath TCP that enables stateless load balancers to support Multipath TCP. Their solution relies on changing the TCP timestamp option. Instead of using this option to encode timestamps only, they encode an identifier of the physical server besides the load balancer inside the low-order bits of the timestamp option [16]. Since clients always echo the timestamp option sent by servers, this enables the load balancer to receive in each packet an identifier of the physical server that needs to receive the packet. This solution has been implemented and tested in lab environment. However, it suffers from three important limitations. First, load balancers and servers need to be modified to extract the information from the timestamp option. Second, this option, like any TCP option, can appear anywhere in the extended TCP header. This implies that a hardware implementation will be more complex than existing hardware solutions that simply extract the source and destination addresses and ports that are placed at fixed locations in all packets. Third and more importantly, there are various

types of middleboxes that are deployed on the global Internet that change the values of the timestamp transported in TCP Options. This solution is thus fragile in the presence of such middleboxes.

Stateless load balancers are much more scalable than stateful load balancers and large content providers want to continue to use stateless approaches for load balancing. In this paper, instead of modifying the load balancer, a device that's usually hard to modify, we choose instead to slightly modify the protocol by slightly changing how addresses are advertised and used. Given that the IETF is currently finalising the revision of Multipath TCP [19] to publish it as a standard track document [20], this is the right time to propose such a modification.

III. Modifications to Multipath TCP

In a nutshell, our modification to Multipath TCP to support load balancers is to assign two addresses to each physical server: a VIP that is the load balanced address and a unique address that is assigned to each physical server. When a client creates a Multipath TCP connection to a load balancer, it uses the VIP and the load balancer forwards the packets to the selected physical server. The physical server advertises its unique address and the client immediately creates a second subflow towards this address.

A. Restricting the initial subflow

Our first modification concerns the initial subflow that is created by the client. This subflow is created by sending a `SYN` packet that contains the `MP_CAPABLE` option. This subflow is established between the client address and the VIP served by the load balancers. We add a new "B" flag to the `MP_CAPABLE` option returned by the physical server. When the "B" flag is set, this indicates that the source address of the packet carrying this option (in this case the VIP address) cannot be used to create additional subflows. If a smartphone receives a `SYN+ACK` packet with the "B" flag set in response to a `SYN` packet sent over its cellular interface, it infers that it cannot create any additional subflow towards this address. This prevents the smartphone from creating a subflow that would not be correctly load balanced by the load balancer, but this unfortunately also prohibits the utilisation of the WiFi interface to reach the physical server.

All the packets of the initial subflow will reach the load balancer and be forwarded to the physical server chosen by the load balancer for this connection. If the client creates another subflow, the packets belonging to this server must also reach the same physical server. For this, we leverage the existing `ADD_ADDR` option defined in [19]. This option allows a host to advertise one of its IP addresses. We configure each physical server with two addresses: the VIP and a unique address. If the physical server advertises its physical address then the client will be able to create additional subflows towards this address which can bypass the load balancer.

B. Reliable `ADD_ADDR`

If the "B" flag has been set in the `MP_CAPABLE` option, the client is prohibited form establishing any additional subflow until it has received the `ADD_ADDR` option that advertises the unique address of the server. Unfortunately, according to the current Multipath TCP specification, the `ADD_ADDR` option is sent unreliably. This implies that any loss of the packet carrying this option could be problematic, e.g. for a smartphone that moves away from the wireless access point used for the initial subflow. To ensure that addresses are reliably advertised, we add the "E" bit to the `ADD_ADDR` option. This bit is reset when a host advertises an address. When a host receives an `ADD_ADDR` option with the "E" flag reset, it must echo this `ADD_ADDR` option with the "E" flag set. This echoing serves as an acknowledgement of the `ADD_ADDR` option.

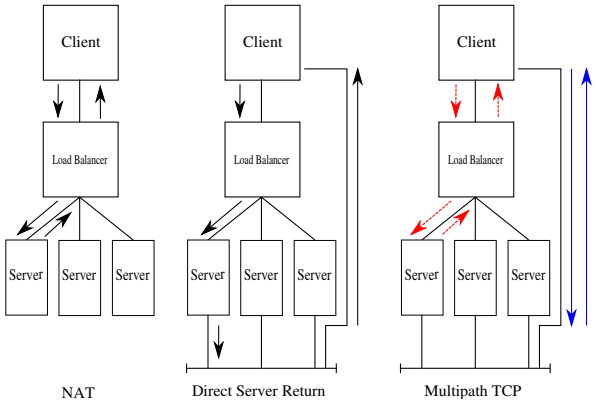

Fig. 2. Different types of load balancer deployments

IV. Use cases

In this section, we describe two use cases of the protocol extension described in the previous section. The first one is that it becomes possible to place the load balancers complete off-path once the Multipath TCP connection has been established. The second one is that with our proposed extension it becomes possible to deploy anycast services over Multipath TCP, even in small networks.

A. Beyond Direct Server Return

Several deployment scenarios exist for load balancer. A simple approach is to place the load balancer in front of all the physical servers such that all the packets sent and received by the physical servers pass through the load balancer. This type of deployment is widely used when only a few physical servers and used. The main advantage of this deployment is that it is simple to deploy and operate. However, since all the packets pass through the load balancer, it could become a bottleneck when the traffic load increases. This is illustrated in the left part of figure 2.

Large web farms use a different approach to deploy their load balancers to support higher traffic loads. HTTP is highly asymmetrical. Most of the HTTP traffic is composed of the data packets that are sent by the physical servers towards the clients. The clients themselves only send the HTTP requests which are much less frequent. Many web farms leverage this

traffic asymmetry by configuring the router/switch attached to the physical server to send the packets generated by those servers directly to the clients without passing through the load balancer. The packets sent by the clients (TCP acknowledgements and HTTP requests) still need to pass through the load balancer to be forwarded to the selected physical server. This deployment is illustrated in the center of figure 2.

With our proposed modification to Multipath TCP, it is possible to go beyond Direct Server Return and completely bypass the load balancer for any type of TCP connection. The client establishes the initial subflow with the load balancer that forwards all packets belonging to this subflow to the selected physical server. The physical server advertises its address and the client creates an additional subflow towards this server address. All the packets sent to and from the physical server address automatically bypass the load balancer. Once the additional subflow has been established, the physical server can terminate the initial subflow so that no packet passes through the load balancer anymore. Storage services like Dropbox and Google Drive where the HTTP traffic is less asymmetrical could benefit from this modification. Several APIs have already been proposed and implemented to enable applications to control the Multipath TCP subflows [23], [22]. This deployment is illustrated in the right of figure 2 in which the red arrows (center) are related to the initial subflow, and the blue arrows (right) to the secondary subflow.

B. Supporting Anycast Services

There are three types of addresses that can be supported in an IP network: *(i)* unicast addresses, *(ii)* multicast addresses and *(iii)* anycast addresses. The unicast service is well-known. Multicast is outside the scope of this paper. Anycast has been initially proposed by Partridge et al. in [33]. Anycast applies to a network that contains several hosts that provide the same service. If each of these hosts is configured with the same anycast address, then when a client sends a packet towards the anycast address associated to the service, the network automatically forwards the packet to the closest host. Anycast has several appealing features such as its resilience to failure or its ability to minimize latency. Anycast is widely used to deploy DNS resolvers in ISP or enterprise networks [18], [10]. Given the privacy and security constraints of the DNS service, several researchers have proposed to run the DNS service above TLS and TCP instead of UDP [37], [25].

Unfortunately, it is difficult to use TCP servers with anycast addresses [28]. To understand this difficulty, let us consider the simple network topology shown in figure 3. There are two anycast servers in this network shown as S in the figure. One is attached to router `R2` and another attached to router `R4`. Both advertise the same anycast address in the network. If the client attached to router `R1` creates a connection towards this anycast address, the resulting packets are forwarded to the server attached to `R2`. If the `R1-R2` link fails, the next packet sent by the client towards the anycast address will be delivered to the server attached to `R4`. Since this server does not have state for this TCP connection, it will send a RST packet to terminate it and the client will have to restart this connection.

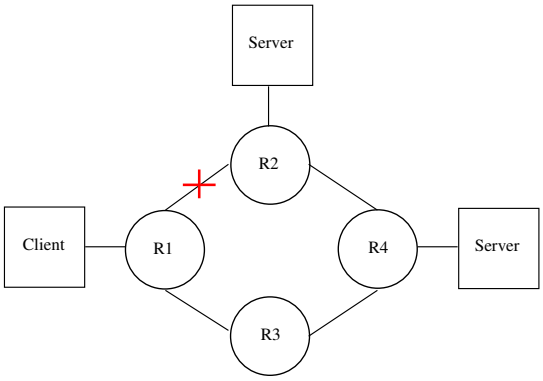

Fig. 3. Anycast workflow.

Thanks to our proposed extension to Multipath TCP, it becomes possible to support anycast services. For this, each unicast server must be configured with two addresses: *(i)* the anycast address that identifies the service and *(ii)* the unique server address that identifies the physical server. Let us consider the same scenario as above. The client creates a Multipath TCP connection towards the anycast address. The network forwards the `SYN` packet to the server attached to router `R2`. This server accepts the Multipath TCP connection and replies with a `SYN+ACK`. The server then advertises its unique address over this initial subflow and then signals to the client to consider this subflow as a backup one. If link `R1-R2` fails, the packets of the initial subflow reach the server attached to `R4`. This server does not have state for this subflow and thus replies with a `RST` packet that terminates the initial subflow. This does not affect the other subflow that is bound to the unique address of the server attached to `R2`. The Multipath TCP connection with the server attached to `R2` continues without any impact on the client.

Anycast TCP services are typically deployed by associating a Fully Qualified Domain Name (FQDN) to each service and using the DNS server to spread the load among different servers. However, there are several situations where a DNS-based solution might not work. A first example are the DNS resolvers mentioned earlier [37], [25]. Those servers must be reachable via an IP address that is advertised by DHCP or through the IPv6 router advertisements. Another example are the different types of proxies that are being discussed within the IETF [14], [36], [15].

V. Performance Evaluation

To demonstrate the benefits of the solution described in the previous section, we first modify the reference implementation of Multipath TCP in the Linux kernel [31]. We then use this implementation to perform experiments in a lab with both load balancers and anycast services.

A. Implementation in the Linux kernel

The Multipath TCP implementation in the Linux kernel [30] is divided in three parts. The first part includes all the functions

```
WHEN A NEW CONNECTION IS ESTABLISHED:
 /* Get the specific IP address */
 ip_addr = GET_SERVER_IP()
 /* Send an ADD_ADDR containing that
    address to the client */
 ADVERTISE_TO_CLIENT(ip_addr)
 /* Change the first subflow to backup mode */
 SET_BACKUP_MODE(get_first_subflow())
```

Fig. 4. Algorithm of the Load balancing.

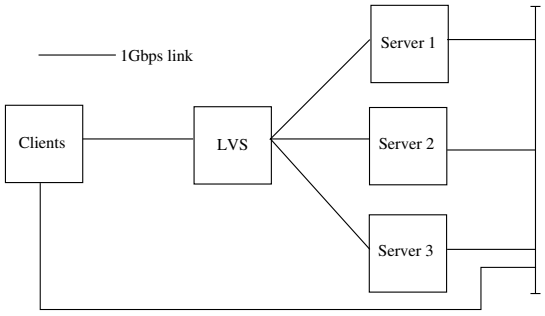

Fig. 5. Evaluation setup.

that send and receive TCP packets. The second part is the path manager. This module contains the logic that manages the different subflow. Several path managers have been implemented [30], [13]. The reference implementation contains the `full-mesh` and the `ndiffports` path managers. The `full-mesh` path manager is the default one. It tries to create a full-mesh of subflows among the addresses available on the client and the server. The `ndiffports` path manager was designed for single-homed clients and servers. On the client side, it creates `n` subflows with different source ports towards the server. It was designed for the datacenter use case described in [34]. The third part is the packet scheduler that selects the subflow that will be used to transmit each packet.

We first add support for the "B" flag in the code that processes the `MP_CAPABLE` option described in section III-A. To support this flag, we had to modify the path manager used by the client to prohibit it form creating any subflow towards the destination address of the initial subflow.

Our second modification was to add the support of the "E" bit in the `ADD_ADDR` option as described in section III-B. We implemented it, by sending the `ADD_ADDR` option in every packet until the reception of the address acknowledgement (reception of an echo, with the"E" bit set to 1), making the transmission of the `ADD_ADDR` option reliable.

To support these two modifications, we have created a new path manager that is tuned for servers behind a load balancer. This path manager does not create any subflow, this is the standard behaviour of path managers running on servers. It advertises the unique server address on the initial subflow and then changes the priority of this subflow to become a backup subflow. Multipath TCP [19] defines backup subflows as follows : *path to use only in the event of failure of other working subflows*. This means that the initial subflow, that passes through the load balancer can still be used but the server encourages the client to use the subflows towards its unique address. An alternative would have been to reset the initial subflow, but this would have been less failure resilient. We have preferred to set the initial subflow in backup mode. The algorithm of this path manager is illustrated by the figure 4.

These modifications represent approximatly 600 lines of kernel code, splitted into three patches (one by feature). 50% of the code lays in the path manager that can easily be plugged into the Linux kernel implementation.

B. Layer 4 load balancers

To evaluate the performances with load balancers, we use the network shown in figure 5. The client is a 2Ghz AMD Opteron 6128 with 16GB of RAM Debian Linux using our modified version of the latest Multipath TCP kernel. This version is based on the Linux kernel version 4.4. The server uses the same hardware configuration and runs lighttpd version 1.4.35 with the same kernel as the client.

The client accesses the web server via a VIP. This VIP is attached to the load balancer. Our load balancer runs on a 2.5Ghz intel Xeon X3440 server running Linux Virtual Server (LVS)[7] configured in NAT mode. We use 1 Gbps Ethernet links between the load balancers and the servers. Each server has a second 1 Gbps interface that is attached to a switch connected to the client.

The purpose of this setup is to mimic a production environment where the servers would have a dedicated network interface directly connected to the Internet. Clients will download web pages, representing a total amount of 4 GB. We use the apache benchmark software [1] to simulate 10 parallel clients. We use `netem` to simulate different delays and different packet loss ratios. To simplify the interpretation of the figures, we started by configuring the load balancer to send all requests to a single server. An evaluation with several servers is done in subsection V-C.

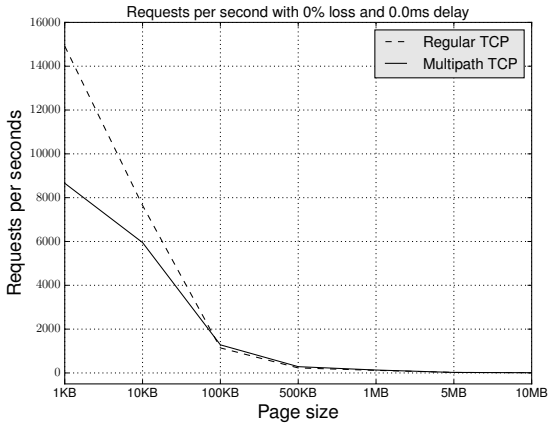

Fig. 6. Number of requests per second without loss or delay.

Figure 6 shows the number of requests completed every second when the client downloads a total of 4 GB using different web pages sizes. The evaluation shows that for small request sizes, Multipath TCP slightly underperforms TCP. This can be explained by the slightly higher cost of establishing Multipath TCP connections[35].

For larger request sizes, starting at 100KB, Multipath TCP and TCP both reach the same amount of requests per second, which is expected because TCP uses the 1 Gbps link connected to the load balancer, while Multipath TCP uses the 1 Gbps link connected directly to the server.

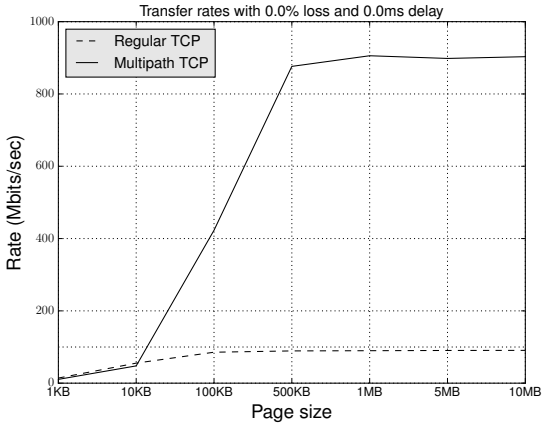

Fig. 7. Transfer rates without loss or delay.

With this experiment, we want to test whether our solution is deployable in a production environment. In this paper we argue that with our proposal it is no longer needed to use costly hardware to run a load balancer. To prove this point, the remaining measurements in this section have been run with the same scripts, but we changed the speed of the link between the client and the load balancer to 100 Mbps.

Figure 7 shows the transfer rate for the same experiment as the one shown in 6, but with a 100 Mbps link between the client and the load balancer. Again, for small request sizes, Multipath TCP slightly underperforms TCP. For larger requests such as 500 KB, Multipath TCP reaches a goodput of 942 Mb/s. The higher Multipath TCP goodput is an illustration that Multipath TCP provides more than TCP. Indeed, shortly after the establishment of the initiation subflow, the client learns the address of the load balanced server and creates a second subflow via the 1 Gbps interface of the server. Multipath TCP then automatically uses the interface going directly to the server and achieves higher goodput than TCP.

1) Impact of the delay: We evaluate in this section whether latency affects the performance of Multipath TCP behind load balancers.

For this experiment, we configure a delay of 20 msec on the link between the client and the load balancer. Figure 8 shows that Multipath TCP is still able to benefit from the 1 Gbps link. Unsurprisingly, the transfer rate for small web objects is lower than when there is no added latency. This is an expected

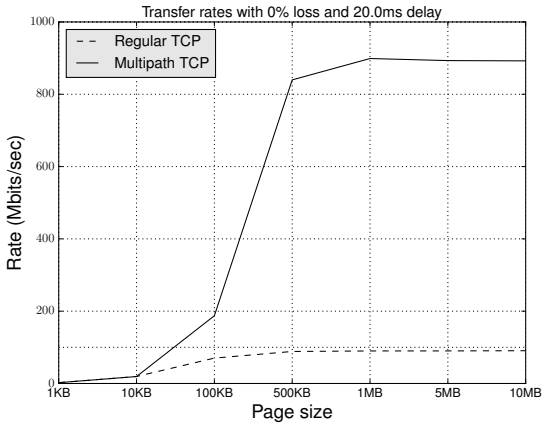

Fig. 8. Transfer rates with no loss and 20ms delay.

and already documented [11] behaviour of Multipath TCP. In our setup, the Multipath TCP starts with an initial subflow that uses the 100 Mbps link. The client sends the HTTP GET over this subflow and it can only start the establishment of the second subflow after the reception of the acknowledgements for this initial data. The 20 msec added latency delays the establishment of the second subflow and thus lowers the total transfer rate.

By increasing the latency to 200ms, as shown by figure 9 we see an important impact on both Multipath TCP and TCP. This high latency increases the time required for congestion control algorithm used on the subflows to ramp up.

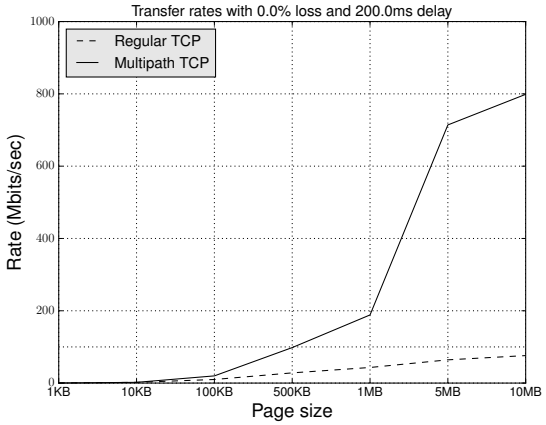

Fig. 9. Transfer rates with no loss and 200ms delay.

2) Impact of packet losses: Packet losses are another factor that can influence the performance of TCP. Measurements over the global Internet have reported packet loss ratios of roughly up to 1%. Our solution needs to cope with two different types of packet losses: *(i)* loss of a TCP packet and *(ii)* loss of a packet carrying the `ADD_ADDR` option that announces the physical address of the server. The standard retransmission and congestion control mechanisms used by

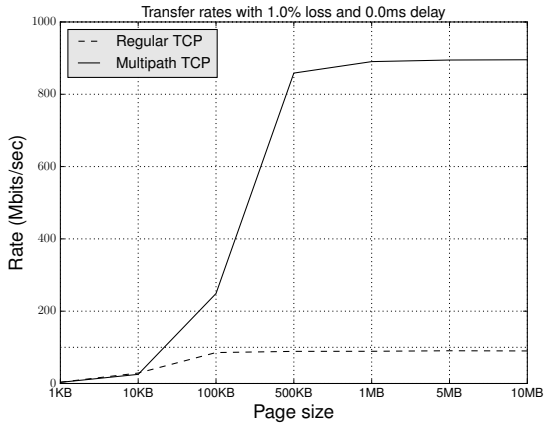

Fig. 10. Transfer rates with 1% loss and no delay.

TCP and Multipath TCP cope with the former type of packet losses. Our implementation copes with the latter by ensuring that the `ADD_ADDR` option is reliably delivered. If a packet carrying the `ADD_ADDR` option is lost, it is retransmitted later to ensure that the remote host has learned the new address. Figure 10 shows that when there is no added latency, the Multipath TCP throughput is not affected by packet losses. A closer look at the packet traces confirmed that a second subflow was created for all Multipath TCP connections.

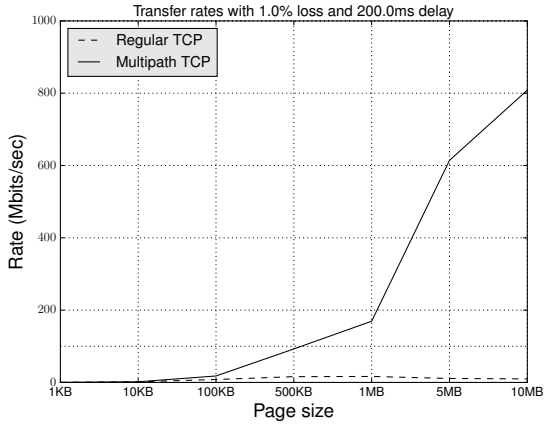

Fig. 11. Transfer rates with 1% loss and 200ms delay.

Figure 11 shows that even when we combine loss and delay, the performance of Multipath TCP is not significantly affected compared to TCP. The important factor being the latency, this can be verified by comparing figures 11 and 9. The high latency playing only for the initial connection establishment, the performances are lower than without latency, but bigger files sizes allows Multipath TCP to achieve a transfer rate of 803Mb/s where TCP achieves 16Mb/s. With these measurements, we demonstrated that with our modification, Multipath TCP works in environments where layer-4 load balancers are used. In this specific setup, we used Linux Virtual Server[7], but any layer-4 load balancer can ben used with the same results. Our measurements show that with our modifications, the load balancer is no longer the bottleneck of the network since it is only used to put the client in relation with the server. With Multipath TCP, the load balancers no longer needs to be expensive machines with a lot of power and network bandwidth as almost all of the traffic can be exchanged direclty between the server to the client.

Direct Server Return (DSR) or Direct return[5] is a technique used in load balancing where the servers are also directly connected to the Internet via a link bypassing the load balancer. This link is used to send the traffic coming from the servers to the client without passing through the load balancer. This technique improves performance in a download scenario, but does not bring benefits for upload scenarios, where most of the traffic is going from the client to the server like in storage scenarios. Our solution, however, fully works in both direction, allowing it to be used in more scenarios.

C. Anycast

A full evaluation of anycast would require a deployment in a larger network that was not possible given the number of servers in our lab. From an abstract viewpoint, an anycast deployment can be considered as a network that distributes the packets sent by clients to the closest server. If the network topology changes, some clients could be redirected to a different server. This change would affect TCP and this is the main reason why anycast TCP is difficult.

To evaluate the support of anycast services, we rely on the network shown in figure 12. Each server has two addresses on its 1 Gbps interface: the anycast address and a unique address. Each server listens to the anycast address but are configured to advertise their unique address with Multipath TCP and set the initial subflow as a backup subflow. These servers are behind a router that uses Equal Cost MultiPath (ECMP) [24] to distribute the load accross the servers. The client is connected to the router via a 10 Gbps link, while the servers are connected to the router via a 1Gbps link each. Like in the previous setup, the clients will establish multiple HTTP connections, 300 in this case, and download files from these servers.

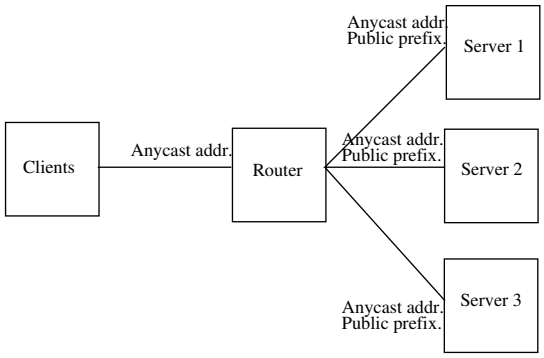

Fig. 12. Anycast Evaluation setup

To simulate network reconfigurations, every 10 seconds, we remove one of the server from the ECMP anycast pool for 5 seconds. After 5 seconds, that server is added again.

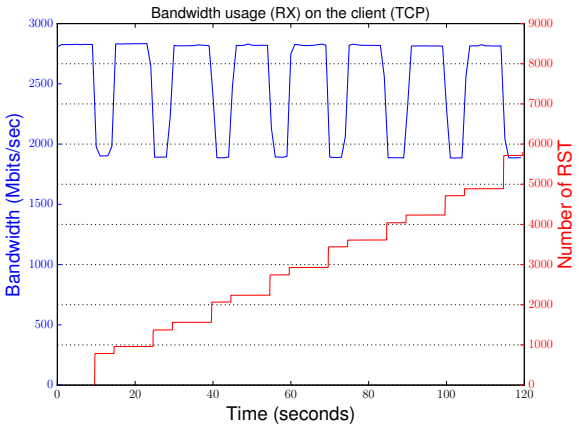

Fig. 13. With TCP, many connections are reset and this affects the utilisation of the client-router link

1) TCP Anycast: Figure 13 shows results obtained with TCP anycast. We perform the measurement during 120 seconds. The client machine runs apache benchmark configured to retrieve very large (100 MBytes) files from one of the anycast servers. We use 300 parallel clients. The figure shows two different curves. The top curve plots the utilisation of the link between the client and the router. When all servers are part of the ECMP pool, the client downloads at 2800Mbps. However, when the router is reconfigured and one of the physical servers is removed from the pool to simulate a topology change in the network, the utilisation of the link drops to 1900 Mbps. This is expected since one server has left the ECMP pool. Unfortunately, a consequence of this network reconfiguration is that some packets towards the anycast address are redirected to a different server than before the topology change. Since this server does have state for the TCP connection, it sends a RST packet and the client needs to restart the entire download. A closer look at the bottom curve of figure 13 reveals that servers also send RST packets when a new server is added to the pool. We experimentally observe that more RST packets are generated when a server is removed from the ECMP anycast pool than when a server is added to the ECMP anycast pool. This is normal because when a server is removed from an ECMP anycast pool, all the TCP connections that were handled by this server are redirected. These RST packets explain why network operators do not want to deploy TCP anycast in entreprise networks.

We now perform exactly the same measurement with our extension to Multipath TCP. Figure 14 shows results that are completely different from those obtained with regular TCP. The first and most important result is that we do not observe any failure of established Multipath TCP connections during the 120 seconds of the experiment. Despite of the 16 topology changes that we simulated, no Multipath TCP connection failed. This is a very important result that confirms that

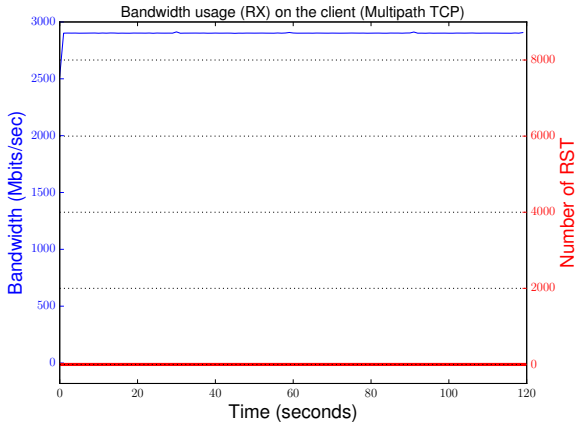

Fig. 14. With Multipath TCP, no connection is reset when the network topology changes

Multipath TCP can be deployed to support anycast services which could bring an additional use case for Multipath TCP. The upper curve of figure 14 reveals that the utilisation of the client-router bandwidth stays at 2.8 Gbps despite the network reconfigurations. When a server is removed from the ECMP anycast pool, the Multipath TCP connections that were handled by this server automatically switch to the second subflow as the initial subflow (towards the anycast address) is redirected to another physical server. This handover is seamless for the application and the connection continues. When a server is removed from the ECMP anycast pool, the packets belonging to an initial subflow are redirected to another server that does not have state for this subflow. This server responds to those packets with RST packets that terminate this initial subflow. However, the second subflow, which is attached to the unique server address, is still up and the data transfer continues.

VI. Security Considerations

Besides distributing the load among different servers, load balancers also shield the physical servers from the open Internet and can filter some of the packets sent to the physical servers depending on their configuration. By advertising the addresses of the physical servers, our solution exposes them more than existing stateless load balancers. If network operators are concerned about the advertisement of the addresses of the physical servers, there are several solutions that can be used to mitigate the security risks.

First, the physical servers only need to accept additional subflows. A security concerned network administrator can easily reject incoming `SYN` packets containing the `MP_CAPABLE` option to prohibit the establishment of new Multipath TCP connection that do not pass through the load balancer. Those filters can be installed on the physical servers or upstream firewalls. This could also be achieved by slightly modifying the Multipath TCP/TCP implementation to reject any `SYN` packets not containing the `MP_JOIN` option on specificed interfaces.

Another point that is worth to be discussed are the additional subflows that can be established by sending SYN packets with the MP_JOIN option towards the physical server that was selected by the load balancer. Multipath TCP [19] relies on two techniques to protect the servers from the establishment of fake subflows. First, the MP_JOIN option contains a 32-bits token that uniquely identifies the Multipath TCP connection. If an attacker wants to add a subflow to an existing Multipath TCP connection, he/se must guess the 32-bits token that identifies this connection. This is not sufficient since the establishment of the additional subflows is authenticated by using HMACs that are computed over 64 bits keys exchanged by the client and the server during the initial handshake. To successfully create an additional subflow, an attacker would need to guess this 64 bits key.

With the above solution, the server plays an active role in mitigating the attack since it needs to match the received SYN with the tokens that it has allocated and then compute the HMAC before sending the SYN+ACK. The computational cost of this HMAC could be a concern in the case of denial of service attacks. In our lan, a single unique address has been assigned to each physical server. In IPv4 networks, this would be the expected deployment given the scarcity of IPv4 addresses. In IPv6 networks, many addresses are available. We could leverage the large IPv6 addressing space and allocate one /64 prefix to each physical server. The server would then announce this prefix to the network to which it is connected. When a new Multipath TCP connection arrives on the server, it assigns a unique IPv6 address from its /64 prefix to this specific connection. We propose to compute the low order 64 bits of this address as $hash(secret, token)$ where $hash$ is a fast hash function, $token$ the token associated to this connection and $secret$ a random number. The server then concatenates the output of this function to its /64 prefix and announces it to its client. This address is unique to this specific connection. If the client creates another subflow towards this server, it will send a SYN packet towards this address with the connection token inside the MP_JOIN option. A simple filter can then be used, either on the physical server or an upstream firewall to verify the validity of the SYN packet without requiring any state. This filter could be implemented as a set of eBFP rules similar to those described by Cloudflare in [12]. Such eBPF rules can process packets at a higher rate than the Linux kernel and thus are very useful when mitigating denial of service attacks.

VII. Conclusion

The deployment of Multipath TCP on servers has been hindered by the difficulty of supporting it on stateless load-balancers. In this paper we have proposed a small modification to Multipath TCP that enables it to work behind any stateless load balancers. This modification has already been accepted by the IETF [20]. We have implemented our modifications to Multipath TCP in its reference implementation in the Linux kernel and have demonstrated its performance with measurements in the lab. An important benefit of our solution compared to existing deployments such as Direct Server Return, is that the load balancer can be placed off-path for long transfers. This is an important feature that could be very useful as the web transitions to the HTTP/2 protocol that will use longer flows than HTTP/1.x.

Our Multipath TCP extension is more generic that simply supporting load balancers in front of servers. It enables network operators to use anycast addresses for Multipath TCP services. This brings another use case for Multipath TCP in addition to the existing deployments that leverage fast failover or bandwidth aggregation.

Repeatability of the results

The measurement results described in this paper were obtained with our modifications to the reference implementation of Multipath TCP in the Linux kernel. These modifications and the measurement scripts is available at https://github.com/fduchene/ICNP2017 to enable other researchers to repeat our measurements and expand them.

References

[1] Apache Bench. https://httpd.apache.org/docs/2.4/programs/ab.html. Accessed: 2017-04-23.

[2] Barracuda Load Balancer ADC. https://www.barracuda.com/products/loadbalancer. Accessed: 2017-04-23.

[3] Citrix Netscaler. http://www.citrix.com/. Accessed: 2017-04-23.

[4] Datanyze - Load Balancers market share report. https://www.datanyze.com/market-share/load-balancers/. Accessed: 2017-04-23.

[5] Direct return in Linux Virtual Server. http://www.austintek.com/LVS/LVS-HOWTO/HOWTO/LVS-HOWTO.LVS-DR.html. Accessed: 2017-05-12.

[6] HAProxy. http://www.haproxy.org/. Accessed: 2017-04-23.

[7] Linux Virtual Server. http://www.linuxvirtualserver.org/. Accessed: 2017-04-23.

[8] Nginx. http://www.nginx.org/. Accessed: 2017-04-23.

[9] Release Note: BIG-IP LTM and TMOS 11.5.0. https://support.f5.com/kb/en-us/products/big-ip_ltm/releasenotes/product/relnote-ltm-11-5-0.html#rn_new/. Accessed: 2017-05-10.

[10] J. Abley and K. Lindqvist. Operation of Anycast Services. RFC 4786 (Best Current Practice), December 2006.

[11] Behnaz Arzani, Alexander Gurney, Sitian Cheng, Roch Guerin, and Boon Thau Loo. Deconstructing MPTCP performance. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 269–274. IEEE, 2014.

[12] Gilberto Bertin. Introducing the p0f BPF compiler. https://blog.cloudflare.com/introducing-the-p0f-bpf-compiler/, 2016.

[13] Luca Boccassi, Marwan M. Fayed, and Mahesh K. Marina. Binder: A system to aggregate multiple internet gateways in community networks. In *Proceedings of the 2013 ACM MobiCom Workshop on Lowest Cost Denominator Networking for Universal Access*, LCDNet '13, pages 3–8, New York, NY, USA, 2013. ACM.

[14] Olivier Bonaventure, Mohamed Boucadair, and Bart Peirens. 0-RTT TCP converters. Internet-Draft draft-bonaventure-mptcp-converters-01, Internet Engineering Task Force, July 2017. Work in Progress.

[15] Olivier Bonaventure and SungHoon Seo. Multipath TCP deployments. *IETF Journal*, 2016, November 2016. http://www.ietfjournal.org/multipath-tcp-deployments/.

[16] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger. TCP Extensions for High Performance. RFC 7323 (Proposed Standard), September 2014.

[17] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. Maglev: A fast and reliable software network load balancer. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 523–535, Santa Clara, CA, 2016.

[18] Xun Fan, John Heidemann, and Ramesh Govindan. Evaluating anycast in the domain name system. In *INFOCOM, 2013 Proceedings IEEE*, pages 1681–1689. IEEE, 2013.

[19] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[20] Alan Ford, Costin Raiciu, Mark J. Handley, Olivier Bonaventure, and Christoph Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. Internet-Draft draft-ietf-mptcp-rfc6824bis-09, Internet Engineering Task Force, July 2017. Work in Progress.

[21] Kensuke Fukuda. *An Analysis of Longitudinal TCP Passive Measurements (Short Paper)*, pages 29–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[22] B. Hesmans, G. Detal, S. Barre, R. Bauduin, and O. Bonaventure. SMAPP: Towards Smart Multipath TCP-enabled Applications. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, pages 28:1–28:7, New York, NY, USA, 2015. ACM.

[23] Benjamin Hesmans and Olivier Bonaventure. An Enhanced Socket API for Multipath TCP. In *Proceedings of the 2016 Applied Networking Research Workshop*, ANRW '16, pages 1–6, New York, NY, USA, 2016. ACM.

[24] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000.

[25] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858 (Proposed Standard), May 2016.

[26] Simon Liénardy and Benoit Donnet. Towards a Multipath TCP Aware Load Balancer. In *Proceedings of the 2016 Applied Networking Research Workshop*, ANRW '16, pages 13–15, New York, NY, USA, 2016. ACM.

[27] Olivier Mehani, Ralph Holz, Simone Ferlin, and Roksana Boreli. An Early Look at Multipath TCP Deployment in the Wild. In *Proceedings of the 6th International Workshop on Hot Topics in Planet-Scale Measurement*, HotPlanet '15, pages 7–12, New York, NY, USA, 2015. ACM.

[28] E. Nordmark and I. Gashinsky. Neighbor Unreachability Detection Is Too Impatient. RFC 7048 (Proposed Standard), January 2014.

[29] Vladimir Olteanu and Costin Raiciu. Datacenter scale load balancing for multipath transport. In *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMIddlebox '16, pages 20–25, New York, NY, USA, 2016. ACM.

[30] Christoph Paasch. *Improving Multipath TCP*. PhD thesis, UCLouvain / ICTEAM / EPL, November 2014.

[31] Christoph Paasch, Sebastien Barre, et al. Multipath TCP in the Linux Kernel. available from http://www.multipath-tcp.org.

[32] Christoph Paasch, Greg Greenway, and Alan Ford. Multipath TCP behind Layer-4 loadbalancers. Internet-Draft draft-paasch-mptcp-loadbalancer-00, Internet Engineering Task Force, September 2015. Work in Progress.

[33] Craig Partridge, Trevor Mendez, and Walter Milliken. Host anycasting service. RFC1546, 1993.

[34] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving Datacenter Performance and Robustness with Multipath TCP. In *ACM SIGCOMM 2011*, 2011.

[35] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? Designing and implementing a deployable Multipath TCP. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 29–29, Berkeley, CA, USA, 2012. USENIX Association.

[36] SungHoon Seo. KT's GiGA LTE: Commercial Mobile MPTCP Proxy service launch. https://www.ietf.org/proceedings/93/slides/slides-93-mptcp-3.pdf.

[37] Liang Zhu, Zi Hu, John Heidemann, Duane Wessels, Allison Mankin, and Nikita Somaiya. Connection-oriented DNS to improve privacy and security. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 171–186. IEEE, 2015.