An Overview of Multipath TCP

OLIVIER BONAVENTURE, MARK HANDLEY, AND COSTIN RAICIU

Olivier Bonaventure is a Professor at Catholic University of Louvain, Belgium. His research focus is primarily on Internet protocols. Bonaventure is the author of the open source textbook *Computer Networking: Principles, Protocols and Practice.* Olivier.Bonaventure@uclouvain.be

Mark Handley is Professor of Networked Systems at University College London. Handley is the author of 32 RFC Internet Standards documents, including those for the Session Initiation Protocol (SIP) and PIM Sparse Mode multicast routing. He was one of the co-inventors of distributed hash tables and was the recipient of the 2012 IEEE Internet Award. m.handley@cs.ucl.ac.uk

Costin Raiciu is currently an Assistant Professor at the Computer Science Department of University Politehnica of Bucharest. In 2011, Raiciu received a PhD from University College London, working under the supervision of Mark Handley and David Rosenblum. Raiciu's research interests are mainly in networking; in recent years, he has been working on designing and implementing Multipath TCP. costin.raiciu@cs.pub.ro

TCP has remained mostly unchanged for 20 years, even as its uses and the networks on which it runs have evolved. Multipath TCP is an evolution of TCP that allows it to run over multiple paths transparently to applications. In this article, we explain how Multipath TCP works, and why you should want to start using it.

Introduction and Motivation

The Transmission Control Protocol (TCP) is used by the vast majority of applications to transport their data reliably across the Internet. TCP was designed in the 1970s, and neither mobile devices nor computers with many network interfaces were an immediate design priority. On the other hand, the TCP designers knew that network links could fail, and they chose to decouple the network-layer protocols (Internet Protocol) from those of the transport layer (TCP) so that the network could reroute packets around failures without affecting TCP connections. This ability to reroute packets is largely due to the use of dynamic routing protocols, and their job is made much easier because they don't need to know anything about transport-layer connections.

Today's networks are multipath: mobile devices have multiple wireless interfaces, datacenters have many redundant paths between servers, and multihoming has become the norm for big server farms. Meanwhile, TCP is essentially a single-path protocol: when a TCP connection is established, the connection is bound to the IP addresses of the two communicating hosts. If one of these addresses changes, for whatever reason, the connection will fail. In fact, a TCP connection cannot even be load balanced across more than one path within the network, because this results in packet reordering, and TCP misinterprets this reordering as congestion and slows down.

This mismatch between today's multipath networks and TCP's single-path design creates tangible problems. For instance, if a smartphone's WiFi loses signal, the TCP connections associated with it stall; there is no way to migrate them to other working interfaces, such as 3G. This makes mobility a frustrating experience for users. Modern datacenters are another example: many paths are available between two endpoints, and multipath routing randomly picks one for a particular TCP connection. This can cause collisions where multiple flows get placed on the same link, thus hurting throughput to such an extent that average throughput is halved in some scenarios.

Multipath TCP (MPTCP) [1, 7, 8] is a major modification to TCP that allows multiple paths to be used simultaneously by a single transport connection. Multipath TCP circumvents the issues mentioned above and several others that affect TCP. Changing TCP to use multiple paths is not a new idea; it was originally proposed more than 15 years ago by Christian Huitema in the Internet Engineering Task Force (IETF), and there have been a half-dozen more proposals since then to similar effect. Multipath TCP draws on the experience gathered in previous work, and goes further to solve issues of fairness when competing with regular TCP and deployment issues as a result of middleboxes in today's Internet. The Multipath TCP protocol has recently been standardized by the IETF, and an implementation in the Linux kernel is available today [2].

Overview of MPTCP Operation

The design of Multipath TCP has been influenced by many requirements, but there are two that stand out: application compatibility and network compatibility. Application compatibility implies that applications that today run over TCP should work without any change over Multipath TCP. Next, Multipath TCP must operate over any Internet path where TCP operates.

Many paths on today's Internet include middleboxes, such as Network Address Translators, firewalls, and various kinds of transparent proxies. Unlike IP routers, all these devices do know about the TCP connections they forward and affect them in special ways. Designing TCP extensions that can safely traverse all these middleboxes has proven to be challenging [4].

Before diving into the details of Multipath TCP, let's recap the basic operation of normal TCP. A connection can be divided into three phases:

- connection establishment
- data transfer
- connection release

A TCP connection starts with a three-way handshake. To open a TCP connection, the client sends a SYN (for "synchronize") packet to the port on which the server is listening. The SYN packet contains the source port and initial sequence number chosen by the client, and it may contain TCP options that are used to negotiate the use of TCP extensions. The server replies with a SYN+ACK packet, acknowledging the SYN and providing the server's initial sequence number and the options that it supports. The client acknowledges the SYN+ACK, and the connection is now fully established. All subsequent packets in the connection use the IP addresses and ports used for the initial handshake. They compose the tuple that uniquely identifies the connection.

After the handshake, the client and the server can send data packets (*segments,* in TCP terminology). The sequence number is used to delineate the data in the different segments, reorder them, and detect losses. The TCP header also contains a cumulative acknowledgment, essentially a number that acknowledges received data by telling the sender which is the next byte expected by the receiver. Various techniques are used by TCP to retransmit the lost segments.

After the data transfer is over, the TCP connection must be closed. A TCP connection can be closed abruptly if one of the hosts sends a Reset (RST) packet, but the usual way to terminate a connection is by using FIN packets. These FIN packets

indicate the sequence number of the last byte sent. The connection is terminated after the FIN segments have been acknowledged in both directions.

Multipath TCP allows multiple *subflows* to be set up for a single MPTCP session. An MPTCP session starts with an initial subflow, which is similar to a regular TCP connection as described above. After the first MPTCP subflow is set up, additional subflows can be established. Each additional subflow also looks similar to a regular TCP connection, complete with SYN handshake and FIN tear-down, but rather than being a separate connection, the subflow is bound into an existing MPTCP session. Data for the connection can then be sent over any of the active subflows that has the capacity to take it.

To examine Multipath TCP in more detail, let's consider a simple scenario with a smartphone client and a single-homed server. The smartphone has two network interfaces: a WiFi interface and a 3G interface. Each has its own IP address. The server, being single-homed, has a single IP address. In this environment, Multipath TCP would allow an application on the smartphone to use a single TCP connection that can use both the WiFi and the 3G interfaces to communicate with the server. The application does not need to concern itself with which radio interface is working best at any instant; MPTCP handles that for the application. In fact, Multipath TCP can work when both endpoints are multihomed (in this case, subflows are opened between all pairs of "compatible" IP addresses), or even in the case when both endpoints are single homed (in this case, different subflows will use different port numbers and can be routed differently by multipath routing in the network).

Let's walk through the establishment of an MPTCP connection. Assume that the smartphone chooses its 3G interface to open the connection. First the smartphone sends a SYN segment to the server. This segment contains the MP_CAPABLE TCP option, indicating that the smartphone supports Multipath TCP. This option also contains a key that is chosen by the smartphone. The server replies with a SYN+ACK segment containing the MP_CAPABLE option and the key chosen by the server. The smartphone completes the handshake by sending an ACK segment.

At this point, the Multipath TCP connection is established and the client and server can exchange TCP segments via the 3G path. How could the smartphone also send data through this Multipath TCP connection over its WiFi interface?

Naively, the smartphone could simply send some of the packets over the WiFi interface; however, most ISPs will drop these packets, as they would have the source address of the 3G interface. Perhaps the client could tell the server the IP address of the WiFi interface and use that address when it sends over WiFi. Unfortunately, this will rarely work: firewalls and similar stateful middleboxes on the WiFi path expect to see a SYN packet before they see data packets. The only solution that will work reliably is to perform a full SYN handshake on the WiFi path before sending any packets that way, so this is what Multipath TCP does. This SYN handshake carries the MP_JOIN TCP option, providing enough information to the server that it can securely identify the correct connection with which to associate this additional subflow. The server replies with MP_JOIN in the SYN+ACK, and the new subflow is established.

An important point about Multipath TCP, especially in the context of smartphones, is that the set of subflows that are associated with a Multipath TCP connection is not fixed. Subflows can be dynamically added and removed from a Multipath TCP connection throughout its lifetime, without affecting the byte-

stream transported on behalf of the application. Multipath TCP also implements mechanisms that allow adding and removing new addresses even when an endpoint operates behind a NAT, but we will not detail them here. If the smartphone moves to another WiFi network, it will receive a new IP address. At that time, it will open a new subflow using its newly allocated address and tell the server that its old address is not usable anymore. The server will now send data towards the new address. These options allow smartphones to easily move through different wireless connections without breaking their Multipath TCP connections [6].

Assume now that two subflows have been established over WiFi and 3G; the smartphone can send and receive data segments over both. Just like TCP, Multipath TCP provides a bytestream service to the application. In fact, standard applications can function over MPTCP without being aware of it—MPTCP provides the same socket interface as TCP.

Because the two paths will often have different delay characteristics, the data segments sent over the two subflows will not be received in order. Regular TCP uses the sequence number in each TCP packet header to put data back into the original order. A simple solution for Multipath TCP would be to reuse this sequence number as is. Unfortunately, this simple solution would create problems with some existing middleboxes, such as firewalls. On each path, a middlebox would only see half of the packets, so it would observe many gaps in the TCP sequence space. Measurements indicate that some middleboxes react in strange ways when faced with gaps in TCP sequence numbers. Some discard the out-of-sequence segments, whereas others try to update the TCP acknowledgments to "recover" some of the gaps. With such middleboxes on a path, Multipath TCP cannot safely send TCP segments with gaps in the TCP sequence number space. On the other hand, Multipath TCP also cannot send every data segment over all subflows; that would be a waste of resources.

To deal with this problem, Multipath TCP uses its own sequence numbering space. Each segment sent by Multipath TCP contains two sequence numbers: the subflow sequence number inside the regular TCP header, and an additional data sequence number (DSN) carried inside a TCP option. This solution ensures that the segments sent on any given subflow have consecutive sequence numbers and do not upset middleboxes. Multipath TCP can then send some data sequence numbers on one path and the remainder on the other path; old middleboxes will ignore the DSN option, and it will be used by the Multipath TCP receiver to reorder the bytestream before it is given to the receiving application.

Congestion Control

One of the most important components in TCP is its congestion controller, which enables it to adapt its throughput dynamically in response to changing network conditions. To perform this functionality, each TCP sender maintains a congestion window, which governs the amount of packets that the sender can send without waiting for an acknowledgment. The congestion window is updated dynamically, growing linearly when there is no congestion and halved when packet loss occurs. TCP congestion control ensures fairness: when multiple connections utilize the same congested link, each of them will independently converge to the same average value of the congestion window.

What is the equivalent of TCP congestion control for multipath transport? To answer this question, we define three goals that multipath congestion control must obey. First, we want to ensure fairness to TCP. If several subflows of the same

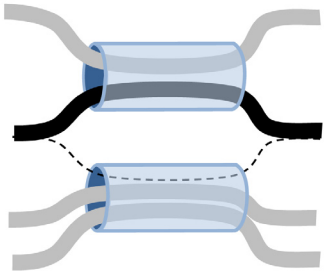

Figure 1: (Resource pooling) Two links, each with capacity 20 pkts/s. The top link is used by a single TCP connection, and the bottom link is used by two TCP connections. A Multipath TCP connection uses both links. Multipath TCP pushes most of its traffic onto the less-congested top link, making the two links behave like a resource pool of capacity 40 pkts/s. Capacity is divided equally, with each flow having throughput of 10 pkts/s.

MPTCP connection share a bottleneck link with other TCP connections, MPTCP should not get more throughput than TCP. Second, the performance of all the Multipath TCP subflows together should be at least that of regular TCP on any of the paths used by a Multipath TCP connection; this ensures that there is an incentive to deploy Multipath TCP. The third, and most important, goal is that Multipath TCP should prefer efficient paths, which means it should send more of its traffic on paths experiencing less congestion.

Intuitively, this last goal ensures wide-area load balancing of traffic: when a multipath connection is using two paths loaded unevenly (see Figure 1), the multipath transport will prefer the unloaded path and push most of its traffic there; this will decrease the load on the congested link and increase it on the less-congested one. If a large enough fraction of flows are multipath, the effect is that congestion will spread out evenly across collections of links, creating "resource pools," links that act together as if they are a single, larger-capacity link shared by all flows.

Multipath TCP congestion control achieves these goals through a series of simple changes to the standard TCP congestion control mechanism. Each subflow has its own congestion window that is halved when packets are lost, as in standard TCP. Resource pooling is implemented in the increase phase of congestion control; here Multipath TCP will allow less-congested subflows to increase proportionally more than congested ones. Finally, the total increase of Multipath TCP across all of its subflows is dynamically chosen in such a way that it achieves goals one and two above. More details can be found in [3, 4].

Implementation and Performance

Next, we briefly cover two of the most compelling use cases for Multipath TCP by showing a few evaluation results. We focus on mobile devices and datacenters, but note that Multipath TCP can also help in other scenarios. For example, multihomed Web servers can perform fine-grained load balancing across their uplinks, whereas dual-stack hosts can use both IPv4 and IPv6 within a single Multipath TCP connection.

The full Multipath TCP protocol has been implemented in the Linux kernel; its congestion controller has also been implemented in the ns2 and htsim network simulators. The results presented in this article are from the Linux kernel implementation, which is available for download at [2].

Our mobile measurements focus on a typical mode of operation in which the device is connected to WiFi, the connection goes down, and the phone switches to 3G. Our setup uses a Linux laptop connected to a WiFi and a 3G network, downloading a file using HTTP. We compare Multipath TCP with application-layer handover, where the application detects the loss of the interface, creates a new connection, and uses the HTTP range header to resume the download. Figure 2 shows the instantaneous throughputs for Multipath TCP and TCP with application-layer handover. The figure shows a smooth handover with Multipath TCP, as data keeps flowing despite the interface change. With application-layer handover, there is a downtime of three seconds where the transfer stops because the application takes time to detect the interface down event and ramp up 3G. In summary, Multipath TCP enables unmodified mobile applications to survive interface changes with little disruption. A more detailed discussion of the utilization of Multipath TCP in WiFi/3G environments may be found in [6].

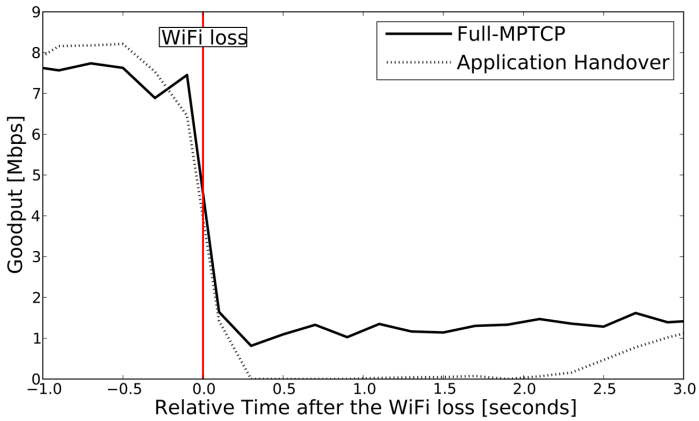

Figure 2: (Mobility) A mobile device is using both its WiFi and 3G interfaces, and then the WiFi interface fails. We plot the instantaneous throughputs of Multipath TCP and application-layer handover.

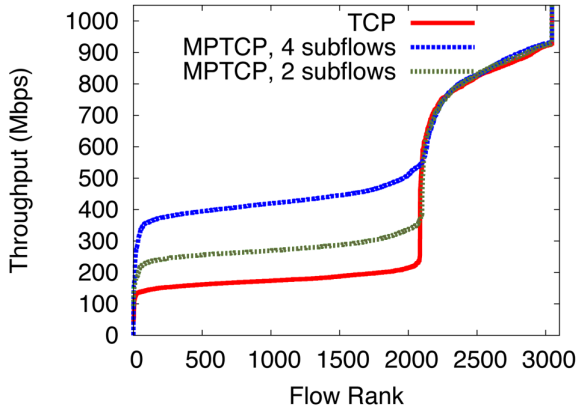

Figure 3: (Datacenter load balancing) This graph compares standard TCP with two- and four-flow MPTCP, when tested on an EC2 testbed with 40 instances. Each host uses iperf sequentially to all other hosts. We plot the performance of all flows (x axis) in increasing order of their throughputs (y axis).

Next, we show results from running Multipath TCP in the Amazon EC2 datacenter. Like most datacenters today, EC2 uses a redundant network topology, where many paths are available between any pair of endpoints, and where connections are placed randomly onto available paths. In EC2, we rented 40 machines (or instances) and ran the Multipath TCP kernel. We conducted a simple experiment in which every machine tested the throughput sequentially to every other machine using first TCP, then Multipath TCP with two and with four subflows. Figure 3 shows the sorted throughputs measured over 12 hours and demonstrates that Multipath TCP brings significant improvements compared to TCP in this scenario. Because the EC2 network is essentially a black-box to us, we cannot pinpoint the root cause for these improvements; however, we have performed a detailed analysis of the cases where Multipath TCP can help and why [5].

Conclusion

Multipath TCP is the most significant change to TCP in the past 20 years; it allows existing TCP applications to achieve better performance and robustness over today's networks, and it has been standardized at the IETF. The Linux kernel implementation shows that these benefits can be obtained in practice; however, as with any change to TCP, the deployment bar for Multipath TCP is high. Only time will tell whether the benefits Multipath TCP brings will outweigh the added complexity it produces in the endhost stacks.

References

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses": http://tools.ietf.org/html/draft-ietf-mptcp-multiaddressed-09.

[2] C. Paasch, S. Barre, J. Korkeaniemi, F. Duchene, G. Detal, et al., "MPTCP Linux Kernel Implementation": http://mptcp.info.ucl.ac.be.

[3] C. Raiciu, M. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, October 2011.

[4] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," USENIX NSDI, March 2011.

[5] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," ACM SIGCOMM 2011, Toronto, Canada, August 2011.

[6] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring Mobile/WiFi Handover with Multipath TCP," ACM SIGCOMM Workshop on Cellular Networks (Cellnet'12), 2012.

[7] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA), 2012.

[8] Multipath TCP resources: http://www.multipath-tcp.org.