Experimenting with Multipath TCP

Sébastien Barré, Olivier Bonaventure
Université catholique de Louvain
B-1348 Louvain-la-Neuve
firstname.lastname@uclouvain.be

Costin Raiciu, Mark Handley
University College of London
{c.raiciu,m.handley}@cs.ucl.ac.uk

ABSTRACT

It is becoming the norm for small mobile devices to have access to multiple technologies for connecting to the Internet. This gives researchers an increasing interest for solutions allowing to use efficiently several communication mediums. We propose a demonstration of our Multipath TCP implementation for Linux, that allows spreading a single TCP flow across multiple Internet paths, without requiring any change to applications. The demonstration will involve a real Internet communication with MPTCP, with simultaneous use of several paths, as well as a demonstration of MPTCP failover capability.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-communication networks—*Networks Protocols*

General Terms

Experimentation

1. INTRODUCTION

Today, most smartphones support at least 3G and 802.11, and so do tablet PCs like Apple's iPad. This has increased interest in using several access mediums in the same connection, so that it becomes possible to transparently change from one medium to another in case of failure. Further, using several paths *simultaneously* can improve end-to-end throughput.

The transport layer is the best place to implement multipath functionality because of the high amount of information it collects about each of the paths (delay/bandwidth estimation), and its knowledge of the application byte stream. The network may know path properties, but simply scattering packets of a single transport connection over multiple physical paths will typically reorder many packets, confusing the transport protocol and leading to very poor throughput. The apps could implement multipath, but such changes are not easy to get right. If we simply switched from TCP to multipath TCP while maintaing the reliable byte stream semantics, unmodified apps could benefit immediately.

The desire to use multiple paths at the transport layer is not new, and has been already the subject of several research papers, some based on TCP [7, 4, 9, 10], others based on modifications to the SCTP protocol [5, 1, 6]. However, to the best of our knowledge, none of these research efforts have produced a real-world implementation despite the importance of such an implementation for verifying how a multipath transport solution behaves in real usage scenarios. In particular the current research literature lacks evaluations of simultaneous use of real communications mediums, like 3G and Wifi. It also does not consider the impact of using an in-kernel implementation of the multipath protocol, compared to what is obtained through simulations.

There is currently fresh interest in making multipath TCP real, as the IETF has created a multipath working group.

We believe an in-kernel implementation of multipath can serve many purposes, chief among which is experimentation in the Internet. It helps show the benefits and drawbacks of using multipath TCP for real applications; it offers a better understanding of how multipath TCP competes with TCP in the Internet; it can highlight errors and gaps in protocol design; it helps test deployability and find unexpected middlebox behaviour. Finally it allows running realistic experiments (e.g. with link speeds of 1 Gbps and up).

Our contribution is to fill this gap by providing a functional implementation of MPTCP, the IETF multipath solution [3][1]. We propose to concretely demonstrate the potential of that solution by showing a media transfer (e.g. video streaming) over multipath TCP between a remote server and a Nokia N900 device.

In section 2, we outline the design of the multipath protocol we are implementing. Then we briefly describe the architecture of our implementation, and conclude with a description of the ongoing work.

2. MPTCP PROTOCOL

The single most important choice when designing a multipath protocol is the choice of the sequence numbering. In [7, 9, 5], one single sequence number space is used, with the consequence of huge reordering of sequence numbers at the receiver. Since reordering is normally mistaken as a packet drop indication, specific algorithms are needed to distinguish between normal multipath reordering and failures. Further, a single sequence number space makes it very difficult to tell which path(s) delivered a segment if the segment was sent redundantly (on more than one path).

To fix these problems, the MPTCP proposal uses a dual sequence number space, where each subflow has its own sequence space that identifies bytes within a subflow as if it were running alone. There is also a data (or connection level) sequence space [3], which allows reordering at the aggregate connection level. Each segment carries both subflow and data sequence numbers.

Another important design choice is the way to deal with shared bottlenecks. There is a fairness problem if several multipath flows share a bottleneck. [10] solves that problem by trying to avoid establishing several subflows across the same bottleneck, thanks to

SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
ACM 978-1-4503-0201-2/10/08.

[1]`http://inl.info.ucl.ac.be/mptcp`

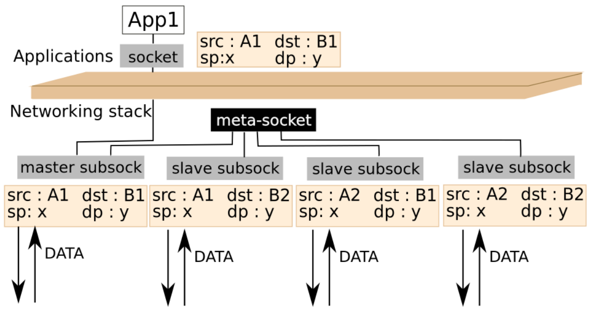

Figure 1: MPTCP architecture

an external tool. Other approaches simply ignore the problem. In MPTCP, congestion control is coupled across paths, so as to ensure fairness without needing to detect shared bottlenecks [8]. MPTCP performs flow control in aggregate (not on individual suflows).

A main goal of the MPTCP approach to multipath transport is that it must be deployable in the current Internet, without changing routers, middleboxes, or even NATs. For that reason each subflow looks to the network as a normal TCP flow, with the only difference that it carries new TCP options. Options are used to declare MPTCP support, exchange alternate addresses and other control messages. The overall MPTCP architecture and design choices are detailed in [2], and the protocol is specified in [3].

MPTCP works on the current Internet, as we will show in our demonstration. Only middleboxes that strip unknown options prevent the MPTCP negotiation; in this case MPTCP falls back to TCP.

3. MPTCP IMPLEMENTATION

The architecture of our implementation is depicted in figure 1. All legacy TCP applications directly benefit from the added multipath capability. When a new TCP flow is started multipath TCP adds the multipath capable option to the SYN packet. If the endpoint replies with a SYN/ACK containing the multipath capable option, this connection is multipath from now on.

Connection-specific information is held in a new structure at the connection-level, called meta-socket. This structure keeps multipath identifiers for the connection, the list of subflows associated to this connection, and connection-level reordering queues.

Initially there is a single TCP socket opened (the master socket), corresponding to the first subflow in the connection. When additional subflows are opened, new socket structures are created and associated to the meta-socket. The master socket is a special socket as it is the only connection to the application. Application writes to this socket are redirected to the meta-socket which segments the bytestream and decides which subflow should send each segment. Application reads from this socket are serviced from the meta-socket's receive buffer.

Data arriving on the subflows is serviced by the master and slave sockets (checking for in-order, in window sequence numbers, etc.), and passed to the meta-socket once it is in order at subflow level. Here the data is reordered according to the connection sequence number, which is carried in each TCP segment as an option. Retransmissions are driven only by the subflow sequence number; hence MPTCP avoids problems due to connection level reordering of packets.

Additional subflows are only opened after the initial handshake succeeds. The stack checks to see if it has multiple addresses that have routes to the destination; if so it will try to open subflows using currently unused addresses (in the picture this could be address $A2$). To get around NATs, addresses are also signalled explicitly to the remote end using TCP options.

Subflows are created with the usual three way handshake with SYN packets carrying a "Join" option and a connection identifier. SYN demultiplexing is done using this connection identifier, and not the destination port as in regular TCP.

The implementation allows opening subflows between different address pairs, or between the same address pairs but different ports. The latter can be used to leverage existing in-network multipath solutions such as Equal Cost Multipath (ECMP), allowing them to load balance at subflow granularity. Finally, our implementation is modular and it is easy to add support for new path management techniques that may become available.

Conclusion.

While simulations are useful to evaluate large scale behaviours of a protocol, a fundamental change like MPTCP requires careful evaluations of its behaviour in real world situations. An implementation can shed light on protocol behaviours and corner cases that cannot be observed with simulators. While several previous works have produced code that take benefit of multiple paths, this implementation is, to the best of our knowledge, the first one that works across the Internet (as opposed to local networks), and that allows unmodified applications to benefit. Our current work uses this implementation to analyse the behaviour of MPTCP in a number of real-life scenarios, including datacenters, mobile communications and multi-homed networks.

Acknowledgements.

The research results presented herein have received support from the Trilogy (http://www.trilogy-project.eu) research project (ICT-216372), partially funded by the European Community under its Seventh Framework Programme. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document.

4. REFERENCES

[1] A. A. E. Al, T. N. Saadawi, and M. J. Lee. LS-SCTP: a bandwidth aggregation technique for stream control transmission protocol. *Computer Communications*, 27(10):1012–1024, 2004.

[2] A. Ford, C. Raiciu, S. Barré, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. Internet draft, draft-ietf-mptcp-architecture-00.txt, Work in progress, February 2010.

[3] A. Ford, C. Raiciu, and M. Handley. TCP Extensions for Multipath Operation with Multiple Addresses. Internet draft, draft-ford-mptcp-multiaddressed-03.txt, Work in progress, March 2010.

[4] H.-Y. Hsieh and R. Sivakumar. pTCP: An End-to-End Transport Layer Protocol for Striped Connections. In *ICNP*, pages 24–33. IEEE Computer Society, 2002.

[5] J. R. Iyengar, P. D. Amer, and R. R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw.*, 14(5):951–964, 2006.

[6] J. Liao, J. Wang, and X. Zhu. cmpSCTP: An extension of SCTP to support concurrent multi-path transfer. *Communications*, 2008.

[7] L. Magalhaes and R. Kravets. Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts. In *ICNP*, pages 165–171. IEEE Computer Society, 2001.

[8] C. Raiciu and D. Wischik. Coupled Multipath-Aware Congestion Control. Internet draft, draft-raiciu-mptcp-congestion-01.txt, Work in progress, March 2010.

[9] K. Rojviboonchai, T. Osuga, and H. Aida. R-M/TCP: Protocol for Reliable Multi-Path Transport over the Internet. In *AINA*, pages 801–806. IEEE Computer Society, 2005.

[10] M. Zhang, J. Lai, and A. Krishnamurthy. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *USENIX 2004*, pages 99–112, 2004.