Towards a Multipath Transport Protocol for Serval

Torin Rudeen
Department of Computer Science, Princeton University
Under the advisement of Michael J. Freedman

Abstract

Multipath transport protocols provide practical and theoretical advantages over traditional single path protocols, enabling better congestion balancing and higher throughput without violating TCP fairness conditions. Multipath TCP (MPTCP) is the main ongoing effort to create a multipath transport protocol, but its efficiency and flexibility is hampered by the need to be entirely backwards compatible with a network designed for single path connections. What would a multipath transport protocol look like if it could be created without these restrictions?

Serval is an experimental network architecture designed to support service-centric networking. It supports seamless mobility across networks and interfaces, multipath communication, and a service-centered network abstraction by creating a Service Access Layer which sits on top of an unmodified Network Layer and below the Transport Layer. In this paper I draw on the ongoing research on MPTCP to design and implement a multipath transport protocol that takes advantage of the abstractions provided by Serval to offer improved efficiency, flexibility, and simplicity, while still playing fair with legacy TCP connections.

1 "Worse is Better" and the Need for Multipath

In his widely read article *The Rise of "Worse is Better"* Richard Gabriel offered a novel theory of why technically imperfect software systems often gain traction and widespread acceptance, even where more sophisticated systems fail [5]. Consider two systems: the first is simple to implement, but sacrifices completeness, consistency, and sometimes even correctness in service of this simplicity. The second is created to be completely correct, consistent, and complete, but sacrifices simplicity to this end. The first system (using what Gabriel calls the "New Jersey approach") is considerably "worse" than the second (the "MIT approach"): it is unreliable, limited in what it can do, and imposes extra burdens on the user. However, it will be created faster, ported more easily, and by the time the second system has been perfected the first will already have been widely adopted. Thus, Gabriel argues, the New Jersey approach will generally create more successful technologies than the MIT approach. C is a classic example of the New Jersey approach; Lisp is an example of the MIT approach. C was so successful not because it was technically superior to Lisp, but because it was much easier to write compilers for.

The Internet is one of the best examples of the New Jersey approach to design. Crucial to the widespread adoption of the Internet is the simplicity of the core protocols. Each piece of functionality is shunted as far up the network stack as possible: for example, instead of relying on the Link Layer or the Network Layer to guarantee reliable delivery, it's left to the Transport Layer. While this may be inefficient, because packets must be sent across the entire network when a loss occurs on any link, it makes switches and routers far simpler and cheaper to implement. Similarly, instead of explicitly allocating bandwidth to each connection, as in telephone networks, the Internet uses statistical multiplexing, relying on Transport level congestion avoidance algorithms to ensure an approximately fair allocation of resources. In fact, the well known end-to-end principle[13] can be viewed as an application of "Worse is Better" to networking: lower layers in the network stack should be as simple as possible, leaving all possible functionality to the higher layers.

While the "Worse is Better" approach may explain the Internet's widespread adoption, modern web applications demand reliablity, consistency, and scalability that the TCP/IP stack cannot provide on its own. This need has given rise to a wide array of technologies that address these issues from the Application Layer, including load balancers, network address translators (NATs), and content distribution networks (CDNs). For each of these systems, all or part of their functionality consists of taking care of problems which would have been addressed within the network, had the Internet had been designed with the MIT approach in mind. Now, this is not necessarily a bad thing: many of these technologies address needs which could not have been forseen when TCP/IP was designed, and solving these problems at the Application Layer allows for flexibility and quick deployment.

However, as Wischik argues in [14], these technologies all use variations of a single technique, *resource pooling*. The basic idea is to make multiple heterogenous resources behave like a larger single resource, for improved efficiency, capacity, and reliability. For example, a load balancer spreads traffic across a number of servers, providing the abstraction of a single server with much greater procesessing power and bandwidth. The problem is that when this technique is implemented over and over in every part of the network, massive redundancy and complexity is created. Wischik states the problem as his *resource pooling principle*: “Resource pooling is such a powerful tool that designers at every part of the network will attempt to build their own load-shifting mechanisms. A network architecture is effective overall, only if these mechanisms do not conflict with each other.”[14]

Wischik’s key insight that motivates the development of a multipath transport protocol is this: many forms of resource pooling currently performed by diverse Application Layer technologies can be handled elegantly and efficiently by a multipath-capable Transport Layer. When a single connection can simultaneously utilize multiple links, it is effectively pooling the capacity of all of these links. If one link fails, traffic can simply be sent over the others. When one link is heavily congested, instead of backing off transmission altogether, more traffic can be shifted to uncongested links. By having two paths go through different ISPs, one can even reap the benefits of multihoming without putting pressure on BGP.

2 Multipath TCP

For the past several years, there has been an ongoing effort to create a multipath transport protocol that can achieve resource pooling in the Transport Layer. The result of this effort is Multipath TCP (MPTCP). Extensive research has gone into designing a congestion control algorithm for MPTCP that balances congestion and improves throughput while still being fair to legacy TCP connections[9][12][15][16]. Furthermore, MPTCP has been shown to work in practice: there is an open source implementation of MPTCP in the Linux Kernel[11], and MPTCP is undergoing standardization by an IETF working group[6].

MPTCP is designed with the primary goal of backwards compatibility with a network designed around single path connections[6]. This goal dictates many of the design choices around MPTCP. For example, MPTCP uses a layered approach to sequence numbers, where every packet is assigned a flow-level sequence number and a connection-level sequence number. Flow sequence numbers are put in the TCP header, while a mapping from the per-flow sequence number space to the connection-level sequence number space is established using TCP options[11]. This makes it so that a single flow of an MPTCP connection appears to be a legacy TCP connection from the perspective of the network, ensuring compatibility with middleboxes. However, this comes at the cost of additional per-flow state, increased complexity of the protocol, and reduced flexibility.

These and other such tradeoffs are fact a result of the “Worse is Better” design of the Internet. Because the network chooses simplicity over providing desired functionality, users (individuals, corporations, and designers) take it upon themselves to create this functionality in a variety of ways, often violating the end-to-end argument and the layering abstraction. This is the basic reason for the proliferation of middleboxes, and for the hoops that MPTCP has to jump through in order to create a protocol deployable on today’s Internet.

There is undeniable value in creating a multipath transport protocol that works in the hostile environment of today’s Internet. However, there is also merit in asking what a multipath TCP look like if it were not subject to the constraints imposed by backwards compatibility. What gains could be made if a transport protocol could be designed on top of a more flexible network abstraction? In this project I have set out to answer just this question, building on top of the Serval network architecture in the hopes of creating a simpler, more efficient, and more flexible multipath transport protocol.

3 Overview of Serval

The network stack, in particular the TCP and IP protocols, was designed around a host-centric abstraction, optimized for an Internet of immobile computers which use only a single network interface. This is a far cry from today’s Internet, where computers are mobile, possess multiple heterogenous network interfaces, and wish to connect to web services which are split across thousands of computers around the world.

For example, IP routes packets based on their IP address, implicitly relying on the assumption that each party in communication is uniquely identified by a single IP address that does not change over time. Unfortunately, each component of this assumption fails in today’s Internet: IP addresses do not uniquely a party, due to the paucity of (IPv4) addresses and the resulting prevalance of NATs; a single party often has more than one IP address, whether because it is a single device with multiple interfaces (i.e. a smartphone with 4G and WiFi), or because it is a large web service run on thousands of different machines (i.e. Google); and the IP address of one party can change over time, whether due to virtual machine migration or to physical mobility. A wide variety of methods are used to deal with these problems, ranging from sophisticated application layer technologies (i.e. NATS and load balancers) to awkward non-solutions (such as restarting the connection whenever a party to a new IP address).

Serval is one attempt to design a network architecture more in line with the needs of today’s Internet. As described in [7], Serval replaces TCP/IP’s host-centric abstraction with a new service-centric abstraction. A connection is a link between two Service IDs, composed of one or more flows, each of which is a point-to-point link between two IP addresses. Instead of early-binding to an IP address using DNS, Serval-enabled hosts late-bind on Service IDs, which are mapped to the IP address of a target machine using anycast routing. Once a connection is established, either host can add additional flows or migrate existing flows without disrupting the

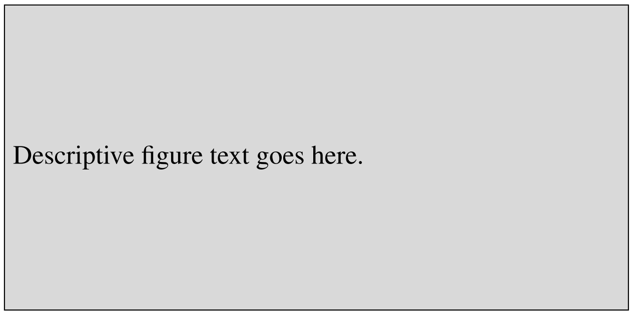

Figure 1: A placeholder caption for a placeholder figure.

ongoing connection, enabling multipath communication and seamless migration. All of this is made possible by the creation of a new Service Access Layer (SAL), sitting between the Network and Transport Layers. The SAL handles connection establishment and termination, flow creation, deletion, and migration, and

- Here

We have some related work in Section 6.

4 Design

5 Evaluation

6 Related Work

I like puppies [7][11][2][16][12][9][14][15][4][8][1][3][10].

7 Conclusion

Therefore, a duck.

References

[1] Allman, M., Paxson, V., and Blanton, E. TCP congestion control. RFC 5681 (Draft Standard), Sept. 2009.

[2] Arye, M., Nordstrom, E., Kiefer, R., Rexford, J., and Freedman, M. J. A formally-verified migration protocol for mobile, multi-homed hosts. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on* (2012), IEEE, pp. 1–12.

[3] Ford, A., Raiciu, C., Handley, M., Barre, S., and Iyengar, J. Architectural guidelines for Multipath TCP development. RFC 6182 (Informational), Mar. 2011.

[4] Ford, B., and Iyengar, J. Breaking up the transport logjam. In *ACM HotNets, October* (2008).

[5] Gabrel, R. The rise of "worse is better". http://www.jwz.org/doc/worse-is-better.html, 1989. Accessed: 2013-05-06.

[6] IETF. Multipath TCP working group charter. charter-ietf-mptcp-03, June 2012.

[7] Nordstrom, E., Shue, D., Gopalan, P., Kiefer, R., Arye, M., Ko, S. Y., Rexford, J., and Freedman, M. J. Serval: An end-host stack for service-centric networking. *Proc. 9th USENIX NSDI* (2012).

[8] Podmayersky, B. An incremental deployment strategy for Serval. Tech. Rep. TR-903-11, Princeton University Department of Computer Science, May 2011.

[9] Raiciu, C., Handley, M., and Wischik, D. Coupled congestion control for multipath transport protocols. *draft-ietf-mptcp-congestion-01 (work in progress)* (2011).

[10] Raiciu, C., Handley, M., and Wischik, D. Coupled congestion control for multipath transport protocols. RFC 6356 (Experimental), Oct. 2011.

[11] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and Handley, M. How hard can it be? Designing and implementing a deployable Multipath TCP. In *NSDI* (2012), vol. 12, pp. 29–29.

[12] Raiciu, C., Wischik, D., and Handley, M. Practical congestion control for multipath transport protocols. *University College London, London/United Kingdom, Tech. Rep* (2009).

[13] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS) 2*, 4 (1984), 277–288.

[14] Wischik, D., Handley, M., and Braun, M. B. The resource pooling principle. *ACM SIGCOMM Computer Communication Review 38*, 5 (2008), 47–52.

[15] Wischik, D., Handley, M., and Raiciu, C. Control of Multipath TCP and optimization of multipath routing in the Internet. In *Network Control and Optimization*. Springer, 2009, pp. 204–218.

[16] Wischik, D., Raiciu, C., Greenhalgh, A., and Handley, M. Design, implementation and evaluation of congestion control for Multipath TCP. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (2011), USENIX Association, pp. 8–8.

This paper represents my own work in accordance with University regulations.