Senior Project Proposal:
Extending Userspace Network Stacks onto Unordered TCP; An Experimentation Based Study

What is the need for uTCP? [6]

Transport layer protocols offer an interface between the network layer responsible for ferrying packets to their ultimate destination and the abstractions and APIs necessary for the application layer. TCP, as an example of a transport protocol, does this, as well as implementing certain functionalities that guarantee packet delivery and ordering of received packets. [8]

As the pre-eminent network protocol emphasizing reliability over speed, TCP became the foundation of the Internet from its early days. To increase speed, many connecting nodes now use information in the TCP header to speed up traffic. Further magnifying the dominance of the TCP protocol is the fact that some middleboxes either do not support any other protocol or drop any packets that do not appear to be TCP packets for security reasons.

Yet the benefits of a lighter weight protocol than TCP are clear. Some programs value reliability and the ordering of packets, but there are other applications such as Voice-over-IP and seamless network based file systems which are less concerned about reliability and more concerned about speed. For such programs, there exist lighter weight, unordered protocols such as the simple UDP and the more recent DCCP. Due to the concerns raised above about middleboxes, these programs suffer connectivity issues when using alternate network layer protocols. Yet these programs suffer a cost that cannot be reduced by relying on TCP.

The implementation of TCP in kernel code ensures that an application will have a complete logical data unit when requested. However, if a few or even one packet is missing, the other packets that exist must be held in a TCP buffer while a round trip is made to retrieve the missing packet.

Finally, as TCP has become more ubiquitous and entrenched in existing code, programmers of late have created additional protocols and transports that build on top of TCP and exist at the application rather than the operating system level. Building on top of TCP makes more sense when considering the ubiquity of TCP application hooks as well as the existence of TCP-only middleboxes. Many of these application level network-like protocols would benefit from being built upon an unordered protocol but use TCP for the aforementioned reasons. Using a transport layer protocol like uTCP would show speedups. On the other hand, some protocols are based on UDP and implement their own unique ordering scheme, but this comes with the connectivity issues that often plague UDP connections.
Existing transport-like userspace protocols

Many different protocols have been created to respond to the inefficiencies created by TCP-only connections. Other protocols attempt to address the intricacies of newer forms of streaming media, Javascript execution, and other modern day realities that did not exist at TCP’s formation. Since a majority of these protocols were created well after TCP was the de facto standard transport layer protocol, as well as after many operating systems had already solidified the protocols they would support within the kernel, these latter-day protocols are built in the application space.

Many newer protocols either write their own transport layer protocols or use UDP despite the connectivity and support issues that both of these approaches face. These protocols often implement their own forms of congestion control (as they cannot rely on that implemented in TCP) as well as other things traditionally implemented by the transport layer protocol. The widely extensible UDT protocol is built on top of UDP but also offers a selection of congestion control algorithms and a socket abstraction. [4] UDT implements a delay-based congestion control protocol for the purposes of Torrent based file transfer. [5] The CCI protocol aims to provide both a portable abstraction as well as high performance and reliability and is built on top of UDP or BSD sockets. [1]

Some protocols that implement transport layer features are built on top of a TCP transport layer. The most well known of these userspace protocols is the ubiquitous SSL/TLS protocol, but others also exist. [2] An older speed-focused protocol known as MUX was proposed in the late 90s that multiplexes numerous connections across a single TCP stream. [3] Google has also created its own SPDY protocol on top of TCP and SSL that also multiplexes and prioritizes certain packets. [7]

The contents of the project and project timeline

At an abstract level, the project aims to port some of these existing application level network-like protocols over to uTCP and the existing Minion framework, and testing various performance metrics.

At a more detailed level, the first step necessary is altering the existing uTCP implementation by properly disabling OS-level congestion control so as to provide an accurate environment for testing. This work will also be valuable for future applications of uTCP outside of the project. This should be complete after the first week of the project.

Then, the next step is to select a handful of candidate userspace protocols that will be expedient to port to the Minion framework yet also be good experimental candidates for uTCP. This should be completed by the second week of the project.

The next step is investigating the state of the current userspace libraries for Minion, and extending these libraries where required by the previously selected protocols. This should take the third and fourth weeks of the project.

Once the libraries are in order, the next step is the actual porting of the protocols to the Minion and uTCP architecture by altering the protocols’ source code where necessary. This should take the bulk of the time, taking from the fifth week and through the eighth week.
The final step is analyzing various performance metrics such as time and storage efficiency, and connection reliability among other interesting possible tests that could be run, both with the protocol in its original state and with Minion/uTCP replacing some de facto transport layer protocols. This should take the tenth week.

Once all the tests are run for the selected protocols in both original and uTCP implementations, a written report will be compiled to outline and present the results gathered from these tests. Compiling this paper should also take place in the tenth week.

Deliverables

The completed project will contain a written report of all experiments run on the chosen protocols and all code in the protocols and in the Minion framework altered for the purposes of this project.

Sources


