Linux-Integrated TCP Acceleration as a Service

Amanda Austin
University of Texas at Austin

Henrique Fingler
University of Texas at Austin

Timothy Stamler
University of Texas at Austin

ABSTRACT

Netowkring speeds have increased while CPU speeds have not. As a result, an increasing portion of packet processing time is spent in the kernel networking stack. To mitigate this effect, TCP Acceleration as a Service (TAS) splits TCP processing into a fast path and a slow path, both of which operate as userspace processes. In doing so, however, TAS loses some information about the network when compared to an in-kernel networking stack (e.g., firewalls, ARP tables). We present a Linux-integrated TAS that interfaces with Linux via a virtual network device. TAS duplicates some slow path packets to Linux via this virtual device, observes Linux's response, and then mimics that response. We show that this method can allow TAS to use Linux's information about the network while retaining the performance of TAS fast path operations.

ACM Reference Format:

Amanda Austin, Henrique Fingler, and Timothy Stamler. 2018. Linux-Integrated TCP Acceleration as a Service. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Networking speeds have become faster while CPUs have not, causing network packet processing efficiency to become important for datacenter networks. Datacenter applications continue to want high throughput and low latency access to the network along with the guarantees provided by TCP: lossless in-order delivery of packets, but this comes at the cost of consuming an increasing fraction of CPU processing resources. For example, nearly 70% of packet processing time for a simple echo server application is spent in the Linux networking stack [**?**].

To cope with this, many alternative TCP stacks have been proposed that seek to increase the efficiency of packet processing. TAS (TCP Acceleration as a Service) splits TCP packet processing into a fast path and a slow path. The fast path handles common data path operations such as handling in-order delivery of packets from established connections and generating acknowledgements. The slow path handles less common, control path operations such as connection management, congestion control, and connection timeouts. Both the fast path and slow path operate as user-level processes.

Implementing a TCP stack in userspace comes with a few drawbacks. Namely, the Linux TCP stack contains a lot of functionality and information about the network that is hard to replicate in userspace. Ideally, a userspace networking stack should make the same decisions about connection management, security, congestion, etc. as the Linux stack.

We present Linux-Integrated TCP Acceleration as a Service, an extension to TAS that interfaces with Linux for some slow path operations. The slow path now sends some packets to Linux, observes Linux's response, and mimics it. In this way, TAS can gain some of the information and functionality of the Linux TCP stack, such as firewall and network information (e.g., ARP tables), while retaining the performance of fast path operations.

Our paper makes the following contributions:

- We design and implement a method for the TAS to interface with Linux for some slow path operations (connection setup and teardown, ARP) in order to gain information and functionality.
- We evaluate our implementation and show that we introduce no overheads for fast path operations. Connection setup and ARP slow down significantly, but these operations are uncommon enough that they do not affect the throughput seen by the application.

In the remainder of our paper, we provide some background on TAS and virtual network devices in Section 2. We discuss the design and implementation of Linux-integrated TAS in Section 3. We evaluate our implementation in Section 4 and finally conclude and discuss future work in Section 5.

2 BACKGROUND

2.1 TCP acceleration as a service (TAS)

It is known that operating systems cause a large overhead on the network stack. Arrakis [**?**] showed that if we remove the kernel from the packet processing path and do everything from user-level, the time required to process one packet can be up to 16 times less.

Because of this and network speeds ramping up, there has been plenty of work implementing user-level network stacks ***cites*** with different ideas that try make general adoption easier.

One of these ideas is TCP acceleration as a service (TAS or SplitTCP ****TBD: check naming on intro/abstract***). The idea behind TAS is that the TCP stack can be divided into two parts: a slow path, which does the connection setup and teardown, and a fast path, where the data packets are sent and/or received. Figure 1 shows the general idea of TAS; the user's application use an unmodified POSIX API to talk to a user-level TCP library. The fast path can directly read from and write to the network card through DPDPK (**cite**); when data packets arrive or need to be sent out, the fast path directly communicates with the user-level library; when it gets packets that have to be handled by the slow path (such as SYN packets), the packet is offloaded to it through shared memory. Because the two paths are executed in different threads, the slow path doesn't affect the performance of the fast path.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

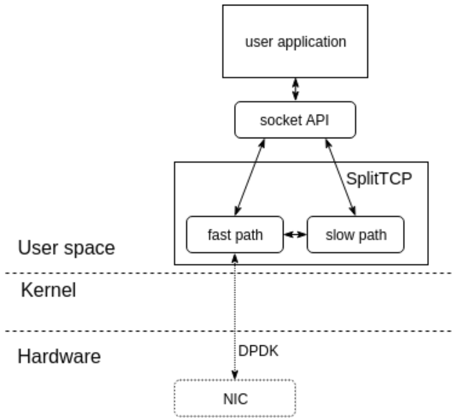

Figure 1: TBD.

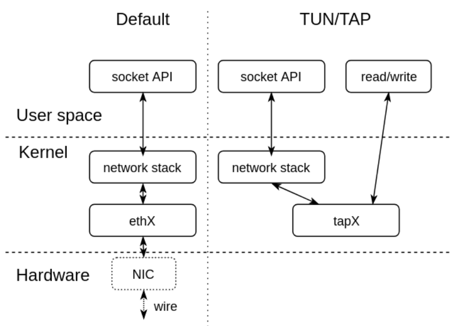

Figure 2: TBD.

This idea yields very good performance for the average case of a datacenter: connections are long-lived and set up pretty infrequently; thus data packets are predominant, making the fast path much more active than the slow path. One issue with this solution (and most other user-level stacks) is that all the desired kernel functionality are lost or need to be reimplemented, such as packet filtering, firewalls, control, etc.

2.2 TAP devices

One of the tools used in this work was TUN/TAP devices. These are virtual network devices that are purely software (hence the "virtual"). These devices mimic a physical NIC, the only difference is that instead of receiving/sending data to the outside through a wire/radio waves, it is done so to a buffer in kernel space that an application can read/write through the POSIX file API. Figure 2 has a basic representation of both concepts: when using a regular (wired) NIC, the kernel gets calls from user space, and sends/receives from a physical device. When using a TUN/TAP device, the data is read/written to a buffer in kernel, and an application in user space can capture and write raw network data to and from it, emulating a real outside connection.

The different between TUN and TAP is the network layer in which they work. TUN devices work in layer 3 (IP packets), while TAP devices work in layer 2 (ethernet frames).

	Latency (us)					
	Original TAS			Linux-Integrated TAS		
# Connections	Avg	99%	99.99%	Avg	99%	99.99%
1	60	70	79	61	62	78
2	63	65	97	62	64	82
4	65	68	105	65	89	118
8	71	79	147	70	78	151
16	68	77	144	69	76	146
32	72	97	170	74	104	153

Table 1: Average and tail latency of RPC Echo server microbenchmark

3 DESIGN

SplitTCP design

Linux integration Design goals -No modifications to fast path -Edge case operations can take as long as necessary

Compromises -Fast path doesn't forward ARP packets after SYN *Handle ARPs manually instead of using real ARPs -Seperate sequence numbers for Linux and Splittcp *Because Splittcp initializes fast path state on receiving SYN, three options: 1) Add artificial delay or add synchronization with tap thread to use Linux seq 2) Modify fast path to identify Linux seq and update state 3) keep separate sequence numbers for Linux and Splittcp (we do 3 but might want to switch to 1 eventually) -Generate ACK packets in slow path *Fast path doesn't forward ACK packets to slow path *Manually generate ACK packets

4 EVALUATION

In this section, we evaluate our implementation of Linux-Integrated TAS. Our evalutation seeks to answer the following questions:

- Do our changes affect the performance of the fast path?
- What is the performance of the slow path?

4.1 Evaluation Setup

To answer the above questions, we run a simple RPC echo server microbenchmark. A client sends a packet with a 64 byte payload to a server, which echos the packet back to the client. Both client and server are single threaded. The server machine is an Intel Xeon Gold 6138 system at 2.0 GHz with a 1G NIC. The client machine is an Intel Xeon E3-1225 v3 system at 3.3 GHz with a 1G NIC. Both client and server run Linux kernel 4.18.

4.2 Fast Path Performance

Table 1 shows the average and tail latencies of the RPC echo microbenchmark with TAS and Linux-Integrated TAS. The average-case latency of the fast path of Linux-Integrated TAS is within 3% of TAS. The performance at the tail varies a bit more between the two versions, but is within 25% of TAS in the worst case.

Figure 3 shows the throughput of the RPC echo server microbenchmark using the original TAS and Linux-Integrated TAS. The throughput of both versions are very similar.

Discussion. Our evaluation results show that the fast path performance of Linux-Integrated TAS is very similar to the fast path

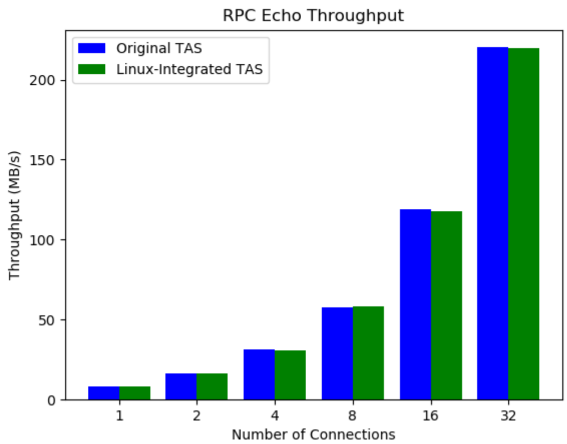

Figure 3: RPC echo throughput for a single threaded client and server.

performance of the original TAS. This result aligns with the fact that we made no changes to the fast path. Additionally, slow path operations such as connection setup and teardown happen infrequently enough that they do not affect the performance of the fast path.

4.3 Slow Path Performance

Our performance results show that the slow path performance is reduced drastically compared to the original TAS. Connection setup times were reported on the order of seconds. This is due to the fact that we now incur system call overheads on the slow path. We must wait for Linux to handle the packets we send to it before we can observe its response. For example, we must issue a blocking accept call to Linux to ensure that TAS does not prematurely move on to the next connection before observing Linux's response.

5 CONCLUSION

Integrating kernel functionality into TAS did not affect the fast path operations, which is where it gets most of its performance benefits. The slow path took a hit in performance due to proxying system calls on every slow path operation and having to read/write from a TAP device, which incurs plenty of mode switches. Because slow path operations are infrequent, we think that the tradeoff of slower connection setup and teardown is worth it to add powerful kernel network functionality such as packet filtering without having to reimplement it all in user space.