TCP Proxy Bypass: all the gain with no pain!

Giuseppe Siracusano
University of Rome, Tor Vergata

Roberto Bifulco
NEC Laboratories Europe

Stefano Salsano
University of Rome, Tor Vergata

CCS CONCEPTS

• **Networks** → **Network protocols**; *Transport protocols*; *Application layer protocols*; **Middle boxes / network appliances**;

KEYWORDS

TCP proxy; proxy offloading; SDN; NFV.

ACM Reference format:
Giuseppe Siracusano, Roberto Bifulco, and Stefano Salsano. 2017. TCP Proxy Bypass: all the gain with no pain! . In *Proceedings of SIGCOMM Posters and Demo '17, Los Angeles, CA, USA , August 22–24, 2017,* 3 pages.
https://doi.org/10.1145/3123878.3123916

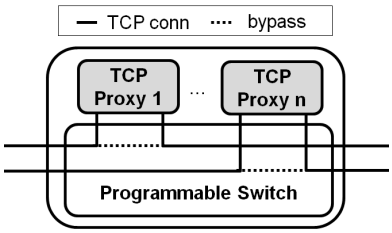


Figure 1: TCP proxy bypass

1 INTRODUCTION

TCP proxies are widely deployed in modern networks [9], and, sitting on the network connections' data path, their efficiency is critical to systems' cost and performance.

Regardless of its specific function, a TCP proxy typically performs a set of common operations. It terminates a client TCP connection, reads/modifies the application-layer header, and then opens a new TCP connection to a selected back-end server. The details of this process finally determine the actual implemented function. For instance, Level 7 load balancers (L7LBs) distribute load across back-end servers. This is the case of HAProxy[1], a L7LB used by many well known services (e.g., Alibaba, Imgur, Reddit, Stack Overflow, Tumblr, Vimeo). Similarly, Redirection Proxies are used to redirect traffic based on an independent DNS resolution of the Host field in the HTTP request. Yet another example are protocol optimization proxies used in cellular networks (e.g., AT&T, T-Mobile, Verizon, Sprint) to delay the initial TCP handshake to a server, until receiving an HTTP request [9]).

In many cases, the proxy is only required during the initial phases of a network connection, becoming just a relay during the later stages, until the connection is finally closed. For example, TCP proxies reading the HTTP request header may require to access only the first few packets of a connection. In fact, many TCP connections carry just a single HTTP request [1]. Furthermore, with the increase of TLS encrypted traffic, only the first few frames of a TCP connection are not encrypted and can be actually accessed by the proxy [6].

[1]http://www.haproxy.org/

SIGCOMM Posters and Demo '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08…$15.00
https://doi.org/10.1145/3123878.3123916

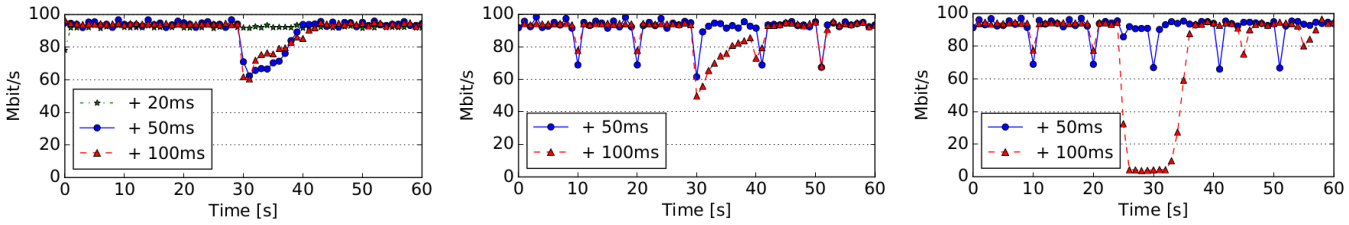
In this paper, we focus on these cases and try to answer the following question: *can established connections be offloaded from the TCP proxy*? Our goal is to save precious resources by transparently removing the TCP proxy from the data path, when the proxy's operations are limited to relaying packets. Notice we do not consider proxies performing protocol optimization, such as WAN accelerators. These proxies modify the TCP protocol behavior, e.g., the congestion control algorithm, and/or modify the data frames, e.g., performing compression, making them unsuitable for the offload.

Performing a TCP connection offload implies fulfilling two requirements. First, the offload should be performed without modifying the end-hosts of a connection, i.e., the client and the server. Second, the offload should not negatively affect the connection performance. Indeed, the main issue is in the need to join together two TCP connections with their specific per-connection states. To this end, our work provides a twofold contribution. First, we identify the minimum set of operations required to implement the joining of two TCP connections fulfilling the above requirements. Then, we show that such operations can be easily supported by already deployed infrastructure elements, such as P4 programmable switches [2], as shown in Fig.1. We implemented a proof-of-concept of our technique, named TCP proxy bypass, using PISCES [7], a P4 programmable software switch based on OVS, and extending Miniproxy [8], a unikernel implementation of a TCP proxy.

2 TCP PROXY BYPASS

TCP Proxy Bypass is implemented using a modified TCP proxy and a programmable software switch. The proxy selects the TCP connections to be offloaded, and performs the corresponding offloading operations. The switch, previously used to forward incoming/outgoing traffic to/from the proxy, performs the packet forwarding for offloaded connections.

When the proxy decides that a connection has to be offloaded, it performs three operations. First, the proxy defines network address and port translation rules to rewrite the client-side and server-side connection's packet headers. The operation is very similar to a NAPT (Network address and port translation) and is required to remove the proxy from the path. Then, the proxy waits until there



(a) Cubic (b) BBR (ProbeRTT) (c) BBR (ProbeBW)

Figure 2: Effects of delay injection on TCP congestion control

are no in-flights packets on both the client- and server-side connections[2]. In fact, once a connection is offloaded from the proxy, any potential re-transmissions cannot be handled anymore. In effect, the proxy monitors the status of the receive and transmission buffers for both the client- and server-side connections. When all the buffers are empty, the offload is finally performed. In this stage, the proxy computes an offset between the two connections sequence and ACK numbers. Depending on the packets direction, adding or subtracting such offset from the corresponding packet's header fields ensures the client and the server will not find a sequence number/ACK number mismatch, once the proxy is removed. The offload is completed by installing a pair of forwarding entries, one per direction, in a P4 programmable switch. The entries perform both the NAPT and offset add/sub operations.

TCP options TCP Proxy Bypass requires the proxy to negotiate the same set of options on both the client- and server-side connections. In practice, a proxy could cache the set of options supported by a server and then negotiate the same set of options with the next client asking for a connection to that server. Note that for some TCP options also the value of the option should be propagated from the proxy to the client, e.g., the window scaling value.

Impact on congestion control Joining the client- and server-side connections may impact the TCP congestion control loop. Most notably, the removal of the proxy corresponds to a sudden variation of the perceived end-to-end round-trip time (RTT). We evaluated the effect on Cubic [5] and BBR [3] algorithms, running a TCP flow for 60s over a 100Mbps link. During the connection, we add additional delay on the link, summing up to a total RTT value of 100ms for all the cases. With Cubic (Fig.2a), an RTT increase bigger than 20ms causes a sudden throughput drop (at T=30s). The effect is due to expiring re-trasmission timeouts, whose value is set according to the measured RTT. In Fig.2b and Fig.2c we see the BBR test results. Recall that BBR detects congestion estimating the RTT and the bottleneck bandwidth (BW). Briefly, it uses two different states, ProbeBW and ProbeRTT. During ProbeRTT, BBR reduces the in-flight packets to drain the queues on the end-to-end path, and estimates the RTT. The ProbeRTT state lasts for 200ms and is triggered periodically (every 10s), unless there are silence or low rate periods. Fig.2b shows the throughput when the delay is added a during the ProbeRTT state (T=30), while Fig.2c shows the throughput when the delay is added a during the ProbeBW state (T=25). In both cases a RTT increase smaller than 50ms does not affect the connection throughput. Larger variations, e.g., 100ms, introduce instead a throughput drop, as the connection reenters the Startup phase, or in the case of Fig.2c it enters in Drain state and then performs the Startup.

To avoid throughput drops, we ensure the proxy introduces a gradual synthetic delay, e.g., in steps of 20ms, before performing the connection offload.

Implementation We implemented TCP Proxy Bypass extending a unikernel operating system [8] to expose additional information at the socket level, e.g., TCP sequence numbers, and to configure the programmable switch. We used PISCES [7] to implement a P4 programmable switch. A P4 program describes a switch pipeline that can read and modify the TCP header fields (e.g., TCP sequence numbers). While we could have used P4 also to describe the required sequence number modification action, we used it only to parse the TCP header fields. For performance reasons the sequence number translation is implemented as a custom action directly inside the OpenvSwitch code. We believe the poor performance to be a limitation of the prototypical version of PISCES we used for the test.

System load We tested the system using a test deployment similar to what we would expect to be a typical modern proxy deployment scenario. That is, we run the proxy as a virtual machine, using Xen 4.4 and PISCES as software switch, on a server equipped with an Intel Xeon Ivy Bridge CPU@3.4GHz, 16GB RAM, with a dual port Intel x450 10Gb NIC. A similar server, connected back-to-back, generates 10Gbps line rate traffic using 100 parallel TCP connections.

We assign one core to PISCES and one to the proxy, achieving 10Gbps line rate. Then, we apply TCP Proxy Bypass to the 100 connections. Also in this case the line rate is easily reached, but this time using just PISCES running on a single core to forward the traffic. With all the connections offloaded, the proxy's CPU core stays idle, while the switch resource consumption does not increase. More specifically, the switch is actually experiencing lower load since it just forwards packets between the server's physical interfaces, instead of performing the additional forwarding step required to deliver packets to the proxy, as it is the case when the connections are not offloaded.

3 ACKNOWLEDGMENTS

This paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 671566 ("Superfluidity"). This paper reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

[2]In case of web traffic, this is easily the case since the communications often have silence periods [4]).

REFERENCES

[1] Mohammad Al-Fares et al. 2011. Overclocking the Yahoo!: CDN for Faster Web Page Loads *(IMC '11)*. ACM, New York, NY, USA. https://doi.org/10.1145/2068816.2068869

[2] Pat Bosshart et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR* 44, 3 (2014).

[3] Neal Cardwell et al. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5 (2016), 50.

[4] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2008. Characterizing user sessions on youtube. In *Electronic Imaging*.

[5] Sangtae Ha et al. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* 42, 5 (2008).

[6] David Naylor et al. 2014. The cost of the S in HTTPS. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM.

[7] Muhammad Shahbaz et al. 2016. Pisces: A programmable, protocol- independent software switch. In *ACM SIGCOMM*.

[8] Giuseppe Siracusano et al. 2016. On the Fly TCP Acceleration with Miniproxy. In *ACM SIGCOMM HotMiddlebox*.

[9] Xing Xu et al. 2015. Investigating transparent web proxies in cellular networks. In *PAM*. Springer.