Poster: Using BPF to Measure Latency at High Link Speeds

Simon Sundberg
Karlstad University
Karlstad, Sweden
simon.sundberg@kau.se

Anna Brunstrom
Karlstad University
Karlstad, Sweden
anna.brunstrom@kau.se

ACM Reference Format:
Simon Sundberg and Anna Brunstrom. 2021. Poster: Using BPF to Measure Latency at High Link Speeds. In *Proceedings of The 2021 Internet Measurement Conference (IMC '21).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

While the throughput on the Internet has continuously increased over the last decade, the latency has not seen a similar improvement [4]. However, for many applications the latency is more important for the quality of experience, and latency measurements have proven useful for better understanding network performance and identifying issues. Being able to monitor network Round Trip Times (RTT) is therefore of great use.

One common approach to measure RTTs is to actively send out a probe packet and measure the time until a reply is received. This approach has been standardized in the ICMP protocol with the echo-reply messages which is used by the common ping utility. Similar active approaches have also been extended to other protocols through tools such as hping [6] and IRTT [3]. However, active network monitoring solutions have some considerable drawbacks.

(1) They introduce additional network traffic and may therefore affect the normal network traffic.
(2) They need to send probes between each pair of nodes of interest, which is cumbersome and does not scale well to large complex networks.
(3) They report the latency experienced by the probe packets, not the normal network traffic. The probes may experience different latencies than normal network traffic due to for example queuing delays, active queue management and load balancers.

Passive latency monitoring techniques avoid these issues by inspecting existing network traffic instead of injecting additional probes. They parse packet headers to match replies against sent packets, and measure the time between them. For TCP traffic this can be achieved by matching sequence- and acknowledgement numbers or using the TCP timestamp option [8]. Kathleen Nichols implemented Passive Ping, PPing [5], based on the latter approach. However, PPing uses traditional packet capturing, where packets are copied to and parsed in user space. This is very resource demanding and does not scale well to the high packet rates encountered in modern multi-Gbps links.

To cope with the high packet rates, recent work [1, 2, 7] have therefore implemented passive RTT monitoring for programmable switches using P4. Although these monitoring tools show promising results, and hardware supporting P4 is becoming more common, we still recognize the need for a general latency monitoring tool that does not require special hardware support. We therefore propose an extended PPing, ePPing, which leverges BPF to parse the packets directly in kernel space. Parsing the packets in BPF avoids the large overhead associated with copying packets to user space, which should allow ePPing to scale to much higher line rates than PPing. Our ePPing tool works on any machine running (a sufficiently recent version of) Linux, either an end host or some middlebox such as a router or firewall. Furthermore, ePPing can employ configurable sampling to further reduce overhead at high line rates, can report RTTs for ICMP echo-reply messages in addition to TCP traffic, and has some additional features such as JSON output.

2 BPF-BASED PPING DESIGN

The general design of ePPing is quite similar to that of PPing. The main difference is that ePPing has been split up into two major components, a user space part and a BPF program that runs in kernel space, as illustrated in Figure 1. The BPF program parses the packets, match reply packets against sent packets and calculate the RTTs. The user space component mainly loads the BPF program, causing it to be triggered on

IMC '21, November 2–4, Online

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

each received and transmitted packet, and then prints out the RTTs reported by the BPF program.

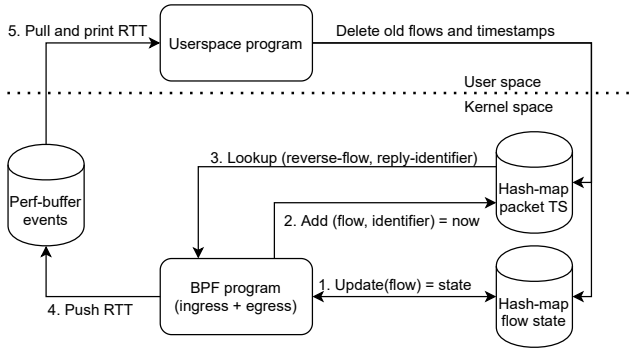

Figure 1: Design overview of the ePPing tool.

The logic for calculating the RTTs is essentially the same as for PPing. The packets are parsed for a pseudo-unique identifier that can be used to match against a future reply. If such an identifier is found, it is saved together with a timestamp in a hashmap. If a packet can then be matched against a previously stored identifier in the reverse flow, the RTT is calculated as the difference between the current time and the stored timestamp.

For TCP traffic, the TCP timestamp option TSval is used as the packet identifier, which is matched against the echoed TSecr value. Using TCP timestamps avoids some of the complexities with tracking sequence-and acknowledgement numbers in the presence of packet loss [8]. However, TCP timestamps are not necessarily unique for every packet, therefore only the first instance of a specific TSval is timestamped and matched against the first matching TSecr. For ICMP echo, ePPing uses the echo identifier as a port number and the echo sequence number as a packet identifier. In the future, ePPing could possibly be extended to additionally work with for example TCP seq/ACK numbers, the QUIC spinbit and DNS queries.

3 INITIAL RESULTS

We are currently in the process of evaluating the performance of ePPing, as well as continuously enhancing its functionality and design. An initial performance result that illustrate the potential of using BPF to parse the packets directly in kernel space, as opposed to copying the packet to user space, is shown in Figure 2.

The results were obtained using a setup with three virtual machines in a string topology. A single bulk TCP flow was uploaded from the VM in one end to the VM in the other end using iperf3. PPing or ePPing was set up on a VM in the middle, which was forwarding the traffic between both end

	min	median	mean	max
baseline	0.00	103.48	99.75	117.32
PPing	2.00	203.08	195.63	210.96
ePPing	0.00	106.52	103.81	115.08

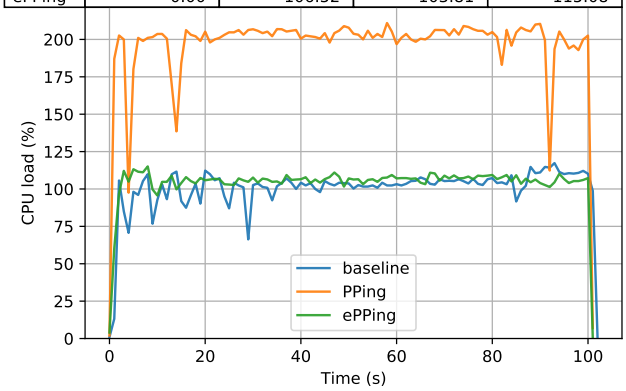

Figure 2: Initial performance results of the CPU overhead of ePPing compared to PPing and a baseline without any latency monitoring.

hosts. Figure 2 shows the CPU utilization on the forwarding VM with PPing, ePPing or without any latency monitoring, respectively. As can be seen, PPing introduces a large overhead compared to the setup with no latency monitoring, whereas the overhead from ePPing is relatively small.

REFERENCES

[1] Xiaoqi Chen, Hyojoon Kim, Javed M. Aman, Willie Chang, Mack Lee, and Jennifer Rexford. 2020. Measuring TCP Round-Trip Time in the Data Plane. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure (SPIN '20)*. ACM, New York, NY, USA, 35–41. https://doi.org/10.1145/3405669.3405823

[2] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. 2017. Dapper: Data Plane Performance Diagnosis of TCP. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. ACM, New York, NY, USA, 61–74. https://doi.org/10.1145/3050220.3050228

[3] Pete Heist. 2021. IRTT (Isochronous Round-Trip Tester). Retrieved September 22, 2021 from https://github.com/heistp/irtt

[4] Toke Høiland-Jørgensen, Bengt Ahlgren, Per Hurtig, and Anna Brunstrom. 2016. Measuring Latency Variation in the Internet. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 473–480. https://doi.org/10.1145/2999572.2999603

[5] Kathleen Nichols. 2018. *Pping (Pollere Passive Ping)*. Retrieved September 21, 2021 from https://github.com/pollere/pping

[6] Salvatore Sanfilippo. 2006. Hping - Active Network Security Tool. Retrieved September 22, 2021 from http://www.hping.org/

[7] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. 2021. Fine-Grained RTT Monitoring inside the Network. In *Measuring Network Quality for End-Users 2021*. IAB. https://www.iab.org/wp-content/IAB-uploads/2021/09/Fine-Grained_RTT_Monitoring_Inside_the_Network.pdf

[8] Stephen D. Strowes. 2013. Passively Measuring TCP Round-Trip Times. *Commun. ACM* 56, 10 (Oct. 2013), 57–64. https://doi.org/10.1145/2507771.2507781