

Setting Up a High-Speed TCP Benchmarking Environment—Lessons Learned

Thomas Lukaseder, Leonard Bradatsch, Benjamin Erb, Frank Kargl
 Institute of Distributed Systems
 Ulm University, Germany
 {firstname}.{lastname}@uni-ulm.de

Abstract—There are many high-speed TCP variants with different congestion control algorithms, which are designed for specific settings or use cases. Distinct features of these algorithms are meant to optimize different aspects of network performance, and the choice of TCP variant strongly influences application performance. However, setting up tests to help with the decision of which variant to use can be problematic, as many systems are not designed to deal with high bandwidths, such as 10 Gbps or more. This paper provides an overview of pitfalls and challenges of realistic network analysis to help in the decision making process.

I. INTRODUCTION

The evaluation of transmission layer protocols is a non-trivial endeavor. The protocols are designed to provide a number of services for end-to-end communication between processes, but their actual behaviors heavily depend on characteristics of the underlying network. By avoiding congestion of the network, the reactive behavior is an inherent feature of many transmission layer protocols such as TCP and its different congestion control algorithms. When artificially testing transmission layer protocols, realistic network properties must be harmonized with the appropriate degree of isolation in order to provide representative yet comparable results. The choice of tunable testing parameters such as latency and drop rate behavior of the network, the patterns and workloads of network traffic, and the congestion control algorithms to be tested constitute the overall framework for test runs. Based on the algorithm properties to be evaluated, various metrics must then be measured during the tests. Furthermore, a structured analysis of results is necessary to assess the protocols after the test runs and to gain insights from the test data. As already observed by Wei et al. [10], an evaluation of TCP congestion control algorithms requires rigorous planning and strict testing procedures.

When we were evaluating the performance of TCP variants in a 10G Ethernet network, we discovered a number of additional testing challenges posed by high-speed networks. As not all parts of the network stack scale equally well to new bandwidths, 10G environments generally force the testers to put more emphasis on hardware components of the setup. For instance, prior tuning of network hardware and NICs [3] becomes mandatory in order to achieve full bandwidths.

In this paper, we share our experience of TCP benchmarking in high-speed networks and provide a list of lessons that we have learned when setting up a 10G testing environment for

TCP variants. Our tests and the results of the performance measurements—including recommendations on which TCP congestion control algorithm to use—can be found in [5].

II. METHODOLOGY

Several decisions have to be made when building a test setup for TCP such as choice of the network topology or which software tools to use. In addition, appropriate criteria have to be chosen to evaluate the properties of TCP. These points are crucial to attain a sensible test setup and test procedure.

A. Criteria

The criteria most frequently used to evaluate TCP congestion control algorithms are the ones described by Li et al. [4]. *Responsiveness* describes the ability of the algorithms to recover quickly from random packet loss by measuring the average throughput at different drop probabilities for a packet in the network. *Efficiency* concerns the utilization of the network resources. A protocol is efficient if the available bandwidth at the bottleneck links is utilized as fully as possible in spite of existing network impairments. *Fairness* is typically determined with Jain's fairness index [2] and describes how equally the bandwidth can be utilized by competing TCP flows. An index of $\frac{1}{n}^{th}$ for example means complete utilization by one link out of n links, an index of 1 means that every link uses exactly $\frac{1}{n}^{th}$ of the utilized bandwidth. *Backwards compatibility* describes a fairness measurement not only with contemporary TCP variants, but with at least one legacy flow. The metric then is usually computed similarly to fairness. In addition to fairness, it is also important to note which of the tested variants dominates and utilizes the larger amount of bandwidth. *Convergence time* is defined as the time required for the short-term average throughput of a new flow to achieve the long-term average throughput.

B. Test Setup

The standard test setup for TCP performance tests is the dumbbell topology (Figure 1a). The topology consists of the same number of sender and receiver nodes. All sender nodes are connected to a switch, which forwards the incoming flows over one link to another switch where they are again separated and forwarded to the receiver nodes. The link between the two switches constitutes a bottleneck, which leads to congestion

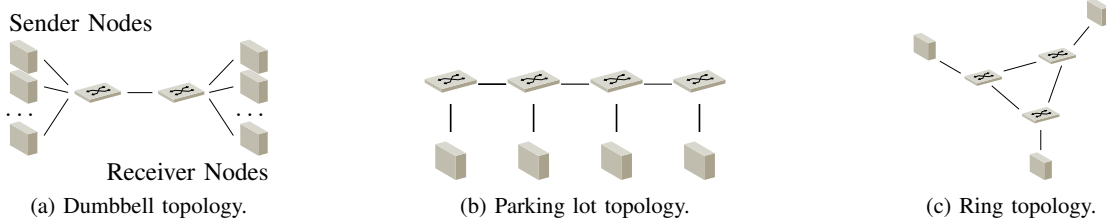


Fig. 1: Different test network topologies.

and therefore to observability of congestion control algorithms. The aforementioned criteria need different numbers of TCP flows in the network. To evaluate responsiveness and efficiency, only one sender and receiver are active. When evaluating the other metrics, two or more sender and receiver nodes are necessary. Besides the dumbbell topology, there are several other possible topologies, such as the parking lot (Figure 1b) or the ring topology (Figure 1c). The parking lot topology is used to simulate multiple bottleneck links, for example a server close to the backbone and clients with different distance to the server. The ring topology is used to observe protocols with asymmetric routing. Wei et al. [10] gives a detailed overview of the existing topologies and when each of those should be used.

The specific test configuration depends on the use case. Traffic patterns can differ greatly depending on the observed network. For example, one short request to a webserver can lead to subsequent requests to other backend (e.g., database) servers. These traffic bursts have to be simulated differently than constant flows that can be observed in ISPs' backbone networks. Therefore, some criteria such as best fitting network topology, the number of flows concurrently in the network, which TCP variants should be tested, and if the sender and receiver nodes should use the same variants should be considered carefully. Additional background noise that emulates the typical environment of the specific use case can also be an option. PCAP files and programs such as *tcpreplay*¹ can be used to induce background traffic. The protocol mix of the use case network has to be taken into account, i.e., the ratio of UDP to TCP or other protocols in the network, how different devices interact, or if the traffic is equal in both directions or tends to be more unidirectional. HDDs are too slow to enable *tcpreplay* to send at 10 Gbps and—depending on the specific hardware—this is also true for many SSDs. Even when the SSD is capable of this high speed, read and write operations by the operating system or other processes can interfere with the read operations of *tcpreplay* and consequently confound the test results. Therefore it is advisable to save the PCAP files on a RAM disk.

There are a large number of TCP congestion control algorithms. Many improved versions of TCP congestion control algorithms were developed, following the now obsolete TCP Tahoe. As the usage of TCP Reno phases out, it still plays its role in many legacy networks and should still be considered as

a baseline for comparison to the newer TCP variants. Another point to consider is the lack of availability of some algorithms for different operating systems. Compound TCP for example is only available on Windows. When comparing this variant with others, the tester is forced to use different operating systems for the comparison. Therefore, it can not be ensured that difference in performance might not stem from different kernel or driver behavior. Another point to consider is that some algorithms are developed with downwards compatibility to Reno in mind while others do not work well with legacy TCP. Delay-based congestion control algorithms such as TCP Vegas might perform well in enclosed network environments but cannot be used when loss-based TCP variants are present [8]. A list of variants that explicitly aim for downwards compatibility should always include Scalable TCP, HS-TCP, BIC, CUBIC, and H-TCP.

Regarding network performance, the presence of flow control in switches represents another issue. Flow control means that switches send pause packets to senders if congestion is imminent. For the evaluation of an existing network environment, this setting should not be changed. However, it should be evaluated how differences in the client settings affects performance of the network. For the dedicated evaluation of TCP congestion control algorithms, flow control should be turned off as it interferes with the congestion control and distorts the results.

C. Test Procedure

To produce TCP flows with the desired bandwidth of 10 Gbps, *iperf3*²—a client-server-application to measure performance in networks—can be used. Version 3 additionally supports the detection of retransmissions during a connection. An example command to start a flow with Reno TCP Congestion Control would look like this:

```
# iperf3 -c 192.168.88.22 -i 10 -t 900 -C reno
```

Convergence time can also be measured:

```
# iperf3 -c 192.168.88.22 -i 0.1 -t 900 -C reno
```

The parameter *t* determines the test duration and *i* the time intervals for the calculation of the average bandwidth. *Iperf* determines the average sending rate over *i*. The option *C* lets the user chose the congestion control algorithm. On the server side, the program needs also to be started:

```
# iperf3 -s
```

Additionally, *iperf* provides the possibility to adjust several

¹<http://tcpreplay.appneta.com/>

²iperf.fr

TCP features such as the usage of the Nagle Algorithm (TCP no-delay) or the TCP maximum segment size. It is highly recommended to use scripts to automate testing. With *iperf* saving its data to JSON files and programs such as *SPSS*, *Matlab*, *gnuplot*, or *R* being completely scriptable, a completely automated toolchain for evaluation is possible.

III. PITFALLS & CHALLENGES

A. Kernel Settings & NIC Configurations

There are several options within the TCP specification that can be tweaked to accomplish higher throughput. It depends on the operating system how the settings can be changed and to what extent this is possible. We concentrate on the features of the Linux 3.16 kernel as it is both adaptable and widespread. The settings are changed using the *sysctl* command. The following settings have to be taken into consideration both on client and server side.

One very important setting concerns TCP Window Scaling. TCP offers the header field *window size* that determines the amount of data the receiver can collect without sending confirmation that the packet has been received. As a default, the window size field is 16 bit, which in turn results in a maximal window size of 65535 bytes. According to Mathis et al. [7], the maximum possible transfer rate is the TCP window size divided by the round trip time. Therefore, to reach 10 Gbps with a window size of 65535 bytes a maximum RTT of 0.066 ms would be needed which even for local networks is not a realistic number. RFC 7323 [1] offers a TCP Window Scaling Option to extend the window size. Both sender and receiver have to accept this option when establishing connections. An entry in the option field of the TCP header facilitates a multiplication of the window size field by 2^{14} . The window size can therefore reach up to 1 GiB.

TCP Timestamp, as described in RFC 7323 [1], extends the precision of the timestamp options already present in lower layers. The TCP header is extended by 8 bytes when this option is turned on. In most cases, this option is not needed and should therefore be turned off as more capacity can be allocated to payload.

In high-bandwidth networks, a high number of out-of-order segments can be observed. Due to the nature of the cumulative acknowledgments of TCP, this leads to a high number of retransmissions. *TCP selective acknowledgement (TCP SACK)* [6] provides the possibility to explicitly acknowledge out-of-order segments and prevents a large number of retransmissions and should therefore be turned on.

The *Nagle Algorithm* [9] facilitates groupings of small data amounts to bigger segments and therefore leads to fewer packets in total, which in turn allows for higher bandwidth as the overhead of the TCP headers declines. In some cases this can be obstructive when instant feedback from the receiver side is required, e.g. in case of SSH connections. Nevertheless, Nagle is especially useful when many small packets can be observed and should be turned on if the use case permits.

Jumbo frames that allow for Ethernet payloads larger than 1500 bytes are advisable to facilitate high bandwidths. How-

ever, hardware support has to be examined as not all networking devices support jumbo frames. For example, the reference NIC implementation of the NetFPGA card—which will be described later in this paper—does not. Therefore, if the test results should be generally applicable, a tester is advised not to activate jumbo frames.

Besides the kernel settings, there are also important configuration parameters to consider for the NIC. *Large Receive Offload (LRO)* merges packets upon entry into a stack to handle large amounts of data more efficiently. As this introduces some drawbacks, notably being limited to IPv4, TCP, and unvirtualized environments, *Generic Receive Offload (GRO)* was developed, which fixes some of the issues. LRO and GRO lead to less computation overhead on the receiver end. However, in some driver implementations, if one of the merged packets was not received correctly, all packets need to be resent, which can lead to inferior link utilization due to the significant number of retransmissions. Therefore, it is advisable to turn those features off.

B. Network Adjustments

There are—amongst others—two widely used adjustable network conditions to evaluate TCP performance: (1) packet loss and (2) packet delay. In a 10 Gbps testing environment, one can implement the aforementioned network conditions in different ways. Testers must consider from case to case which approach is best. There are several things to consider: Available hardware, evaluated use cases, related metrics, and the tester's experience with different implementations accompanying the chosen approach. In the following, various approaches are described.

1) *Packet Loss*: Packet loss is one of the most significant indicators for network congestion and is also used by many TCP congestion control algorithms for this purpose. There are various mechanisms to induce packet loss.

The simplest but least predictable way is to expand the test topology by adding additional senders. These senders initialize additional network flows to create actual congestion on a bottleneck switch or router. It is extremely challenging to create accurate and repeatable packet loss using this approach.

Another way is to use network environment simulator software. However, for a 10 Gbps test setup, the available software range is reduced to a few workable software tools. One of the best known network environment simulators is *NetEM*³. The user can adjust the drop chance and the burst length. The software needs to run on a computer with a 10G NIC with at least two interfaces, to be able to forward the incoming packets. Both aforementioned methods facilitate constant or bursty packet loss.

Another possibility to induce packet loss is to use a hardware device in the network that normally forwards packets but can also drop packets at a specific rate. For example, the NetFPGA-Sume or its predecessor the NetFPGA-10G⁴

³<http://www.linuxfoundation.org/collaborate/workgroups/networking/netem>

⁴netfpga.org

can be used as such a bump-in-the-wire forwarding device. The NetFPGAs are fully programmable network devices. A Virtex FPGA Processor on the PCIe NetFPGA card allows for traffic manipulation at line speed. With feasible effort, the official reference NIC implementation can be adjusted and it is possible to implement packet loss with different drop patterns. One downside is that familiarity with FPGA programming in Verilog is necessary.

2) *Packet Delay*: Packet delay has a strong impact on TCP sending performance as it significantly influences the growth of the TCP congestion window. In addition to the physical propagation delay, packet processing, buffer time, or suboptimal routing can slow down a packet on its route.

As with packet loss, there are software tools available to emulate packet delay as it would occur in real networks. For example, *NetEM* is able to delay incoming packets for multiple milliseconds. It is highly recommended to check the available hardware whether it is capable of buffering packets at a bandwidth of 10 Gbps and for how long this is possible. The available RAM—including data rate and read/write latencies—has a significant impact on the maximum possible buffer time.

Besides software solutions, there are some hardware approaches. A 10G NetFPGA card can be used to simulate long distance packet delay. The NetFPGA comes with RLDRAM, which provides low read and write latencies, and can buffer processed packets in a 10 Gbps environment. As part of the official NetFPGA repository on github⁵, there are some contributing projects, which can be used as a basis to implement a packet delaying device.

Hardware network simulators offer a more expensive but less implementationally complex approach. Such simulators are able to perform at line speeds of up to 40 Gbps. Additional features often include altering bandwidth, additional packet loss, or induced data corruption. For example, the EDS-10/40G Ethernet Delay Simulator⁶ can delay incoming packets for up to 8 seconds.

The most realistic solution—but also the most expensive one—is to use an actual long distance network. However, access to such a network is typically not easily available and long pipes often come along with quality impairments. Electrical noise on the link can cause bit corruptions. Furthermore, transceiver and muxponder in the network can also cause bit flipping. These occurrences lead to CRC errors. TCP reacts with retransmitting the corrupted packets. Besides the induced packet loss and packet delay, these additional retransmissions have to be taken into account when analyzing the test results. The longer the round trip time, the stronger the impact of the additional packet drops on the TCP performance by causing extended recovery phases.

IV. CONCLUSION

The performance evaluation of TCP variants in 10G high-speed networks provides new challenges, as testers are re-

quired to take the network hardware into account even more closely. The choice of relevant testing criteria and appropriate metrics to be measured still governs the overall testing procedure. Responsiveness, efficiency, fairness, downwards compatibility, and convergence time are the primary characteristics for comparing different TCP variants. In order to vary the network environment, the test setup should allow the modification of different drop rates, data corruption, and end-to-end round trip times. We believe that it is beneficial to induce such network characteristics using hardware-based solutions, because software-based approaches so far did not match our expectations regarding reliability and dependability in a 10G environment. In our setup, we relied on a NetFPGA for the induction of packet loss and corruption at line speed. Delays were induced using the propagation delay of a configurable network testbed with long distance links.

In general, prior optimizations of the network stack are urgently required to be able to use full bandwidths in the tests. This includes, but is not limited to, network settings of the OS kernel, NIC configurations, and settings of network equipment used in the test topology, e.g., advanced features of Ethernet switches.

If not considered carefully, one single subpar setting of the network setup can have huge effects on the actual performance of the TCP variants in a 10G network. With a change of bandwidth in an order of magnitude also comes a multiplying effect of any uncontrolled parameter confounding the results.

ACKNOWLEDGMENT

This work has been supported in the bwNET100G+ project by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

REFERENCES

- [1] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, “TCP Extensions for High Performance,” RFC 7323, 2014.
- [2] R. Jain, D. Chiu, and W. Haw, “A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System,” Digital Equipment Corporation, Hudson, USA, Tech. Rep., 1984.
- [3] B. H. Leita, “Tuning 10Gb network cards on Linux,” in *Proceedings of the Ottawa Linux Symposium (OLS)*, 2009.
- [4] Y. T. Li, D. Leith, and R. N. Shorten, “Experimental Evaluation of TCP Protocols for High-Speed Networks,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1109–1122, Oct 2007.
- [5] T. Lukaseder, L. Bradatsch, B. Erb, R. W. van der Heijden, and F. Kargl, “A Comparison of TCP Congestion Control Algorithms in 10G Networks,” *2016 IEEE 41st Conference on Local Computer Networks (LCN 2016)*, 2016.
- [6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgment Options,” RFC 2018, 1996.
- [7] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm,” *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997.
- [8] J. Mo, R. J. La, V. Anantharam, and J. Walrand, “Analysis and Comparison of TCP Reno and Vegas,” in *INFOCOM ’99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, Mar 1999, pp. 1556–1563 vol.3.
- [9] J. Nagle, “Congestion Control in IP/TCP Internetworks,” RFC 896, 1984.
- [10] D. Wei, P. Cao, and S. Low, “Time for a TCP Benchmark Suite?” Caltech, Tech. Rep., 2005.

⁵<https://github.com/NetFPGA/>

⁶<https://ecdata.com/eds-10g.html>