A Comparison of TCP Congestion Control Algorithms in 10G Networks

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

***Abstract*—The increasing availability of 10G Ethernet network capabilities challenges existing transport layer protocols. As 10G connections gain momentum outside of backbone networks, the choice of appropriate TCP congestion control algorithms becomes even more relevant for networked applications running in environments such as data centers. Therefore, we provide an extensive overview of relevant TCP congestion control algorithms for high-speed environments leveraging 10G. We analyzed and evaluated six TCP variants using a physical network testbed, with a focus on the effects of propagation delay and significant drop rates. The results indicate that of the algorithms compared, BIC is most suitable when no legacy variant is present; CUBIC is suggested otherwise.**

I. Introduction

The concept of layering separates concerns on different levels of network protocols. However, there are obvious and less obvious dependencies between different layers. While higher layers build on the services of lower layers, the actual characteristics of the lower layers also influence those higher layers. The transport layer is responsible only for providing end-to-end communication services between remote applications. Idealy, characteristics of lower layers should not influence its behavior. However, congestion avoidance, one of the common services of transport layer protocols, relies on a set of properties of the underlying network links, primarily latencies, drop rates, error rates, and bandwidth.

Due to this fact, advances in network technologies also have an effect on the behavior of higher protocols. For the predominantly TCP/IP-based stack of the Internet, continuous advances over the past decades have led to an increase of available bandwidth by several orders of magnitude, significantly affecting the performance of congestion control algorithms. The interplay of bandwidth, latency, and packet loss is key to all congestion control algorithms when balancing optimal link utilization while still preventing network congestion. When the increasing link bandwidth raises the bandwidth-delay product (BDP), the effects of occasional packet loss impacts the utilization disproportionally and requires counter-measures by extending TCP [6].

Furthermore, specific TCP congestion control algorithms have emerged to address the challenges introduced by high-speed networks. By adapting behavior and optimizing window parameters, these TCP variants aim for better and faster utilization while still remaining fair to unmodified TCP connections.

With the advent of widely available 10G Ethernet networks, the current state of the art of TCP comparisons primarily focusing on 1G becomes outdated. In this paper, we provide an in-depth comparison of various different TCP variants in a physical 10G network testing environment.

The remainder of this paper is organized as follows. Section II introduces the TCP variants to be evaluated and points to previous work that also compared TCP in high speed networks. In Section III, we present our methodology and the metrics used for our comparison. Section IV illustrates our test setup for the evaluation. In Section V, we discuss our results, and summarize take aways in Section VI, followed by a conclusion in Section VII.

II. Background

In this section we give background information about the evaluated TCP variants. Furthermore, the related work section summarizes other relevant papers comparing TCP variants and their results.

A. TCP Congestion Control Algorithms

The following provides a brief overview of these widely known TCP congestion control algorithms: TCP Reno, Scalable TCP, HS-TCP, H-TCP, BIC and CUBIC. Here, we only provide the additive increase and multiplicative decrease (AIMD) parameters, which are of particular interest for our analysis. The reader is referred to the original literature for more detailed information. The AIMD behavior can be described with an additive parameter (ACK received) and a multiplicative parameter (triple duplicate ACK, packet lost):

$$\textbf{ACK} : cwnd \leftarrow cwnd + \alpha \tag{1}$$

$$\textbf{LOSS} : cwnd \leftarrow cwnd \cdot \beta \tag{2}$$

Note that for small congestion windows (cwnd), all of these variants fall back to behave like Reno.

a) TCP Reno: TCP Reno is the most common TCP variant, and is also referred to as standard TCP. This variant initially uses slow start ($\alpha = 1$, $\beta = \frac{1}{2}$); after the first loss, α is set to $\frac{1}{cwnd}$.

b) Scalable TCP: Scalable TCP [8] is designed to achieve high throughput more quickly than Reno by making the recovery time independent of window size, which is beneficial for high bandwidth, high latency links. The AIMD parameters for Scalable TCP are $\alpha = 0.01$ and $\beta = 0.875$.

DOI 10.1109/LCN.2016.121

c) HighSpeed TCP: HighSpeed TCP (HS-TCP) [2] is designed to increase robustness of the transmission rate against packet loss, which is especially important for networks with long links. For this algorithm, the following AIMD parameters apply: $\alpha = f_\alpha(cwnd)/cwnd$ and $\beta = g_\beta(cwnd)$, where g_β (decreasing) and f_α (increasing) are logarithmic functions.

d) H-TCP: H-TCP [9] is designed to provide a better use of bandwidth for long, high-speed links with high BDP, while maintaining backwards compatibility with regular TCP flows. Unlike previous approaches, the authors use the time (Δ) since the last congestion event to set the AIMD parameters, which can be summarized as follows: $\alpha = \frac{2(1-\beta)f_\alpha(\Delta)}{cwnd}$ and $\beta = \frac{RTT_{min}}{RTT_{max}}$, unless the measured throughput changes significantly (controlled with a parameter $\Delta_B = 0.2$). $f_\alpha(\Delta)$ is 1 for backwards compatibility (below a threshold Δ_L); $f_\alpha(\Delta) = 1 + 10(\Delta - \Delta_L) + 0.25(\Delta - \Delta_L)^2$ is suggested to achieve high utilization quickly.

e) BIC TCP: BIC TCP [13] was developed to address the observed suboptimal RTT fairness of earlier congestion control algorithms. RTT fairness refers to fairness between flows with different RTTs. The authors point out that the problem is inherent to the increased utilization, due to the way earlier algorithms are designed, and develop BIC to solve this challenge. Their algorithm uses two phases to update the bandwidth; linear increase to approach a fair window size, and binary search to improve RTT fairness. Linear increase is similar to additive increase, while binary search essentially uses two window sizes (W_{max} and W_{min}) that updates these windows and the actual window size to approximate the optimal window size. Once W_{max} and W_{min} are converging, BIC falls back to linear increase.

f) TCP Cubic: CUBIC TCP [3] is an improvement of BIC, which aims to compensate the aggressive behavior of BIC to more reasonable levels, and simplifies the algorithm. The impact of this aggressive behavior was especially notable in networks with low RTT. Like BIC, CUBIC uses the W_{max} window; however, it sets the window size using a cubic function that plateaus at W_{max}:

$$W(t) = C \cdot (t - K)^3 + W_{max} \tag{3}$$

where C is a scaling factor, t the time since the last window reduction and $K = \sqrt[3]{W_{max} \cdot \beta / C}$. This results in a window increase that is similar to BIC's binary search.

B. Related Work

There are many TCP performance evaluation papers that cover environments with link speeds up to 1 Gbps [4], [12], but there are very few that look at 10 Gbps transmission rates. In the following, we briefly review two relevant studies from this body of work.

Li et al. [10] measured the performance of Scalable TCP, HS-TCP, H-TCP, BIC and FAST-TCP on the basis of fairness, backward compatibility, efficiency, and responsiveness including convergence time. All tests were performed in a test setup based on the dumbbell topology with two competing flows starting at different points. The authors varied the parameters of propagation delay (up to 320 ms), the bottleneck bandwidth (up to 250 Mbps) and different numbers of parallel web traffic flows. The tested TCP variants provide poor fairness but better link utilization than standard TCP. Beyond that, the algorithms Scalable TCP, HS-TCP and BIC suffer from high convergence times. While this paper provides a detailed overview and a comprehensive analysis of the different TCP congestion control algorithms and their performance, the results are not transferable to high-speed networks because congestion control algorithms behave differently depending on bandwidth. For example, the additive increase function changes linearly with the bandwidth in Scalable TCP and Reno, but logarithmically in HS-TCP as we describe in more detail in the following section. Similarly, the behavior of BIC and CUBIC depends on time and bandwidth differently than the other variants. Therefore, results of lower bandwidth tests do not apply to networks with higher bandwidth.

Arokkiam et al. [14] evaluated the performance of the TCP variants Reno, BIC and H-TCP over XG-PON. The authors assess the results on the basis of efficiency, fairness, responsiveness and convergence. One single or two competing highspeed flows were induced, alternately with or with no competing UDP background traffic, into a 10 Gbps XG-PON network. All algorithms show good link utilization in a single flow environment with very small RTTs, but the link utilizations decreases with increasing round-trip times. With two competing flows with different starting times the authors were not able to achieve a proper convergence between the flows. With existing background traffic, two new TCP flows show varying convergence behavior. This paper provides an extensive analysis of the mentioned TCP variants. However, key variants such as CUBIC and HS-TCP are missing.

III. Methodology

We chose an experimental approach for our comparison. Utilizing a dedicated testbed, we conducted a large number of measurements of the different TCP variants recording various metrics.

A. Use Cases

Because we focus on high-speed networking, we motivate our tests with a traffic pattern and workload typical for data centers. For economic reasons, Ethernet has been replacing more specialized link layer technologies within data centers for TCP/IP-based networks. Alizadeh et al. [1] analyzed large amounts of data center traffic and identified two major traffic types: (i) relatively short flows with low latency requirements and (ii) large flows requiring high throughput. The former type is primarily a result of web application requests, database queries, and similar interactions within distributed application architectures. The latter type is based on long-running interactions, such as software updates, continuous database replications, data-intensive application workloads, or backup processes.

For our own experimental setup, we concentrate on this second traffic type, as it is more interesting to consider for

high-speed networks. When long-running, latency-insensitive flows concurrently compete for utilization on a shared high-speed network, characteristics of different congestion control algorithms become apparent.

Furthermore, we include network traffic both inside and between data centers. The *intra-data center* traffic is the more obvious use case and is characterized by shorter physical links and corresponding lower end-to-end latencies between nodes. The *inter-data center* traffic represents communication between geographically separated data centers, yielding much higher latencies. This use case includes the usage of multiple data centers for higher availability, increased locality, and improved resilience. Still, the traffic patterns between sites yielded by continuous data synchronization, database replication, and periodically disseminated backup remains very similar to traffic within a single data center site. Note that we expect data centers to have dedicated remote connections, as we do not take into account background Internet traffic for our tests.

Although we motivate our experiments with large flows between and within data centers, we believe that results can be generalized to many other use cases with similar traffic and network infrastructure properties.

B. Criteria

We used the five criteria used by Li et al. [10] and adapted them to the aforementioned use cases where necessary.

Responsiveness describes the ability of the algorithms to recover quickly from random packet loss. We measure the average throughput at different drop probabilities for a packet in the network. We also measured this with different propagation delays as this has a major impact on the recovery speed.

Efficiency is the utilization of the network resources. A protocol is efficient if the available bandwidth at the bottleneck is utilized as fully as possible. In some applications — in addition to this metric — not only the average utilization is important but also the maximum and minimum utilization. For instance, when incoming data is processed immediately, applications will be slowed down if the bandwidth fluctuates heavily. Therefore, we chose to also assess the quantiles to measure if high throughput at its peaks can be accomplished (Q 0.75), if the average throughput is acceptable (Q 0.5) and if the throughput has an acceptable minimum (Q 0.25). In addition to the responsiveness, which shows the average throughput, this metric shows how stable the algorithms are.

We measure *Fairness* with Jain's fairness index [7]:

$$J(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

with x_i being the mean of flow i with n flows overall. We analyze two flows running in parallel, each with the same configuration, e.g. TCP variant and parameters. A perfect algorithm would result in $J = 1$, worst case would be $J = \frac{1}{n}$.

Backwards compatibility is a measurement of fairness within networks where older systems are still in use. We adapted our fairness test by using Reno for congestion control

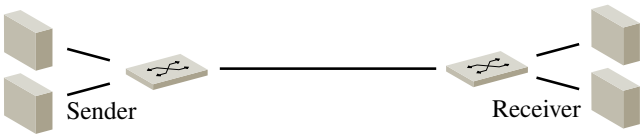


Fig. 1: Standard Dumbbell Topology.

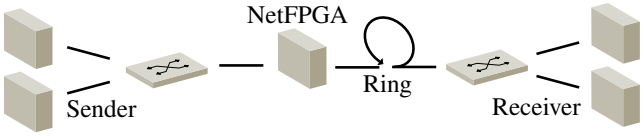


Fig. 2: Extended Dumbbell Topology.

in one of the two flows to evaluate how the congestion control algorithms behave in networks with legacy systems. Reno was chosen as it is the most common legacy variant still in use.

ε-Convergence time t_c was defined by Li et al. [10] as the time required for the short-term average throughput to achieve $\varepsilon \cdot \bar{u}_i$, where $\bar{u}_i$ is the long-term average throughput of stream i. Here, the short-term average throughput is defined as:

$$u_i(t+\delta) = (1-\lambda) \cdot u_i(t) + \lambda \frac{\Delta u}{\delta}$$

Where Δu is the number of bytes transferred, and λ is a parameter that specifies how quickly the short-term average throughput changes. In practice, we observe that the new stream always takes a longer period of time to converge to its long-term average throughput; results from [10] suggest that convergence time should be measured by analyzing this new stream. Extending the metric provided in prior work, we compute the average distance from the long-term average throughput after the convergence time is reached, in order to quantify the stability of this convergence, which we refer to as $spread$ (where T is the number of measurements):

$$s = \frac{\sum_{t=t_c}^{T} |\bar{u}_i - u_i(t)|}{T}$$

For every TCP variant, we conducted tests for each aforementioned metric using the test setup described in the following section. Every test was conducted five times over a period of 15 min to ensure that measurement errors can be ruled out and random deviations do not distort the results.

IV. Test Setup

Our test setup is based on the standard dumbbell topology with two senders and two receivers (Figure 1). Each of them is equipped with HP NC523SFP Dual 10 Gbps NICs and Ubuntu 14.04.1 (Linux 3.16). We used 10G Ethernet with IPv4 on the lower layers as these are the most common protocols in our use cases. For the efficiency and responsiveness measurements, only one sender and one receiver were active. Two HP 5920 JG296A switches (Firmware HPE Comware Software, Version 7.1.045, Release 2422P01) were used to combine and separate the flows. Flow control in the switches was turned off. The program *iperf3* was used to produce TCP traffic with up to

TABLE II: Test overview; every listed parameter permutation was tested.

Criterium	Metrics	Parameters			# of Tests
Responsiveness	Average throughput	Variant: *Reno, Scalable, HS-TCP, BIC, CUBIC, H-TCP*	RTT: *0.2 ms, 6.2 ms, 12.2 ms, 24.2 ms*	Drop rate: *0 or* $10^{-i}, i \in \{7,6,5,4,3,2\}$	840
Efficiency	Throughput distribution				
Fairness	Jain's Fairness Index, Link utilization				120
Downwards compatibility					110
Convergence time	Convergence time, Spread			Variant for 2^{nd} flow: *same as first variant or Reno*	220
					Σ 1,280

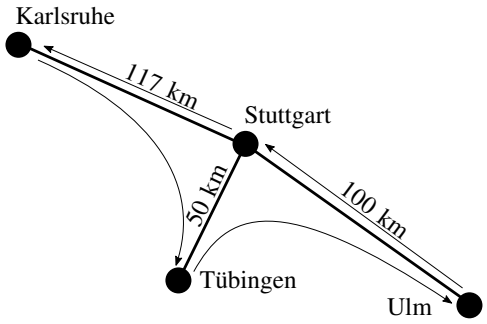


Fig. 3: Ring configuration in the state-owned research network.

TABLE I: Used TCP parameters.

TCP variant	Parameters
Scalable	$\alpha = 0.02, \beta = 0.875$, Low_Window = 50
HS-TCP	Low_Window = 38, High_Window = 83000, High_P = 10^{-7}, High_Decrease = 0.1
BIC	$S_{max} = 16$, B = $4, \beta = \frac{819}{2014}$, Low_Window = 14
CUBIC	$\beta = 717/1024$, legacy if *cwnd* $< W_{tcp}(t)$
H-TCP	$\Delta^L = 1s, \Delta_B = 0.2$

10 Gbps for each sender. Several adjustments in the software settings were necessary to enable the senders to satisfy the bandwidth requirements. For example, *Large Receive Offload (LRO)* and *Generic Receive Offload (GRO)* had to be turned off. LRO and GRO merge packets in the NIC upon receiving them. As the loss of one packet means that the whole group has to be resent, this can lead to very low link utilization. The *TCP Window Scaling Option* had to be turned on to allow bigger window sizes exceeding the default maximum of 65535 bytes to a maximum of 1 GiB. The *TCP Timestamp* which extends the TCP header by 8 bytes was deactivated as it is not used in our scenarios and the bandwidth can therefore be used for payload instead. *TCP selective acknowledgement (TCP SACK)* was activated to accommodate the high number of packets in 10 Gbps networks and to avoid retransmissions. The *Nagle Algorithm* to facilitate groupings of small data amounts to bigger segments was also turned on to allow for higher throughput. Table I shows our settings for the TCP variants and when they fall back to legacy mode. These values are the default values of Linux 3.16 and are predominantly based on the parameters set by the original developers of the congestion control algorithms. For further details on how to facilitate such tests and which points to consider when setting up a TCP benchmarking environment, we refer to our extensive discussion in [11].

To test our criteria, the test network had to fulfill two essential requirements: Variable propagation delay and an adjustable drop rate. To achieve this, the dumbbell topology was extended as described in the following.

We used a dedicated, state-owned research network which is equipped with configurable 10x10 Gbps connections between research institutions in Ulm, Tübingen, and Karlsruhe to achieve variable propagation delay, as shown in Figure 3. The connections between the universities were used to form 4 ring structures between the locations. The rings begin and end in Ulm at the patch panel directly connected to the switches used for the tests. There are 38 transceivers but no switches within one 534 km long ring yielding a delay of 6 ms. Connecting all four rings therefore enables us to induce up to 24 ms of real physical delay in our tests. This network setup experiences rare bit flips, which in turn cause retransmissions of TCP packets. Those retransmissions can be observed 15 times on average with a standard deviation of 2.8 within a 15 min test at 10 Gbps. Therefore, the likelihood of a loss to occur is 2.2×10^{-8} for every packet. For comparison: to comply with 802.3ae [5], the error rate of optical fiber connections cannot exceed 1×10^{-12}. At the time of writing, the cause of these errors could not be found. However, in the following sections, we clearly document when this error has an impact on the quality of the results.

A NetFPGA 10G Card as a standalone bump-in-the-wire device was used to realize the adjustable drop rate. The reference NIC implementation of the NetFPGA project was used and adjusted to our needs[1]. In its standard configuration the card is a simple forwarding device at linespeed. Our extension makes it possible to set an evenly distributed likelihood for a packet to be dropped.

The NICs are able to send 9.5 Gbps without using jumbo frames and are only able to send full 10 Gbps with jumbo

[1]https://github.com/NetFPGA/

frames activated. Unfortunately, jumbo frames are not supported by the NetFPGA. Therefore, all tests were conducted with a maximum throughput of 9.5 Gbps per NIC.

Table II shows an overview of all undertaken tests and the achieved results. As every permutation of variant, RTT, and drop rate or second variant was tested and every test was conducted five times, a total of 1,280 tests were undertaken.

V. Results

Testing the *responsiveness* and *efficiency* at an RTT of 0.2 ms and without packet drops, every TCP congestion control algorithm fully utilizes the link, as can be seen in Figure 4a. Even with drop rates of up to 1×10^{-6}, the average path utilization does not fall under 9.48 Gbps. At a drop rate of 1×10^{-5}, however, Reno and CUBIC show significant performance losses. CUBIC shows the same behavior as Reno because it operates in legacy mode and thus behaves just like Reno in this scenario. The other variants perform significantly better with a $cwnd$ that is at least 10 times bigger. At a drop rate of 1×10^{-4} packets, it can clearly be seen that BIC works best in lossy networks with low latency and reaches 8.3 Gbps.

Without additional delay in the network, all variants are equally efficient and show low scattering without packet drops. Differences occur only at 1×10^{-5} and above when CUBIC and Reno start to perform worse than the other variants. Q(0.5) is 8.6 Gbps, Q(0.25) 8.4 Gbps and Q(0.75) is 8.8 and 8.9 Gbps respectively. They both show the same behavior as CUBIC is in legacy mode. At a drop rate of 1×10^{-4}, CUBIC and Reno show significant scattering of 1.5 Gbps between 4 and 5.5 Gbps; 25% of all values lie above 4.8 Gbps.

As Figure 4b shows, the differences between congestion control algorithms become apparent at a drop rate of 1×10^{-3}. Reno, CUBIC and HS-TCP show the same behavior with bandwidth scattering between 1.9 and 2.7 Gbps. BIC performs best and reaches up to 8.5 Gbps with 50% of all measurements between 8.2 and 8.27 Gbps.

With an RTT of 6.2 ms, different behavior than with 0.2 ms can be observed with drop rates above 1×10^{-7} (Figure 5a). Reno reaches only 3.6 Gbps while the high-speed variants reach the uppermost limit, with the exception of H-TCP with 9.35 Gbps. At a drop rate of 1×10^{-6}, Reno reaches 1.4 Gbps. CUBIC (6.26 Gbps) and HS-TCP (6.52 Gbps) also show significant performance drops, despite not performing in legacy mode. H-TCP with a similar $cwnd$ still shows better performance than the aforementioned variants as the algorithm raises the bandwidth faster after a reduction. Note that BIC outperforms all other algorithms. Even at a drop rate of 1×10^{-2} BIC still reaches an average throughput of 2.3 Gbps while every other variant lies below 1 Gbps. BIC's very good performance can also be observed in Figure 5b showing the efficiency at a 1×10^{-5} drop rate.

At 12.2 ms RTT — as can be seen in Figure 6a — Reno loses bandwidth even with low drop rates quite drastically. The other variants show very diverse reactions. At 1×10^{-5}, only BIC can keep the bandwidth high with 4.9 Gbps on average. The runner-up, Scalable, only reaches 1.6 Gbps.

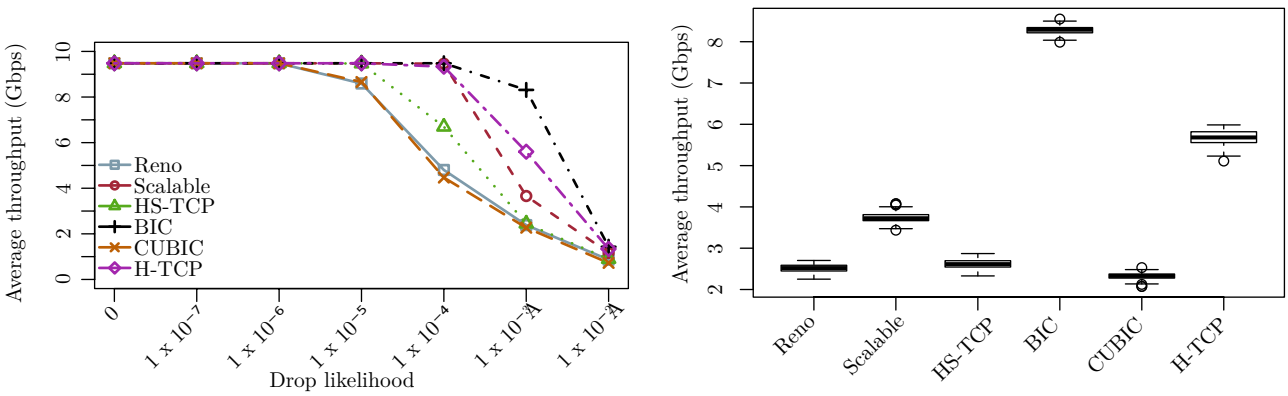
Reno only reaches the bandwidth it does because of a high throughput in the beginning of the test, because with very low drop probability, it takes several seconds for the first drop to occur. After the first drop, Reno only reaches between 949 Mbps and 1.8 Gbps at 1×10^{-7} drop rate. At a drop rate of 1×10^{-6} (Figure 6b), some variants but especially H-TCP and HS-TCP show high scatter. H-TCP reaches a an average of at least 6.2 Gbps in 75% of all measurements. HS-TCP reaches an average of 3 Gbps. However, 50% of all measurements lie between 1.7 and 2.9 Gbps. The highest bandwidth is 9.42 Gbps, the lowest 611 Mbps.

At an RTT of 24.2 ms, the aforementioned CRC errors in the networks warp the results (Figure 7a). Even with no induced drop rate, the errors from the network infrastructure lead to low throughput for some of the variants. Relative to the other variants, CUBIC's performance improves significantly with higher RTT, which shows that CUBIC is designed with high RTTs in mind. At 12.2 ms, CUBIC reached 71% of the throughput of H-TCP at a drop rate of 1×10^{-6}; at 24.2 ms it already reaches 78%.

The effects that can be observed at 12.2 ms are even more significant at 24.2 ms. The network CRC errors also take their toll, as can be seen in Figure 7b. Reno loses bandwidth solely because of the scarce CRC errors and the following retransmissions and fluctuates heavily between 742 Mbps and 9.48 Gbps. HS-TCP is also highly influenced by the errors. 50% of the measurements show an average of 4.5 to 7.3 Gbps. Outliers can be observed between 3.3 Gbps and 9.48 Gbps.

It comes with no surprise that in a local network with no additional latency (0.2 ms) — shown in Figure 10 — all TCP congestion control algorithms show complete *fairness*. The biggest difference between the two flows is 150 Mbps and the two flows utilize the ring fully (Figure 8a). Even legacy Reno performs as well as the other variants. The bandwidth utilization is lowest for CUBIC with 9.29 Gbps and best for BIC with 9.42 Gbps for both flows combined. At 12.2 ms, fairness lies between 0.96 (CUBIC) and 0.989 (BIC). The very good fairness values come from the flows increasing their bandwidth in parallel as their algorithms behave identical under those circumstances and as there is no overload for most of the test duration. A short overload period in the beginning when both flows try to send at 10 Gbps and long recovery times afterwards lead to very low link utilizations between 3.6 Gbps and 5.6 Gbps with all variants. Reno performs worst in this regard but the other variants HS-TCP, H-TCP and CUBIC show little improvement. BIC and Scalable perform best. With even longer recovery times, the link utilization becomes worse with an RTT of 24.2 ms. Here, H-TCP performs a lot better than before compared to the other variants. Except for HS-TCP, all variants show more than two times the link utilization of Reno. The logarithmic functions of HS-TCP seem to work badly with high latency.

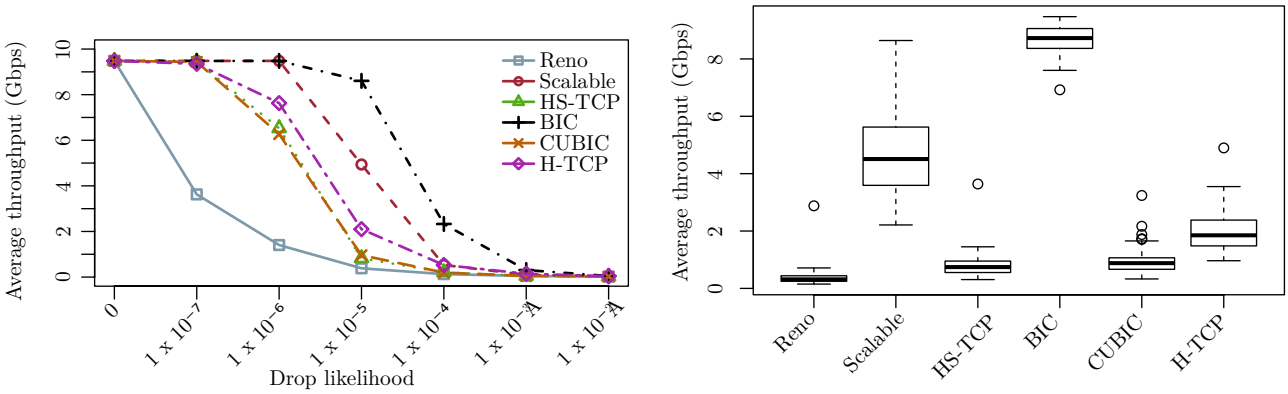
The resulting fairness is worse for every permutation when the different variants are not competing with themselves but with Reno to evaluate *downwards compatibility*, which can be seen in Figure 8b. With higher RTT, the flow with the high-



(a) Responsiveness.

(b) Efficiency; 1×10^{-3} drop rate.

Fig. 4: Responsiveness & efficiency at 0.2 ms RTT.



(a) Responsiveness.

(b) Efficiency; 1×10^{-5} drop rate.

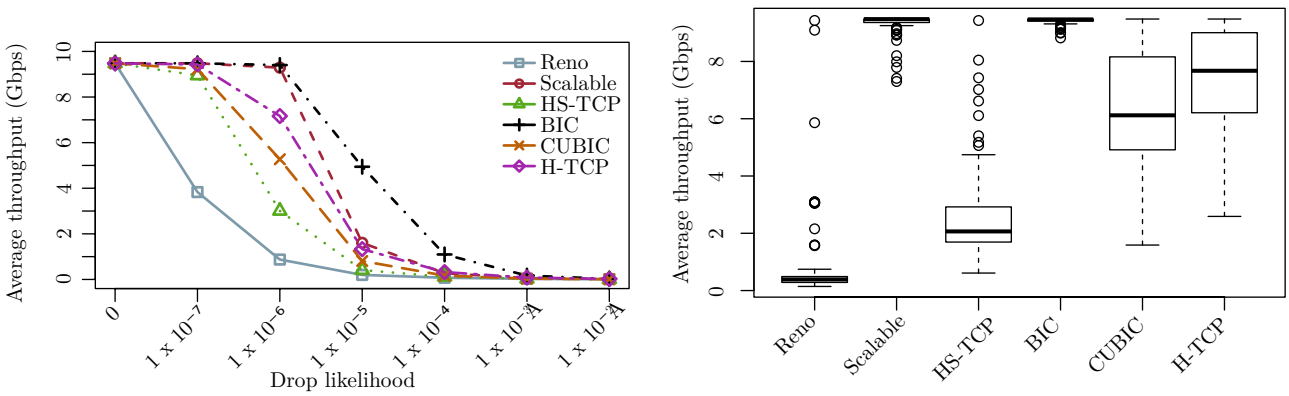
Fig. 5: Responsiveness & efficiency at 6.2 ms RTT.

speed variant always takes more bandwidth than the Reno flow as the algorithms have a shorter recovery time after packet drops. H-TCP, CUBIC and HS-TCP are the fairest which negatively reflects in their results concerning link utilization as a fair bandwidth propagation means more bandwidth for Reno and less bandwidth for the high-speed flows with a therefore smaller *cwnd*. Hence, in case of overload, the high-speed flow needs more time to recover and the link utilization is lower. As BIC and Scalable are less fair, the overall link utilization is higher, which becomes especially apparent with 24.2 ms RTT.

In previous work [10], evaluating the *convergence time*, λ was not clearly specified: we set $\lambda = 0.1$, which appeared to be a good trade-off; a smaller λ leads to a smoother convergence, but increases the overall convergence time. It is out of scope for this work to provide an extensive analysis of the effect of this metric parameter (just as we do not change the $\varepsilon = 0.8$ given by Li et al. [10]). Instead we measured the effect of round trip time and congestion control algorithm on ε-convergence time. In addition, we computed a measure for the stability of this convergence, as discussed in Section III-B. The results are shown in Figure 9, where the bar chart represents convergence time and the scatter plot reflects the spread. For clarity, we left out the measurements with an RTT of 0.2ms: these measurements converged virtually instantly with very little spread. A significant result is that any high speed TCP variant leads to a very unstable convergence when streams with Reno are involved. This is due to the fact that the distribution of bandwidth is very uneven (see Figure 10), which amplifies the effects of retransmissions and leads to higher spread. As expected, Reno itself shows unstable behavior as the RTT increases. This is related to the fact that Reno's responsiveness is dependent on RTT, and it was one of the reasons the new congestion control algorithms were developed in the first place. Finally, we point out that convergence time goes down as the RTT increases, which is in part due to reduced utilization. This reduced utilization is caused by the CRC failures of the link, as discussed in the setup.

VI. Take Aways

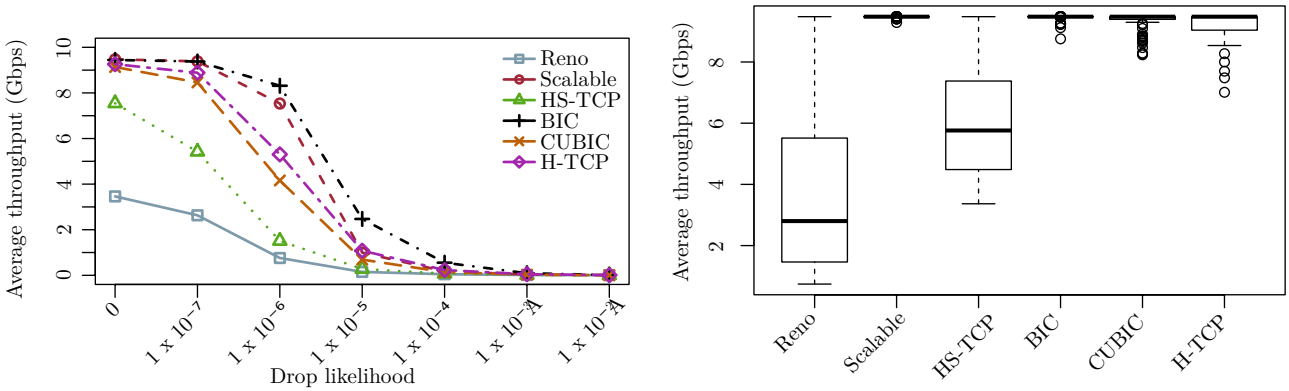
It becomes evident that, when there is no packet loss in the network, the actual TCP variant has no strong influence on the performance. However, as soon as there is packet loss — especially with noticeable RTT — the congestion control algo-



(a) Responsiveness.

(b) Efficiency; 1×10^{-6} drop rate.

Fig. 6: Responsiveness & efficiency at 12.2 ms RTT.



(a) Responsiveness.

(b) Efficiency; no induced packet drops.

Fig. 7: Responsiveness & efficiency at 24.2 ms RTT.

rithm affects the quality of service — e.g., link utilization and responsiveness — immensely.

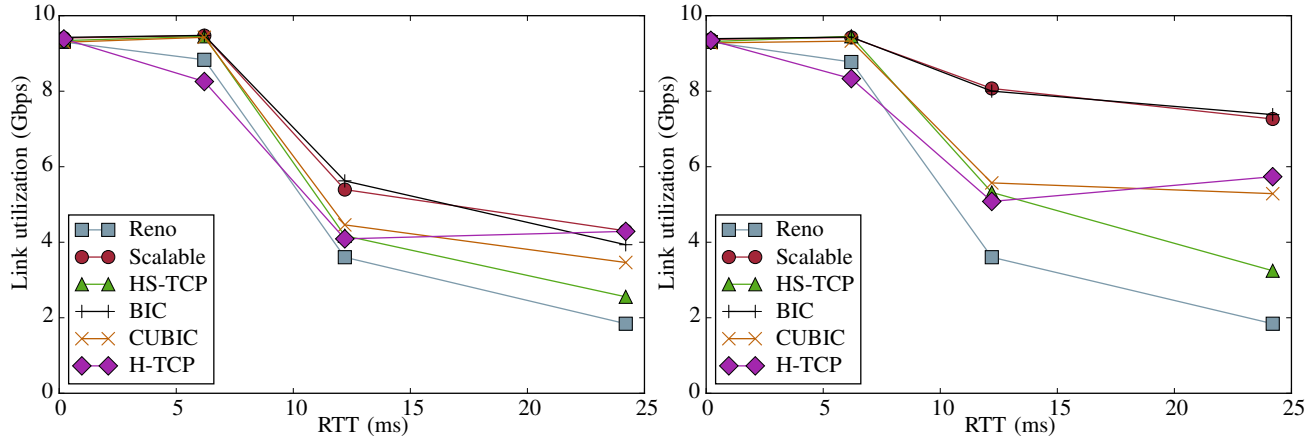
Taking into account our use cases mentioned before, BIC shows the best properties for intra-data center and inter-data center communication alike. Especially for higher drop rates, BIC outperforms every other variant. In every test, BIC shows the best behavior or — in case of link utilization and fairness against itself — is one of the best variants.

However, when looking into backwards compatibility, BIC shows its weakness. Like HS-TCP and Scalable, at an RTT of 6.2 ms, BIC dominates the network link. Here, CUBIC is by far the best alternative, which also performs well in the other tests. At an RTT of 12.2 ms and above, fairness becomes less important as low link utilization leads to a network without overload and therefore enough bandwidth for every user.

The decision as to which variant is best for a network cannot be based on a general answer. It depends on the amount of influence one has over the network. For example, if the network is within a data center where every system is under control of the local administrators, our analysis shows that BIC should be used on every system. However, the variants we investigated are all developed with downwards compatibility to Reno in mind and other variants that were not part of our analysis might perform better in this scenario. On the other hand, CUBIC is the best choice if the composition of the network components is not known and older systems might still be present, as it offers a reasonable trade-off between link utilization and efficiency on the one hand and also fairness towards other systems on the other hand.

VII. Conclusion

In order to assess the performance and behavior of TCP congestion control algorithms in 10G Ethernet networks, we evaluated TCP Reno and five modern variants (Scalable TCP, HS-TCP, BIC, CUBIC, and H-TCP) in a separate test environment. While we induced drop rates with a custom NetFPGA device, we caused latencies by using a dedicated research network with wide-area physical links yielding actual propagation delay. For each variant, we assessed responsiveness, efficiency, fairness, downwards compatibility, and convergence time with varying RTT and drop-rate parameters. In total, we executed more than 1,200 isolated test runs to collect data for all metrics.



(a) Variant against itself.

(b) Variant against Reno.

Fig. 8: Link utilization of different TCP congestion control algorithms.

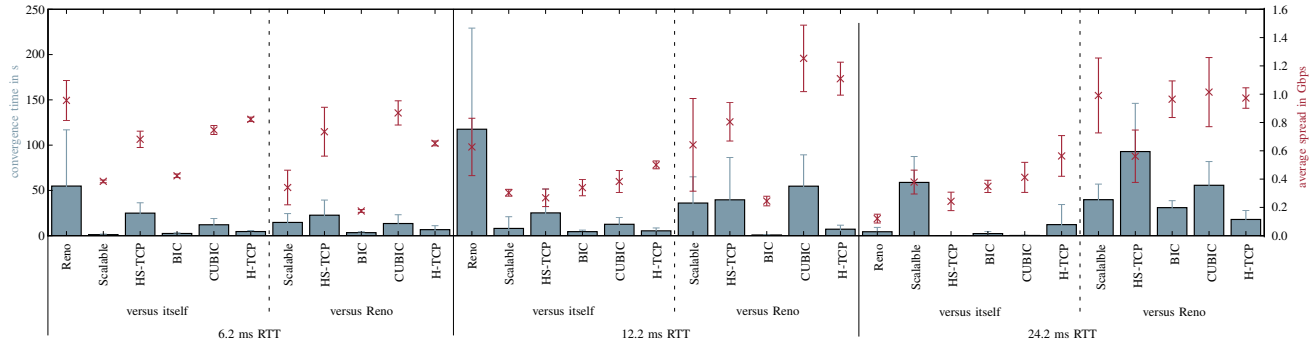


Fig. 9: Convergence time and spread of the converged flows.

While all modern variants generally outperformed TCP Reno in a 10G setting, the comparison among the other variants yielded more varying results. BIC leads most of our results in the test setup due to its aggressive behavior. At the same time, BIC performed poorly at backwards compatibility and its applicability in higher-latency and heterogeneous networks should be considered carefully. For such networks, CUBIC may represent a more appropriate alternative. In summary, we recommend the switch to a modern TCP variant for 10G networks and the selection of a variant based on the predominant latency and drop rate characteristics that the networked applications will experience in that network.

A. Future Work

While we compared several loss-based congestion control algorithms that promise downwards compatibility to legacy TCP variants such as Reno, algorithms that were not designed with this in mind, promise improved bandwidth utilization. One example for such a TCP variant would be the delay-based TCP Vegas algorithm. Application of delay-based algorithms in a heterogeneous network with loss-based variants present is still an open challenge. In more homogeneous networks, however, these variants could significantly improve performance.

Furthermore, we used the dumbbell topology for our test setup in line with prior research. However, more complex topologies such as the parking lot topology could yield more insights whether certain TCP congestion control algorithms are suitable for corresponding scenarios.

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] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[2] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, 2003.

[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel*, vol. Volume 42 Issue 5, pp. Seite 64 – 74, 2008.

[4] ——, "Comparison of High Speed Congestion Control Protocols," *International Journal of Network Security & Its Applications (IJNSA)*, vol. Volume 4 Issue 5, 2012.

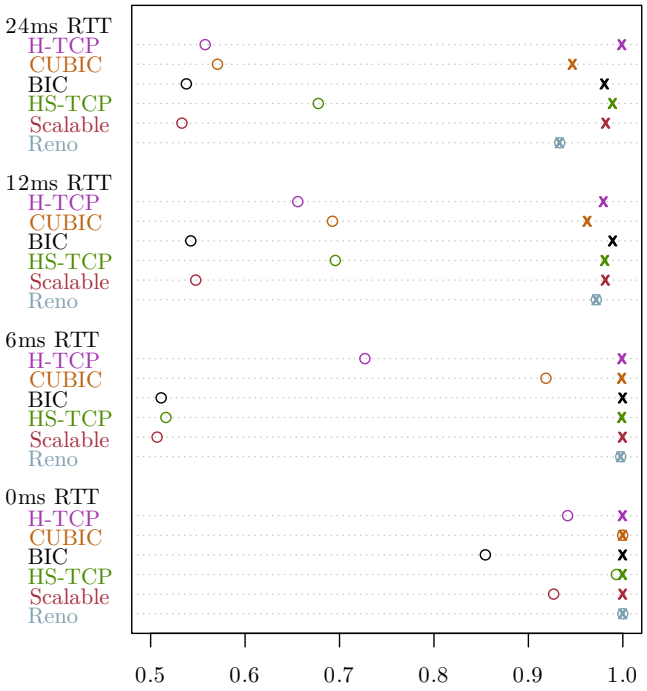


Fig. 10: Fairness of the variants against themselves (x) and downwards compatibility against Reno (•).

[5] IEEE, "Standard for Information technology - Local and metropolitan area networks - Part 3: CSMA/CD Access Method and Physical Layer Specifications - Media Access Control (MAC) Parameters, Physical Layer, and Management Parameters for 10 Gb/s Operation," *IEEE Std. 802.3ae-2002*, 2002.

[6] V. Jacobson and R. Braden, "TCP Extensions for Long-Delay Paths," RFC 1072, Mar. 2013.

[7] 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.

[8] T. Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," *ACM SIGOPS Operating Systems Review*, vol. Volume 33 Issue 2, pp. Seite 83 – 91, 2003.

[9] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proceedings of the PFLDnet Argonne*, 2004.

[10] 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, Oc5 2007.

[11] T. Lukaseder, L. Bradatsch, B. Erb, and F. Kargl, "Setting Up a TCP Benchmarking Environment—Lessons Learned," *2016 IEEE 41st Conference on Local Computer Networks (LCN 2016)*, 2016.

[12] M. Weigle, P. Sharma, and J. Freemen, "Performance of Competing High-Speed TCP Flows," in *NETWORKING 2006*, F. Boavida, T. Plagemann, B. Stiller, C. Westphal, and E. Monteiro, Eds. Berlin, Germany: Springer, 2006, pp. 476 – 487.

[13] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *Proceedings of the INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies* , 2004.

[14] ——, "Experimental Evaluation of TCP Performance over 10Gb/s Passive Optical Networks (XG-PON)," in *Proceedings of the Globecom 2014 - Symposium on Selected Areas in Communications: GC14 SAC Access Networks and Systems*, 2014.