# TCP at 100 Gbit/s – Tuning, Limitations, Congestion Control

Mario Hock, Maxime Veit, Felix Neumeister, Roland Bless, Martina Zitterbart
Karlsruhe Institute of Technology
Karlsruhe, Germany
E-Mail: mario.hock@kit.edu, maxime.veit | felix.neumeister@student.kit.edu, bless@kit.edu, zitterbart@kit.edu

***Abstract*—Link capacities increase at an enormous pace, with 100 Gbit/s becoming standard in data centers, campus networks, and the Internet. These ever increasing data rates are challenging since end-system performance (esp. CPU performance) cannot keep up with the growth rates.**

**Still, the TCP protocol and today's hardware are capable of transferring 100 Gbit/s with a single sender/receiver pair. However, extensive tuning is necessary down to manual interrupt configuration and corresponding CPU core pinning for the applications. A major issue is packet loss within the receiving end-system that cannot be prevented by TCP's flow control. This, in turn, affects TCP's default congestion control that interprets the losses as congestion signal. In this paper we show how to tune end-systems that are driven at their performance limits, what data rates are feasible, where the limitations are, and discuss the impact on and by TCP's congestion control.**

## I. Introduction

Data transmissions at 100 Gbit/s are a challenge, but yet achievable even with commodity server hardware. 100-gigabit Ethernet devices are becoming available at reasonable costs and can be installed in regular end-systems. In this paper we address the questions where the performance limits of off-the-shelf servers are, which tuning is necessary to achieve 100 Gbit/s and we discuss the impact *on* and *by* congestion control. Supporting such speeds up to the end-systems is relevant, e.g., for the transfer of scientific data between locations or other large data transfers. The knowledge where the limitations of commodity end-systems are is also relevant for network expansions and for the planning which connection options should be provided in university or commercial data centers.

Congestion control has to dynamically detect the network capacity and to limit senders in order to keep the network from severe overload. Ever increasing data rates pose a particular challenge: The range of possible network capacities has increased tremendously over the years, which requires an enormous *scalability* of congestion control algorithms. At the start-up of a new connection it is completely open what a suitable sending rate in the current context would be, e.g., 1 kbit/s or 100 Gbit/s. For very high speeds, a capacity seeking sender must quickly increase its sending rate to make efficient use of the available bandwidth. Similarly, the reaction to (non-congestion related) packet loss can be quite strong, so that capacity is also wasted by the back-off and subsequent ramp-up phase, leading to inefficient utilization of the capacity [2]. In addition to that, new challenges can be observed if the end-systems are driven at their performance limit. This includes non-congestion related packet losses in the end-systems, as we will detail within this paper.

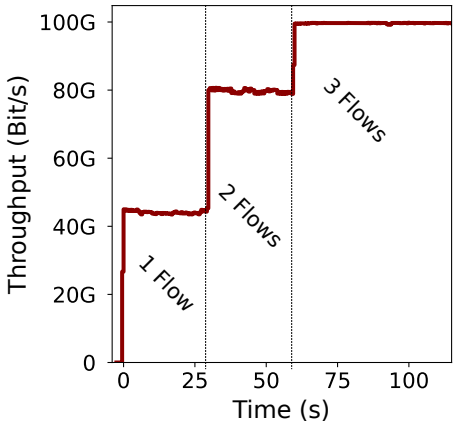



Fig. 1: 100 Gbit/s with single sender/receiver pair; 3 flows

Commodity servers are often powered by multi-socket, multi-core CPUs at comparatively low clock speeds. A common setup would be for example: Two Intel Xeon CPUs, each with 10 physical cores at 2.20 GHz interconnected over Intel's *Quick-Path Interconnect (QPI)*. In conjunction with *hyperthreading* this makes a total of 40 logical low-speed cores. Such a setup is common today but has distinctive limitations with respect to high-speed data transmissions and requires special tuning that considers specifics of the hardware and the setup.

This paper makes the following contributions:

- We show that 100 Gbit/s can actually be achieved between a single sender and a single receiver with existing hardware. For this we equipped non-cutting-edge mid-range server systems from the year 2014 with 100 Gbit/s *network interface cards (NICs)* and tuned them extensively.
- We give detailed tuning instructions to drive these systems to their performance limits and show that multi-socket systems require particular attention.
- We discuss demands for future high-speed congestion controls.

Hereby, we focus on commodity servers with 100-gigabit Ethernet devices, high performance computing (HPC) clusters with Infiniband networking are out of scope of our research.

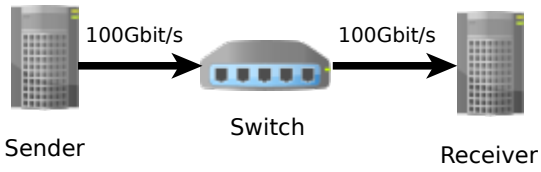



Fig. 2: Minimalistic testbed setup; no congestion

The remainder of the paper is structured as follows. Section II shows the capabilities and limitations of carefully tuned end-systems. The tuning is detailed in Section III. For end-systems driven at their performance limits, packet loss within the end-systems becomes relevant. Section IV gives background how this happens and why this cannot be prevented by TCP flow control. Section V discusses how these packet losses affect the congestion control performance and shows promising mitigation approaches. Related work is discussed in Section VI.

## II. Capabilities and Limitations

Parallelism does not come naturally to network transmissions. Consider a simple *network interface card (NIC)* with a single interrupt line that is triggered when data arrives. In this case all interrupts have to be handled by a single CPU core. Furthermore, a single TCP connection cannot be handled by multiple CPU cores in the Linux kernel.

Therefore, modern NICs have multiple internal queues. Based on hashing of IP addresses and port numbers, packets of the same TCP flow are always enqueued in the same queue. Each of the queues has its own interrupt. This kind of hardware support makes it possible that different TCP flows can be handled by different CPU cores, if they are hashed into different queues.

At the examined hardware a single CPU core is not able to handle 100 Gbit/s. However, after intensive tuning (cf. section III), 100 Gbit/s could be achieved with three TCP flows. Figure 1 shows the throughput between a single sender and a single receiver. In the beginning there is only a single TCP flow, achieving about 43 Gbit/s. A second and a third flow are started after 30 s and 60 s.

This experiment was conducted in the simplified testbed setup shown in fig. 2. A detailed description of the hardware is given at the end of this section (section II-B). Since all links have a capacity of 100 Gbit/s, there cannot be any congestion in the network. Unexpectedly, we still saw packet losses. After excluding the switch as the reason for the packet losses, we could trace them down to the receiver. A detailed explanation how this can happen is given in section IV.

### A. Performance Characteristics of Senders vs. Receivers

In order to track down how an overloaded sender differs from an overloaded receiver, we set up a flexible testbed, shown in fig. 3. It consists of four servers, each equipped with either a single port or a dual port 100-gigabit Ethernet NIC. The switch is partitioned via VLANs into two logical switches. A

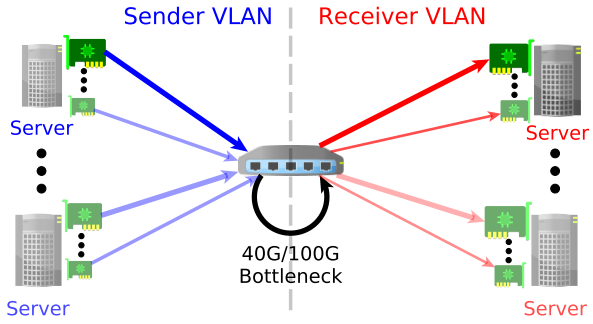



Fig. 3: Flexible testbed configuration

physical cable connects the two logical switches and acts as the bottleneck link. This way we get a logical dumbbell topology. Dumbbell topologies are often used for congestion control experiments. However, our logical dumbbell allows us to easily reconfigure the setup; i.e., changing to which of the logical switches a server is connected to, without any manual rewiring. Experiments with a single sender are always conducted on an end-system with a dual-port NIC (both ports in use), so that the sending interface is never the bottleneck.

In this testbed we conducted experiments in the following four setups:

1) Two-hosts: Single sender, single receiver (similar to fig. 1).
2) Three senders, one receiver; i.e., potentially overloaded receiver
3) One sender, three receivers; i.e., potentially overloaded sender
4) Two senders, two receivers; i.e., reduced load on sender and receiver

Figure 4 shows the throughput and the number of retransmission events per second in the just mentioned setup. In all cases four concurrent TCP flows were used in total (e.g., in case of three senders/receivers, one of the hosts handles two flows, the others a single flow, each). Each experiment was repeated 20 times, the error handles on the bar chart shows the standard deviation; the retransmission events are shown as box-plots. If packet loss occurred, often multiple packets were lost at once. Therefore, we combined all packet losses that occurred within 100 ms as a single *retransmission event*. (This covers several RTTs, which is not ideal. However, 100 ms was the highest resolution in which retransmissions could be reported by the traffic generator `iperf3`. We found it to be sufficiently high to show the frequency of packet loss in the given cases and how they correlate with the throughput.)

The experiments show that a single sender can reproducibly achieve 100 Gbit/s, if the receivers are not (or only slightly) overloaded. In experiment 3 (three receivers), the number of retransmission events are significantly lower than in experiment 1 (one receiver). But even though the number is quite low, it is still above zero. Overloading the receiver leads to large amounts of packet losses. Experiments 1 and 2 show an increased number of packet losses and a reduced throughput.

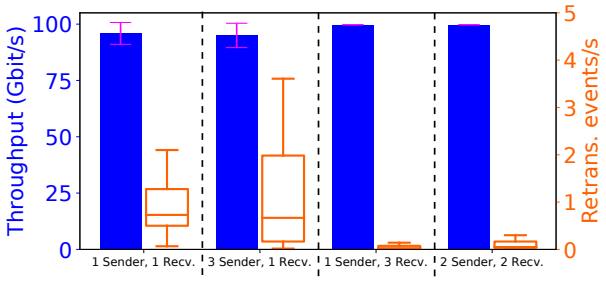



Fig. 4: Performance with different numbers of senders/receivers

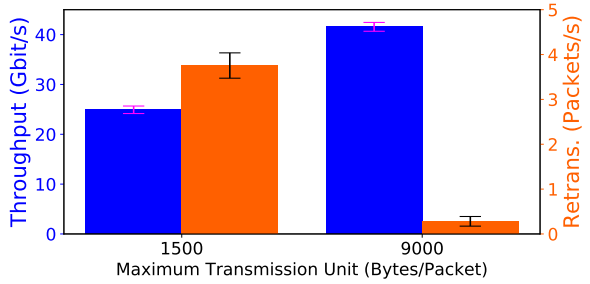



Fig. 5: Impact of the MTU on single flow performance

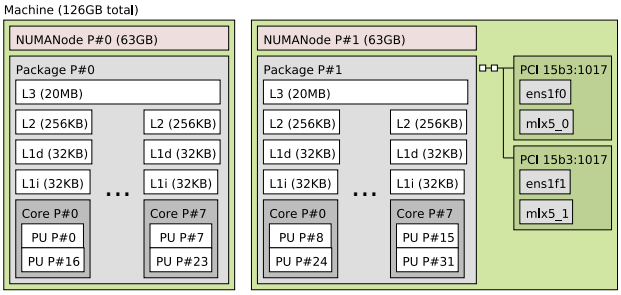



Fig. 6: Memory / NUMA Configuration

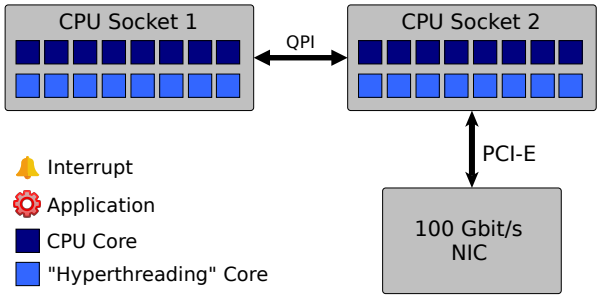



Fig. 7: Multi-core, multi-socket architecture

Experiment 4 (reduced load on sender and receiver) shows high and stable throughput and low loss rates (error bars almost non-existent). But it has to be noted that one sender with three receivers shows a better performance (i.e., lower loss-rates) than two senders with two receivers. This means that an overloaded receiver is the weak spot for high speed data transmissions.

Along with the load that is created by the data rate itself, the *per-packet overhead* is a significant factor, as well. Figure 5 shows throughput and retransmissions/s of a single flow with regular packet sizes (1500 Bytes MTU) and so-called *Jumbo frames* (9000 Bytes MTU). With regular sized packets only 25 Gbit/s can be achieved in conjunction with significantly increased loss rates. Jumbo frames show a much better performance in throughput and loss rates.

### B. Hardware

Our testbed consists of the following hard- and software: Four servers each equipped with a *Supermicro X10DRW-i* mainboard. Two servers have an Intel Xeon E5-2630 v3 @ 2.40GHz (dual socket) CPU, the other two the slightly faster E5-2640 v3 @ 2.60GHz (dual socket) CPU. As NICs, Mellanox ConnectX-5 (MT27800) dual-port and Mellanox ConnectX-4 (MT27700) either single or dual port were in use. We did not observe any differences in performance in our experiments. All NICs used the driver: *MLNX OFED LINUX-4.5-1.0.1.0*. The servers used the operating system with Ubuntu Server 18.04 with Linux kernel 4.15.0-38-generic/4.15.0-42-generic. Hardware offloading (TSO, GSO, GRO) is enabled by default, additional tuning parameters are shown in table I (section III). Non-Uniform Memory Access (*NUMA*) is enabled, hardware details are shown in fig. 6; `ens2f0` and `ens2f1` are the two interfaces of the dual-port ConnectX-5 NIC. The figure shows the output of `lstopo` of one of the servers (slightly modified for better readability). The other servers are built similarly.

As switch a *DELL EMC S4248FB* with *OPX-2.3.1* was used. It consists of six 100 Gbit/s ports, two 40 Gbit/s ports and 40 10 Gbit/s ports. A noticeable feature of this switch is its deep packet buffer of 6 GByte. In our experiments about 1.25 GByte were allocated to the bottleneck, which results in a maximal queuing delay of approximately 100 ms. Each server had one NIC port connected with 100 Gbit/s to the switch. Since the number of 100 Gbit/s ports are limited, two of the dual-port NICs were also connected to the 40 Gbit/s ports. This enables experiments with a single sender with a capacity $> 100\,Gbit/s$. If not noted otherwise, Linux's default congestion control CUBIC TCP [9] was used. `iperf3` was used as traffic generator.

## III. Tuning

On multi-core, multi-socket systems, special attention has to be given to the interrupt distribution and the placement of the sending/receiving applications. An exemplary architecture is depicted in fig. 7 (which resembles our hardware shown in fig. 6). The selection of the CPU socket (section III-A) shows the largest impact. But the distribution of applications and interrupts on the CPU cores (i.e., within the socket) is also relevant (section III-B). Furthermore, a number of additional tuning parameters are presented in (section III-C).

### A. Selection of the CPU Socket

At 100 Gbit/s, the interconnection between the CPU sockets becomes a bottleneck. In our test systems this is the Intel *QuickPath Interconnect (QPI)*, other multi-socket architectures

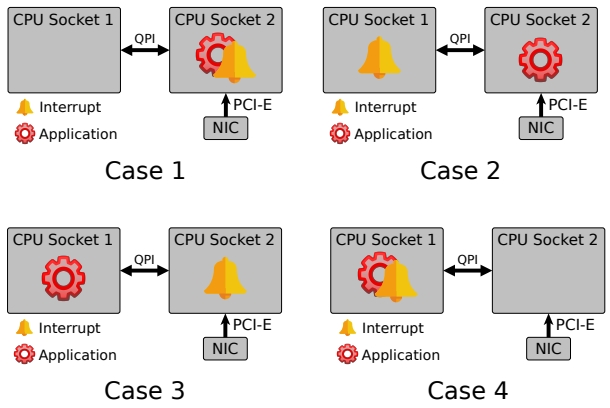



Fig. 8: Placement of interrupts and applications to sockets

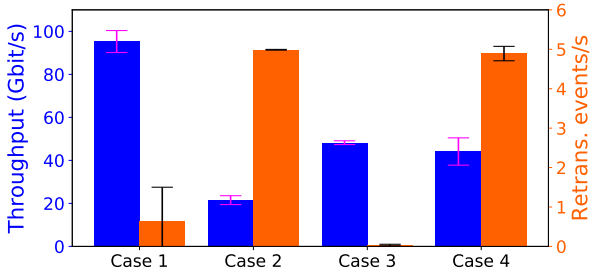



Fig. 9: Effects of placements on CPU sockets

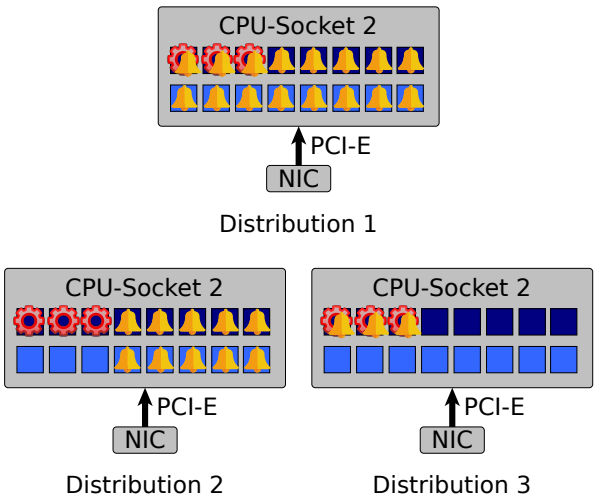



Fig. 10: Placement of interrupts, applications to cores

have similar technologies. By experimentation we measured a capacity of about 68 Gbit/s (in each direction) of the QPI, conducted with the Intel *Memory Latency Checker (mlc)*[1]. This speed depends on the CPU generation and clock speed. Obviously, newer models achieve higher data rates. Still, since NIC speeds are also increasing (400 Gbit/s and even 1 Tbit/s are in reach), this will remain an issue to consider.

Placing the *applications* on the "wrong" socket affects the performance differently than the placement of the *interrupts*. We conducted four experiments where we placed the receiving applications and the interrupts on either of the sockets, as depicted in fig. 8. This shows which individual effect has the placement for applications and interrupts and how they interplay. Within the socket they were placed equally among the CPU cores (cf. fig. 10, distribution 1). Again, the setup shown in fig. 2 (single sender, single receiver, three TCP flows) was used.

The NIC is physically connected to one specific socket. If applications or interrupts are on a different socket, data has to go over the QPI. In our testbed, the 100 Gbit/s NIC was connected to socket 2 over *PCI Express 3 x16*. The PCIe connection did not pose a bottleneck, as we saw in the experiments. But this can also be calculated[2]: $\textit{PCIe bandwidth} = \textit{PCI Width} \cdot \textit{PCI Speed} \cdot \textit{Encoding Scheme} - 1\,\textit{Gbit/s}$. Since an 128b/130b encoding is used, this results in: $\textit{PCIe bandwidth} = 16 \cdot 8\,\textit{GT/s} \cdot 128/130 - 1\,\textit{Gbit/s} \approx 125\,\textit{Gbit/s}$

As expected, best performance can be achieved in case 1, when applications and interrupts are placed on socket 2, i.e., the same socket the NIC is connected to. In many runs, 100 Gbit/s and low loss rates could be achieved, however, the aggregation over 20 runs (as shown in fig. 9) shows a noticeable variance in throughput and loss rate. This will be investigated in section III-B. Case 3 shows the lowest loss rates but also significantly reduced throughput. Here, the interrupts are handled locally, but the applications are on the other side of the QPI. It can be expected that the TCP flow control slows down the sender to match the processing speed of the receiver; in this case: the speed of the QPI. This is exactly the task of the TCP flow control. Note that this does not result in exactly the same speed as measured with the Memory Latency Checker; most likely due to overhead. Case 2 shows a different behavior. Here, the TCP flow control does not seem to work properly. The reasons are discussed in section IV, but in fig. 9 we can already see the effects: a massively increased number of packet losses and a severely reduced throughput. Case 4 shows similar issues but better performance than case 2, most likely due to cache locality. Furthermore, in case 2 we expect that data has to cross the QPI twice (from the NIC on socket 2 to the interrupt handler on socket 1 and then back to the application on socket 2). This means that placing applications and interrupts on the same CPU socket is an advantage. But placing the interrupts on the "wrong" CPU socket can massively increase the number of packet losses.

### B. Distribution to Cores

The throughput variance in case 1 (fig. 9) is caused by inauspicious flow hashing (see section II). Figure 10 (distribution 1) shows that the three cores where the applications are pinned to, also have an interrupt of the NIC assigned. As explained above, ideally each TCP flow is hashed to a different queue and each queue has a separate interrupt. With three TCP flows this means that at most three interrupts are actually in use in the

[1] https://software.intel.com/en-us/articles/intelr-memory-latency-checker

[2] https://community.mellanox.com/s/article/understanding-pcie-configuration-for-maximum-performance

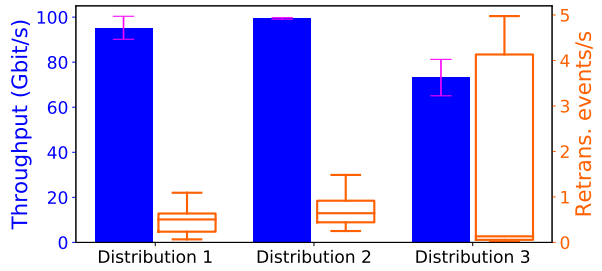



Fig. 11: Effects of distribution to cores

experiments. If, by chance, these three interrupts are assigned to different cores than the applications, the performance increases. In order to back this theory, we conducted the following experiments. As visualized in fig. 10, we moved all interrupts away from the application cores and their hyperthreading twins in distribution 2. Distribution 3 represents the cross check: All interrupts happen on application cores. Distribution 1 is the same setting than in the experiment above (case 1). The results are shown in fig. 11.

With distribution 2, a throughput of 100 Gbit/s was reliably achieved in all repetitions of the experiment (error bars almost non-existent). While the number *retransmission events/s* is slightly higher in distribution 2 than in distribution 1, the number of *retransmissions/s* (not shown in fig. 11) is significantly lower (63 vs. 20 retrans./s). Conversely, distribution 3 shows inferior performance with significantly reduced throughput and a large amount of packet losses. It has to be noted that with only three cores handling the interrupts, the probability that two or even all three flows are handled by the same CPU core is elevated. This explains the large difference between the median and the 75% quantile of the retransmission events (rightmost box plot). For maximal performance and high reliability, we conclude, that applications should not be pinned to cores that actively handle interrupts from the NIC.

### C. Additional Tuning

Besides the selection of the CPU socket and the distribution on cores, a number of additional tuning parameters affect the performance. The usage of jumbo frames (i.e., using a *Maximum Transmission Unit (MTU)* of 9,000 bytes instead of the usual 1,500 bytes) significantly improves the performance. However, jumbo frames are not supported over all paths. TCP Segmentation Offload and the corresponding receive offload (TSO, GSO, GRO), therefore, work with oversized packets only within the end-systems. These oversized packets are then re-segmented at NIC or driver level. However, in our experiments, the combination of jumbo frames and segmentation offload was beneficial over segmentation offload alone. Segmentation and receive offload (TSO, GSO, GRO) are enabled by default and we kept it this way.

The entire set of tuning parameters is shown in table I. In the following, we briefly discuss some of the parameters. By default the boundaries for the TCP flow control are too small for high bdp networks (`[r/w]mem_max, tcp_*mem`). Unlike [8] we observed a reduction of packet loss by enlarging the rx-ring to its maximum size. `tcp_no_metrics_save` is relevant to ensure independent experiment runs; it is no actual tuning parameter. The *qdisc* `fq` is used since it provides an efficient packet pacing feature, which reduces the burstiness of the data stream. As default usually `pfifo_fast` (a simple FIFO queue) or `fq_codel` is set. It has to be noted that `fq_codel` is an Active Queue Management mainly designed to be used in routers and switches, thus, it also may deliberately drop packets within the end-system.

| Attribute | Default | Tuned |
|---|---|---|
| Maximum Transmission Unit | 1500 | 9000 |
| RX-Ring | 1024 | MAX [8192] |
| net.ipv4.tcp_timestamp | 1 | 1 |
| net.core.wmem_max | 212992 | 2147483647 |
| net.core.rmem_max | 212992 | 2147483647 |
| net.ipv4.tcp_mem | 140964 187954 281928 | 2147483647 2147483647 2147483647 |
| net.ipv4.tcp_rmem | 4096 87380 6291456 | 4096 87380 2147483648 |
| net.ipv4.tcp_wmem | 4096 16384 4194304 | 4096 87380 2147483648 |
| net.core.netdev_max_backlog | 1000 | 250000 |
| net.ipv4.tcp_mtu_probing | 0 | 1 |
| net.ipv4.tcp_no_metrics_save | 0 | 1 |
| net.core.default_qdisc | fq_codel | fq |
| CPU Governor | powersave | performance |
| Irqbalance | enabled | disabled |
| TSO, GSO, GRO | enabled | enabled |

TABLE I: Tuning Parameters

## IV. Packet Loss in the End-System

TCP's flow control is designed to protect the receiver from overload. It uses a sliding window that is continuously announced to the sender. This window reflects the available capacity in the TCP receive buffer. Consequently, a packet is only sent if there is already space allocated for it in the receive buffer. Still, we observed packet loss within the receiving end-system, caused by overload. These losses happen at lower layers.

Figure 12 visualizes the path of a packet from the sender to the receiver, including the protocol stacks in the end-systems. Origined in the application, data is copied to the *TCP send buffer*. After the data is segmented into packets, the packets are enqueued into the so-called *Queuing Disciplines (qdiscs)*. When they are scheduled to be transmitted, the packets are copied into the *TX-Ring*, a data structure that can be accessed by the operating system and the NIC (via *direct memory access (DMA)*). In fact, the ring only contains pointers to the packets, which are fetched directly from the main memory. If the TX-ring is full, packets are not lost but stay in the qdisc and can be enqueued at a later time.

In intermediate systems (i.e., routers and switches) packets can be dropped. This usually happens in a congestion situation, e.g., when the buffer at a bottleneck overflows. In this case we speak of a *congestion loss*.

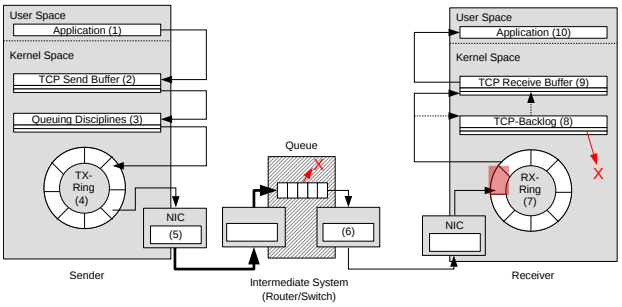



Fig. 12: Packet path including protocol stacks in end-systems

At the receiver, the NIC places the packet into the *RX-ring* (analogous to the TX-ring at the sender). However, there is a distinctive difference. If the RX-ring overflows (i.e., it is still full when new packets arrive), packets are dropped. Since the TCP flow control only manages the TCP receive buffer but not the RX-ring, it cannot prevent these losses; i.e., the RX-ring may be full while there is plenty of space in the TCP receive buffer. Increasing the size of the RX-ring can reduce the number of packet losses. Packet loss in the RX-ring is logged; thus, we can distinguish these losses from other kinds of losses (e.g., caused by bit errors) in our experiments.

Another buffer that is not managed by the TCP flow control is the *TCP backlog*. This is an auxiliary buffer for situations when the receive buffer data structure is locked (i.e., to prevent concurrent read and write access). Packets can also be lost if the TCP backlog overflows. These losses are also logged.

Once the packet reaches the TCP receive buffer, it is safe. In normal operation this buffer never overflows. There are rare cases when even the TCP receive buffer may overflow, despite TCP flow control. Since the flow control window announces free buffer space for *payload* data, the actual buffer has to be sized larger to accommodate overhead, such as packet headers. If the overhead is unexpectedly large (e.g., many small packets), the buffer may overflow. In practice this corner case was not relevant in our experiments.

## V. Congestion Control

For congestion control algorithms, 100 Gbit/s networks are challenging due to several reasons. As mentioned above, a higher maximum speed increases the range of possible network speeds. Moreover, the following points create an area of tension:

- Non-congestion related packet loss can happen even in wired networks.
- We do not want an over cautious congestion control to slow down our high volume data transfers.
- We do not want a 100 Gbit/s sender with an aggressive congestion control to overstrain slower networks.

The issue of non-congestion related packet loss is already known from shallow buffered switches and middleboxes. The *Science DMZ* [3] approach advises to create special zones at the edge of the network for high volume data transfers. The goal is to eliminate as many sources for non-congestion related packet loss as possible. The rationale is that each packet loss can significantly slow down data transfers over wide area networks. Google noticed that commodity switches are often *shallow buffered*. This means that the switch buffer is only a tiny fraction of the bandwidth-delay product. Small bursts, which are caused by aggregation effects and observed in practice, can already overload the bottleneck buffer. In this case, packet loss actually happens at the intermediate systems, but it is not useful to treat this short-term overload situation as (persistent) congestion. Therefore, the congestion control *BBR* [2] was developed. It addresses the situation in two ways: 1.) The outgoing data is *paced*. This means the burstiness of the outgoing data stream is reduced and data is sent at an even rate. 2.) Packet losses are no longer considered an indicator for congestion.

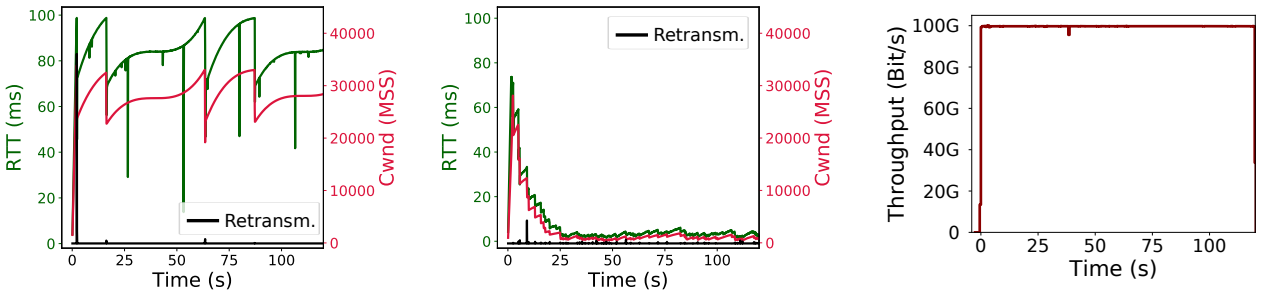
BBR, as designed in [2], is a very aggressive congestion control. Experiments have shown that this approach is in fact able to avoid underutilizing a fast network. However, BBR can also cause massive amounts of *congestion related* packet loss, i.e., massively overload a slower network [7].

In our own experiments we used a deep buffered switch (buffer size: 1.25 GByte $\widehat{=}$ 100 ms queuing delay), in order to focus on additional sources of packet loss, apart from shallow buffered switches. Having discovered packet loss within the receiving end-systems, shows that the *Science DMZ* approach alone is not sufficient, since a *Science DMZ* only minimizes packet loss caused by intermediate systems.

### A. Effects of Packet Loss on Existing Congestion Controls

CUBIC TCP is the default congestion control in all major operating systems (Linux, Mac, Windows) and can achieve very high throughput, if bottleneck buffers are reasonably sized and if packet loss is only caused by congestion (i.e., no random loss). Sections II and III showed that non-congestion related packet loss, indeed, can noticeably impact the throughput of CUBIC TCP even in a LAN setup with a *round-trip-time (RTT)* below 1 ms. With intensive tuning the number of packet losses can be significantly reduced. However, they could not be fully avoided in our experiments. In real world deployments, we expect that administrators will often not have the time for such an extensive tuning. Therefore, we expect noticeable loss-rates to be common for high-speed data transfers.

In all following experiments, tuning as shown in table I is applied. Figures 13a and 13b show the behavior of a CUBIC TCP flow under different circumstances. The plots show the CWnd and RTT of a single TCP flow; in total there were four TCP flows active in parallel. Full link utilization was achieved in both cases. In fig. 13a two senders and two receivers were used and the receivers produced a very low amount of packet losses. CUBIC TCP's distinctive shape of the CWnd can be clearly seen. As expected for CUBIC TCP, the bottleneck buffer is repeatedly filled up to exhaustion, then, the CWnd is reduced. This can be seen at the course of the RTT. Due to the queuing delay the RTT is increased. At an RTT of about 100 ms the buffer is exhausted and a *congestion related* packet loss happens. In fig. 13b only a single receiver was used which produced a larger amount of packet losses. Even though 100 Gbit/s could be achieved (cf. fig. 13c), it is




(a) With only congestion losses

(b) With non-congestion losses

(c) Throughput (with non-congestion losses)

Fig. 13: CUBIC TCP (LAN)

evident that CUBIC TCP's behavior is completely dominated by the lossy receiver, so the CWnd is reduced to comparatively low values. Only a small queuing delay builds up. There are *no congestion related packet losses*, at all.

It can be argued that the lossy receiver is a blessing for the overall performance due to the reduced delay. In this very experiment this is actually true. But the experiment shows that the workings of the congestion control is impaired. CUBIC TCP is not able to raise its CWnd over a certain level. But on wide area networks with larger RTTs, larger CWnd values are *required* to achieve the same throughput (i.e., $\overline{rate}_{sent} = \frac{CWnd}{RTT}$). It can be expected that the data transfer is slowed down well below 100 Gbit/s at higher RTTs. In order to confirm this conclusion we set-up a delay emulator that emulates a WAN network with 20 ms base-RTT (RTT if buffers are empty). The same delay emulator[3] as in [6] and [7] was used. Since it only supports 10 Gbit/s, we configured asymmetric routing in our testbed. The delay emulator was only included on the path from the receivers to the senders. Since only ACKs were sent over this path, a capacity of 10 Gbit/s was more than enough (typical data rate on the reverse path: 50 Mbit/s). Figure 14 shows a run with two senders (two flows per sender) and one receiver that creates a noticeable amount of *non-congestion related packet loss*. It can be clearly seen that the CUBIC TCP flows are affected by these packet losses. The CWnds of all four flows (fig. 14a) are often reduced even though the bottleneck buffer and/or the bottleneck link have residual unused capacities. Consequently, the link utilization often falls well below 100 Gbit/s (fig. 14b).

### B. Novel Congestion Controls

*1) BBR:* BBR was not able to achieve 100 Gbit/s in the same experiment setup as above (no delay emulator, four TCP flows, tuning from table I applied). This was caused by a bug in the BBR code in the Linux kernel v4.15 which we used. It limited the single flow throughput in low RTT networks to $\approx 10\,Gbit/s$. With $> 10$ flows, 100 Gbit/s could be achieved. This shows that 100 Gbit/s are also a challenge on the implementation level of congestion control algorithms.

Apart from this issue, BBR works exceptionally well in this setup. Due to the small *base-RTT* of $< 1$ ms, BBR creates only a very small queuing delay, yet it is still able keep a high throughput (see fig. 15a).

However, in earlier experiments at 10 Gbit/s [7] it was shown that BBR will behave different at larger base-RTTs: BBR often produces a queuing delay that roughly equals the base-RTT; hence, doubling the effective RTT. Indeed, experiments with delay emulator (20 ms base-RTT) show that the same behavior can be observed at 100 Gbit/s (cf. fig. 15b). The noticeable bandwidth drops every 10 s are caused by BBR's "Probe RTT" phase where the amount of inflight data is reduced to four packets, in order to drain the queues. Apart from that, BBR achieves full link utilization, but also causes significant queuing delay. BBR's aggressiveness is most pronounced when the buffer capacity is below one *bdp* [7]. To solve these issues, Google has announced to work on "BBRv2".

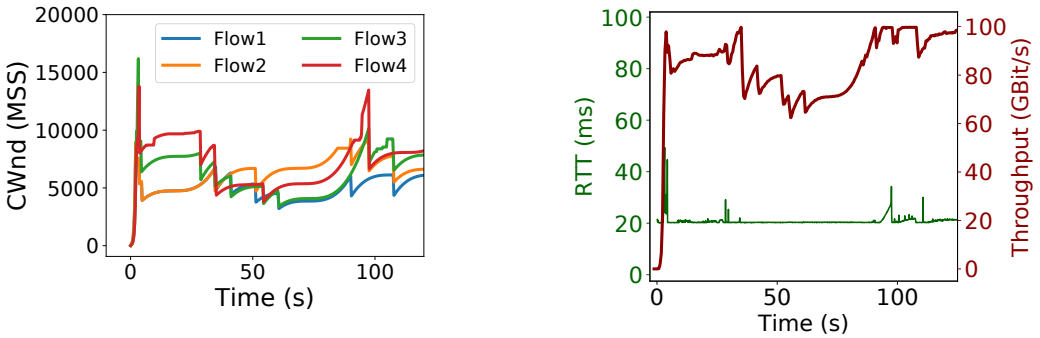
*2) TCP LoLa:* TCP LoLa [6] is another experimental congestion control that is under active development. TCP LoLa's goals are to enable a high throughput while keeping a low queuing delay as well as to provide a convergence to fairness among competing TCP LoLa flows. Due to its *delay based* approach it can be considered less aggressive than most other congestion controls, including BBR and CUBIC TCP.

As shown in fig. 16a, TCP LoLa limits the queuing delay slightly above 5 ms ($RTT \approx 6$ ms; LAN setup, low loss rate). In contrast to BBR, the induced queuing delay of TCP LoLa is a configurable parameter and does not depend on the base-RTT. Thus, it is larger than BBR's queuing delay at very low base-RTTs, but will not grow in wide area networks.

Since TCP LoLa detects congestion based on queuing delay, it does not rely on packet loss as congestion signal. But as a precaution, TCP LoLa still reacts on packet loss similar to CUBIC TCP. Therefore, TCP LoLa is negatively affected by non-congestion related packet loss, too. Experiments using the delay emulator (20 ms base-RTT) show that this can significantly reduce the throughput in a WAN setting (see fig. 16b).

In order to investigate the potential of TCP LoLa's delay based approach in the face of lossy end-systems, we experimentally modified TCP LoLa to not decrease its CWnd on packet loss. Figure 16c shows improved stability and high link utilization in face of non-congestion related packet loss.

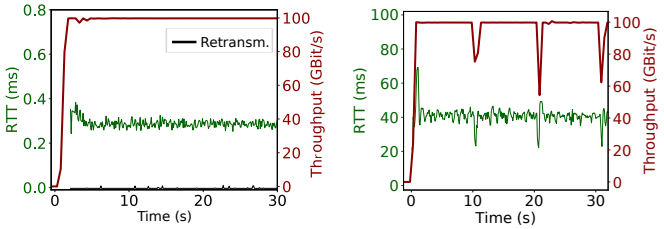
[3] https://git.scc.kit.edu/TM/DPDK_AQM_Switch




(a) CWnds react on non-congestion related packet loss

(b) Underutilization, due to non-congestion related packet loss

Fig. 14: CUBIC TCP (WAN)




(a) LAN

(b) Emulated WAN

Fig. 15: BBR

Figures 16d to 16f show the CWnds in the above experiments. The LAN scenario without losses (fig. 16d) is shown for comparison. For better readability only a single flow is shown. Due to the losses in fig. 16e TCP LoLa falls back to a CUBIC TCP-like behavior, while in fig. 16f the distinctive TCP LoLa behavior is mostly restored. Thus, the CWnds oscillate in a small corridor that keeps the link fully utilized but the queuing delay low. Also the CWnds of the four flows approach each other, due to TCP LoLa's fairness properties.

This experiment shows the potential of TCP LoLa and other non loss-based approaches to be well suited for high volume data transfers at high speeds. However, it has to be kept in mind, that simply ignoring packet loss as congestion signal is not expedient, as shown in the case of BBR. TCP LoLa, for example, defines a queuing delay above 5 ms as congestion. On shallow buffered switches such a queuing delay can never be achieved. Therefore, further research is needed to detect and adapt to shallow buffered bottlenecks.

## VI. Related Work

*High-speed data transmissions* is an ongoing field of research, that has to adapt to altering challenges that come with each new hardware generation. Since the performance of different hardware components evolves in distinct ways, this is not a simple scale-up. The *Fasterdata Knowledge Base*[4] from ESnet is a valuable source for high performance data transmissions and tuning, as well as experimental evaluations such as [8]. Along with other tuning guides[5] the information is often focused on *how* a good performance can be achieved. We built on this work and extended it toward the questions: *Why* is a certain configuration superior and *what* does this mean for further research?

There is recent focus on improving the efficiency of network operations by bypassing the operating system kernel, such as Intel DPDK[6], Netmap [10], or XDP[7]. These approaches are very effective for certain kinds of tasks. The *packet generator* MoonGen [5], for example, built upon DPDK is able to generate packets significantly faster than traffic generators (like `iperf3`) or real world applications using TCP. But since these performance improvements stem from bypassing the operating system's networking stack, they lack, i.a., TCP protocol features and congestion control.

The *Science DMZ* [3] approach is used in practice to improve the throughput for high volume data transfers of scientific data. It focuses on eliminating all obstacles within the end-to-end path that may cause packet loss. Our research assesses the components in this path that cannot be eliminated: the end-systems. GridFTP[8] is often used in practice to parallelize large file transfers. Such tools can directly benefit from our results.

Congestion control research is, again, a very active field of research. BBR [2], L4S [1], PCC [4], and TCP LoLa [6] are under active development and under ongoing discussion in the IETF/ICCRG. We consider our research a valuable input into these discussions.

## VII. Conclusion

Network bandwidth grows significantly faster than single core CPU performance. Therefore, end-system tuning becomes increasingly important to exploit the available transmission capacities. In this paper, we brought existing server hardware to its performance limits and investigated which bottlenecks arise in the end-systems. We showed that overloaded receivers impact

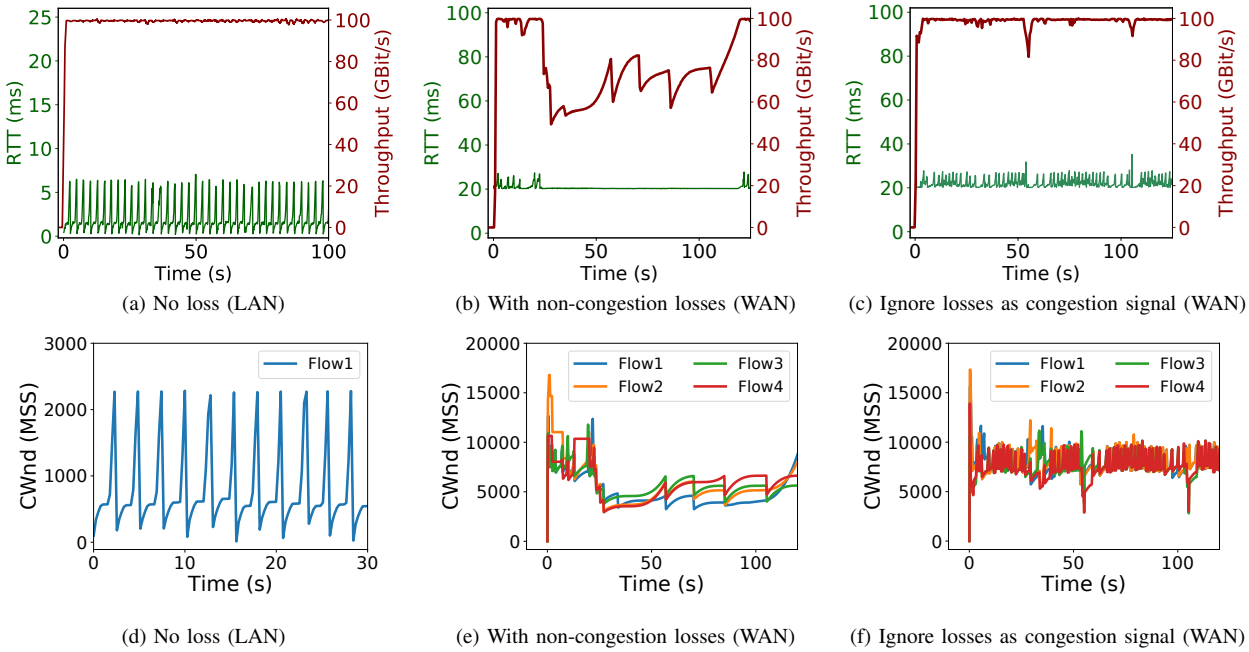
[4] http://fasterdata.es.net/

[5] https://access.redhat.com/sites/default/files/attachments/20150325_network_performance_tuning.pdf, https://community.mellanox.com/s/article/performance-tuning-for-mellanox-adapters

[6] https://www.dpdk.org/

[7] https://www.iovisor.org/technology/xdp

[8] http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/




(a) No loss (LAN)
(b) With non-congestion losses (WAN)
(c) Ignore losses as congestion signal (WAN)
(d) No loss (LAN)
(e) With non-congestion losses (WAN)
(f) Ignore losses as congestion signal (WAN)

Fig. 16: TCP LoLa

performance stronger than overloaded senders. For multi-socket CPUs, hardware specific tuning is required. Internally, the network interface card is usually connected to one of the sockets directly. Placing interrupts or applications on a different socket results in massive performance penalties. Misplacing the applications leads to a common pattern: Since the processing power of the receiver is below the network capacity, the TCP flow control slows down the transmission rate. But we showed that this mechanism does not work reliably if the interrupts are placed on the wrong socket. In this case, a significant amount of packets are dropped within the receiving end-system. Such packet losses, in turn affect the congestion control. CUBIC TCP, the default congestion control in all major operating systems, cannot cope with such non-congestion related packet loss. This limits the achievable performance, especially in wide area networks. Newer approaches exist that may tolerate non-congestion related packet loss. But, additional research is necessary to reliably serve 100 Gbit/s and beyond. Moreover, there is a field of tension about the balancing between aggressively utilizing available capacity, e.g., even in the face of packet loss, and carefully avoiding to overload slower network paths. Congestion control research that looks beyond packet loss as congestion signal seems to be on the right track, but care has to be taken if packet losses *are actually* congestion related.

## 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] B. Briscoe, K. D. Schepper, and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture," Internet Engineering Task Force, Internet-Draft draft-ietf-tsvwg-l4s-arch-03, Oct. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-03

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, no. 5, pp. 50:20–50:53, Oct. 2016.

[3] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science dmz: A network design pattern for data-intensive science," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 85:1–85:10. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503245

[4] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 395–408. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong

[5] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Internet Measurement Conference 2015 (IMC'15)*, Tokyo, Japan, Oct. 2015.

[6] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion control for low latencies and high throughput," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 215–218.

[7] M. Hock, R. Bless, and M. Zitterbart, "Experimental Evaluation of BBR Congestion Control," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Oct 2017.

[8] B. T. Nate Hanford, "Recent Linux TCP Updates, and how to tune your 100G host," https://fasterdata.es.net/assets/Papers-and-Publications/100G-Tuning-TechEx2016.tierney.pdf.

[9] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks," RFC 8312 (Informational), RFC Editor, Fremont, CA, USA, pp. 1–18, Feb. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8312.txt

[10] L. Rizzo, "netmap: A novel framework for fast packet i/o," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association, 2012, pp. 101–112. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/rizzo