TCP ROCCET: An RTT-Oriented CUBIC Congestion Control Extension for 5G and Beyond Networks

Lukas Prause and Mark Akselrod
Institute of Communications Technology
Leibniz Universität Hannover
{lukas.prause, mark.akselrod}@ikt.uni-hannover.de

***Abstract*—The behavior of loss-based TCP congestion control algorithms like TCP CUBIC continues to be a challenge in modern cellular networks. Due to the large RLC layer buffers required to deal with short-term changes in channel capacity, the behavior of both the Slow Start and congestion avoidance phases may be heavily impacted by the lack of packet losses and the resulting bufferbloat. While existing congestion control algorithms like TCP BBR do tend to perform better even in the presence of large bottleneck buffers, they still tend to fill the buffer more than necessary and can have fairness issues when compared to loss-based algorithms.**

In this paper, we analyze the issues with the use of loss-based congestion control algorithms by analyzing TCP CUBIC, which is currently the most popular variant. To mitigate the issues experienced by TCP CUBIC in cellular networks, we introduce TCP ROCCET, a latency-based extension of TCP CUBIC that responds to network congestion based on round-trip time in addition to packet loss.

Our findings show that TCP ROCCET can reduce latency and bufferbloat compared to the standard CUBIC implementation, without requiring a specific network architecture. Compared to TCP BBRv3, ROCCET offers similar throughput while maintaining lower overall latency. The evaluation was conducted in real 5G networks, including both stationary and mobile scenarios, confirming ROCCET's improved response to network congestion under varying conditions.

I. Introduction

As the maximum data rates achievable in current Radio Access Networks (RANs) continue to grow, the size of the Radio Link Control (RLC) layer buffers that is needed to deal with bursty TCP traffic and short-term signal quality variations also increases. While the larger RLC layer buffer size is required when dealing with the traffic when the signal quality is at its highest, long-term decreases can lead to a standing queue at the RLC layer buffer, resulting in bufferbloat.

In cases where the available bandwidth or the capacity of a link decreases, loss-based congestion control algorithms like TCP CUBIC [1] expect the bottleneck buffer to start dropping packets as their sending rate exceeds the rate at which the packets can be processed. However, the deep RLC layer buffer tends to just be filled up by the initially excessive sending rate, and then the sender self-adjusts their sending rate to the ACK-rate. In the case that the initial excessive sending rate is not enough to cause a packet loss, TCP's self-clocking prevents the buffer from ever being drained of the excessive packets.

This problem can be particularly severe in current Non-Standalone-New Radio (NSA-NR) networks: In such deployments, a User Equipment (UE) is typically connected to multiple carriers at the same time. As the UE changes its position, it can move out of range of one of the connected carriers, thus losing all of its bandwidth. This can lead to severe changes in the channel capacity, especially if the connection to the NRcarrier is lost. We have previously explored this problem in a major commercial NSA-NR network by conducting a mobile and stationary measurement campaign [2]. However, this problem is not limited to NSA-NR networks, since LTE is expected to continue to be used as a fallback for NR, and even in NR-only scenarios, multiple NR carriers can also be connected using carrier aggregation.

Many different ways have been proposed to deal with this problem. On the sender's side, novel congestion control algorithms like TCP Bottleneck Bandwidth and Round-trip time (BBR) [3] or Copa [4] adjust their sending rate in a way that tries to avoid filling up the buffer. Active Queue Management (AQM) algorithms like RED [5] or CoDel [6] can be deployed at the RLC layer buffer to try to drop or mark packets if the buffer is too full or the sojourn time is too high. Finally, some smartphone manufacturers limit the receive window when a cellular connection is used, thus limiting the maximum size of the sender's send window and the amount of bufferbloat.

In this paper, we propose a delay-based extension to the popular loss-based CUBIC algorithm called RTT oriented CUBIC congestion control exTension (ROCCET). In contrast to using an AQM method, a sender-based solution does not require the service provider to rely on a particular network architecture to reduce delays. We evaluate ROCCET in both stationary and mobile scenarios. We compare it to both the base CUBIC implementation and TCP BBRv3 [7] as these two are some of the most popular congestion control algorithms currently in deployment. Our measurements show that TCP ROCCET can achieve significantly lower RTTs than CUBIC while maintaining a high throughput. When compared to TCP

ISBN: 978-3-903176-79-9

BBRv3, ROCCET also achieves a similar throughput while keeping the latency lower.

The rest of the paper is organized as follows. In Section II, we provide a short primer on the functionality of TCP CUBIC. In Section III, we discuss the issues that we have observed when using TCP CUBIC in mobile cellular networks. In Section IV, we present ROCCET, our delay-based extension for TCP CUBIC, to mitigate these issues. In Section V, we present our real-world measurement setup that we use to evaluate ROCCET's performance in cellular networks in stationary and mobile scenarios. We present and discuss our stationary and mobile measurement results in Sections VI and VII. In Section VIII, we evaluate TCP ROCCET's bandwidth sharing behavior with CUBIC and BBRv3. In Section IX, we present a detailed view of how TCP ROCCET behaves when the capacity of a link changes, when compared to TCP CUBIC and BBRv3. We discuss the implementation of the Kernel source code, the limitations of our algorithm, and considerations for alternative parameter choices to tune its behavior in Section X. Section VIII evaluates ROCCET's fairness compared to itself, TCP CUBIC, and TCP BBRv3. Finally, Section XI concludes the paper and provides an outlook on further development and rollout steps.

II. A Short TCP CUBIC Primer

In this section, we provide a short primer on TCP CUBIC, as TCP ROCCET is based on TCP CUBIC and extends its functionality. TCP CUBIC is an extension of TCP Binary Increase Congestion control (BIC) [8]. BIC was introduced to better utilize high-speed networks with large Bandwidth-Delay Products (BDPs). Without going into detail, BIC uses binary search to find a congestion window (cwnd) for the available bandwidth. For that, the cwnd before a congestion event (CE) W_{max} and the cwnd after a reduction W_{min} are considered. If the calculated increase is larger than a predefined maximum increment per RTT S_{max}, S_{max} gets added to the cwnd size instead of the calculated increment. If the cwnd size exceeds W_{max}, BIC enters a "max probing" phase. In this phase, BIC uses a cwnd growth function which is symmetric to the growth function before. A downside of this cwnd calculation is that it is too aggressive for short RTT or low-speed networks. At this point, TCP CUBIC provides a solution for these problems by approximating the cwnd growth function of BIC with a cubic function. TCP CUBIC adopts the value of W_{max} and uses it as the saddle point for the cubic cwnd growth function. This results in the cwnd being adjusted more cautiously when it approaches the point of the last CE, and increasingly more aggressively the further away the cwnd is from W_{max}. In addition, TCP CUBIC's cwnd growth function is independent of the RTT, because only the time since the last CE is considered. The characteristic cwnd growth function of TCP CUBIC is shown in Fig. 1 and the corresponding equation is shown in Eq. (1). The parameters β and C are implementation dependent, β is the reduction factor of the cwnd after a CE, and C is a scaling factor of the cwnd. In this paper, we refer to the TCP CUBIC implementation

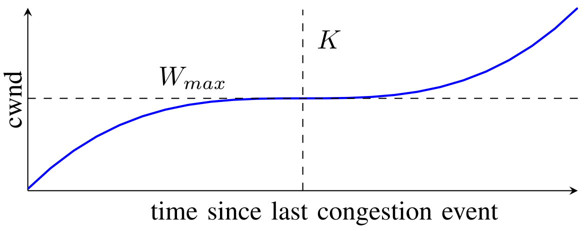


Fig. 1: Characteristic cwnd growth function of TCP CUBIC. With the cwnd size before the last congestion event (W_{max}) as a saddle point. The saddle point is reached at time K after the last congestion event.

of the Linux kernel, where TCP CUBIC additionally uses HyStart [9], an extension of Slow Start. HyStart differs from Slow Start by not only relying on the TCP Slow Start threshold as the beginning of the congestion avoidance phase. HyStart additionally monitors the inter-packet gap of ACKs and the RTT to find an early beginning of congestion, to start into the congestion avoidance phase.

$$W(t) = C \cdot (t - K)^3 + W_{max}$$
$$K = \sqrt[3]{\frac{W_{max} \cdot \beta}{C}} \tag{1}$$

III. Known Congestion Control Issues in 4G and 5G Cellular Networks

In this section, we summarize congestion control issues caused by deep RLC layer buffers and carrier aggregation in an non-standalone (NSA)-NR network, which we have analyzed in detail in our previous work [2]. Furthermore, we highlight problems in the Linux kernel implementation of CUBIC. In our previous measurement analysis, none of the evaluated congestion control variants, i.e., Reno, CUBIC, and BBRv1, were able to reliably react to changes in carrier bandwidth. Especially the deactivation of the wide 5G carrier through carrier aggregation usually did not result in a large enough reduction of the congestion window, which led to a major increase in delays due to the resulting bufferbloat. Similarly, for Reno and CUBIC, an activation of the 5G carrier did not result in a faster growth of the congestion window, leading to wasted link capacity. Small changes in carrier bandwidth also did not always cause an adjustment of the congestion window, resulting either in bufferbloat or unused link capacity.

For the Linux Kernel implementation of CUBIC, we have observed additional issues. CUBIC can use HyStart or Slow Start, and for both, we have observed several problems:

HyStart: is an extension of Slow Start that additionally observes spacings between ACKs to find a good start for congestion avoidance. Because of the nature of cellular networks (shared medium, scheduled, and wireless), inter-packet gaps are not sufficiently meaningful for congestion detection, which means that the use of HyStart most often leads to a premature start of congestion avoidance, as has already been observed [10]–[12]. This premature start of congestion avoidance leads to an underutilization of available bandwidth. Therefore, Slow Start is a better option for transmissions over 5G networks, which leads to the next issue.

Slow Start: doubles the cwnd during each RTT until a threshold is reached or a CE in the form of loss occurs. In the Linux kernel implementation of CUBIC, the initial threshold is the maximum value for an unsigned 32-bit long integer, which in this scenario would be equivalent to infinity. The Slow Start threshold is initially set to this value to prevent too early a start of the congestion avoidance phase. Therefore, a loss has to occur; otherwise, CUBIC won't enter congestion avoidance. Since cellular networks use deep buffers, Slow Start has to fill a buffer that is possibly too large for the connection before a packet loss can occur. If CUBIC can fill the buffer, the resulting loss will often only partially drain the buffer, which can lead to bufferbloat.

Another implementation detail of CUBIC that can lead to problems during Slow Start is that before CUBIC increases the cwnd, it checks if the application layer generates enough data for the current cwnd. If this isn't the case, CUBIC does not increase the cwnd. In other words, the cwnd is frozen until enough data is generated to utilize the current cwnd. This often leads to CUBIC never exiting Slow Start and the cwnd being stuck on a value that is too large for the current connection. In some cases, a loss may occur, but this loss is mostly caused by the L2 mechanism DRX or HARQ, which triggers retransmissions by inter-arrival time jitter. In other words, this loss is not related to congestion. Nevertheless, CUBIC then enters the congestion avoidance phase. In this state, due to the absence of loss, the cwnd is still frozen at a too high value. An exemplary real-world measurement showing the described behavior can be seen in Fig. 2. We choose an example transmission where loss in Slow Start occurs, to additionally show the lock cwnd behavior in congestion avoidance.

The sample transmission is 60 s long. In the first 4 s of the transmission, CUBIC is in Slow Start, which massively overshoots a desirable cwnd. Then a loss occurs, and CUBIC enters the first congestion avoidance phase. At 7 s the next loss occurs, and after a fast retransmit and fast convergence, CUBIC stays in the congestion avoidance phase. During the one-minute-long transmission, except for the first few RTTs of Slow Start, the cwnd is frozen at a value that is too high.

IV. TCP ROCCET

Taking into account the issues we mentioned in Sec. III, we developed an RTT-oriented CUBIC congestion control extension (ROCCET). ROCCET relies on the change of Smoothed Round Trip Time (sRTT), which is an internal kernel calculation of the ACK RTT smoothed by an Exponentially Weighted Moving Average (EWMA), and the amount of ACKs arriving during a time period. With our implementation of ROCCET, we solve the locked cwnd of CUBIC in Slow Start and congestion avoidance. We extend the current Linux kernel implementation of CUBIC by adding two additional metrics for detecting congestion on the network. The first metric is the

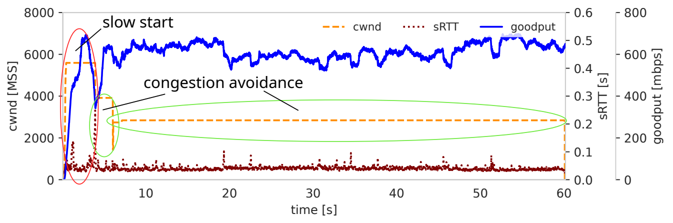


Fig. 2: TCP CUBIC first gets stuck in Slow Start and then in congestion avoidance during an unwanted side effect of the Linux kernel implementation. The two loss-based CE in the first seven seconds of the transmission are caused by L2 mechanisms and are not related to congestion.

smoothed relative RTT (srRTT). For this, ROCCET calculates the relative change of the sRTT (RTT_{curr}) compared to the measured minimum sRTT (RTT_{min}). After this calculation, the result gets smoothed using an EWMA with α as weight factor. This smoothing results in the srRTT. The calculation is shown in Eq. (2). For the required RTT_{min}, ROCCET compares each ACK RTT with the current RTT_{min}. If the current RTT is lower than RTT_{min}, it gets updated.

The second metric is the amount of ACKs that were received over a time interval, compared to the cumulative sum of the last cwnds. We further call this sum of cwnds *cum_cwnd*. The *cum_cwnd* is calculated over the same time interval as the monitoring of the amount of ACKs. The length of the time interval depends on the current phase.

$$\begin{aligned} x_t &= \frac{RTT_{curr} - RTT_{min}}{RTT_{min}} \\ srRTT_t &= \alpha \cdot x_t + (1-\alpha) \cdot srRTT_{t-1} \end{aligned} \tag{2}$$

Using these metrics, ROCCET can be partitioned into the phases: Latency-Aware Utilization for Network Congestion Handling (LAUNCH), which is an extension of Slow Start, and the congestion avoidance phase Optimal Rate Balancing with Incremental Throughput Enhancement and Reliability (ORBITER). A diagram of the mechanism is shown in Fig. 3.

LAUNCH For the exit out of the Slow Start, ROCCET tracks the amount of incoming ACKs and the cum_cwnd over 100 ms intervals. If the difference of the incoming ACKs and the cum_cwnd is larger than ten segments, and in addition the srRTT is more than 100 %, ROCCET will exit Slow Start. If it is the initial Slow Start of the transmission, the cwnd gets halved; otherwise, a CUBIC CE gets generated. In addition, ROCCET ignores loss during Slow Start, since we observed performance issues in this phase. These issues come from a loss not generated by congestion or falsely detected by a retransmission timeout. One of the reasons for this can be an inter-arrival time jitter caused by the 5G L2 DRX mechanism or L2 HARQ retransmissions at the start of the transmission.

ORBITER The congestion avoidance phase is entered after LAUNCH using the window growth function of CUBIC to

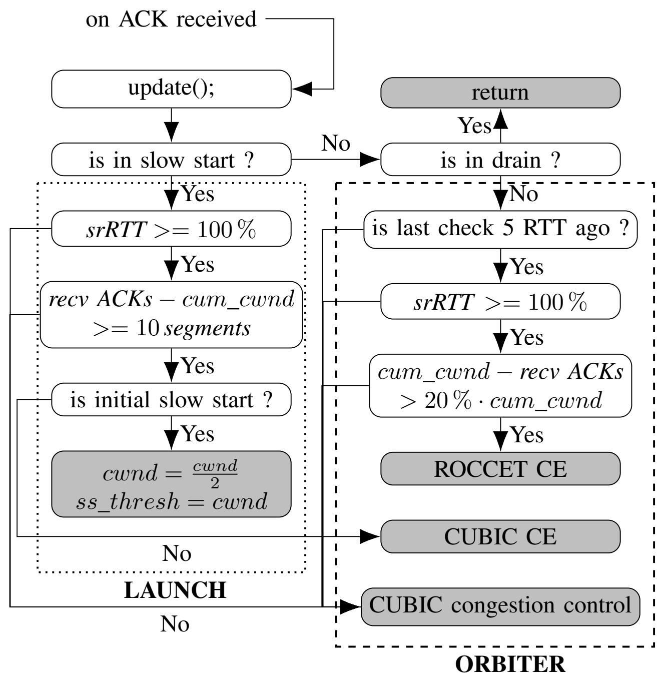


Fig. 3: Schematic flow chart of the ROCCET algorithm. Split in the Slow Start phase extension LAUNCH and the congestion control phase ORBITER. Gray states are end states.

increase the cwnd. If the amount of incoming ACKs over 5 RTTs deviates more than 20 % from the *cum_cwnd* over the same time period, the first condition for a ROCCET CE is fulfilled. For the second condition, the relative change of the RTT must be higher than 100 %. In that case, a ROCCET CE is generated. This CE leads to a cwnd reduction. To create a robust mechanism for cellular networks, the jitter of the connection is also taken into account for the srRTT bound of 100 %. To drain the already bloated buffer, 100 ms after a ROCCET CE occurs, ROCCET does not increase or decrease the cwnd. On these CEs, CUBIC's W_{max} is only set if the current cwnd is larger than the old W_{max}. This results in a faster growth of the cwnd after an ROCCET CE occurs. On a ROCCET CE, ROCCET reduces the cwnd by the factor β. In addition, if a loss is detected, ROCCET behaves like CUBIC, reducing the cwnd by the factor β and doing fast convergence, if enabled.

V. Real World Measurement Setup

In this section, we present our real-world 5G testbed. For evaluating the performance of ROCCET, we used a *Sierra Wireless EM9293* modem [13]. According to its specifications, the EM9293 can achieve download speeds of up to 4.9 Gbps, and upload speeds of up to 1.3 Gbps. The UE was connected to a *Raspberry Pi 5* running *Debian GNU/Linux 12 (bookworm)* via a USB adapter. For each measurement with *iPerf3* [14], we have collected data on sRTT, the send window, and the cwnd size from the Linux debug module on the sender's side. On the receivers end, we used *tcpdump* to capture TCP traffic

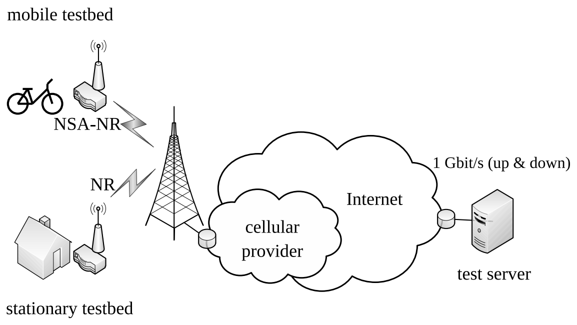


Fig. 4: Measurement topology for mobile (standalone) and stationary downlink (non-standalone) measurements in two commercial NR networks.

and documented the assigned/used NR and LTE component carriers, along with their bandwidths (expressed in MHz). Our two measurement campaigns were conducted within a major commercial NSA-NR and Stand Alone (SA)-NR network in Hannover, Germany. Due to the limited availability of these networks, we conducted our stationary measurements in an SA-NR and our mobile measurements in an NSA-NR network. The measurement setup is illustrated in Fig. 4. For mobile measurements, we selected a bike route that passes through a densely populated urban area with numerous buildings, as well as a large park. Our stationary UE was placed on the 5th floor in an office building. For evaluation, we compare sRTT and achieved goodput from ROCCET, CUBIC, and BBRv3. We used BBRv3 as the third comparator, as BBRv1 is one of the most popular TCP congestion control algorithms, and BBRv3 is the latest version of BBR currently in deployment.

VI. Stationary Measurements Results

To evaluate the performance of ROCCET, we have performed 360 greedy throughput measurements in a commercial NR standalone network. During the measurements, we have alternated between CUBIC, ROCCET, and BBRv3 (120 measurements for each). Each measurement is 30 s long, and since we want to improve the Slow Start performance with CUBIC, HyStart is disabled. Our results are shown in Fig. 5. When we compare ROCCET and CUBIC, it is observable that all sRTTs caused by ROCCET are below the 25 % quantile of CUBIC. A simple explanation for this is that for most transmissions with CUBIC, the cwnd was frozen as we described in Sec III. Therefore, CUBIC was sending data at a steady rate and bloating the RLC buffer, similar to our example in Fig. 2, Sec. III. This is also the explanation for higher goodput with CUBIC, since the RLC buffer is never empty. In other words, CUBIC overfills the pipe and gets every available goodput, which is not detectable by ROCCET. Therefore, ROCCET's median goodput is nearly 10 Mbps lower than CUBIC's. Nevertheless, ROCCET has a better performance than CUBIC, since ROCCET reacts earlier to changes in the network, like congestion, and causes less bufferbloat than CUBIC.

When we compare ROCCET with BBRv3, we observe that both variants can reach nearly the same goodput with a dif-

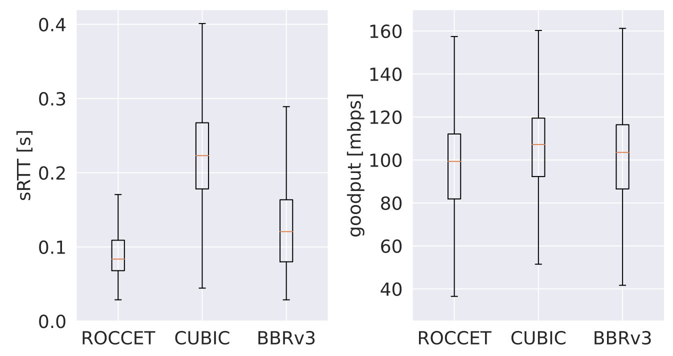


Fig. 5: Goodput and sRTT boxplots for stationary 5G standalone greedy TCP throughput measurements in the downlink direction. Each measurement is 30 s long; during the run, the congestion control was alternated. We did 120 measurements for each congestion control.

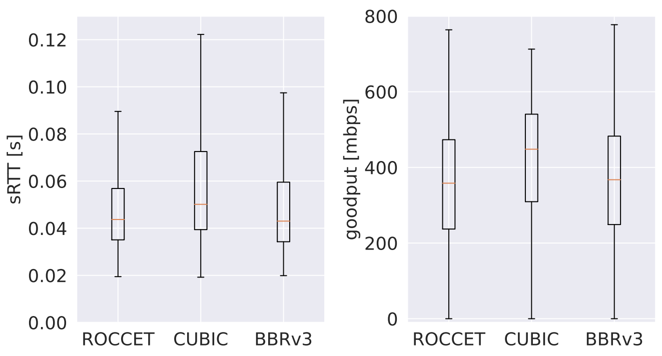


Fig. 6: Goodput and sRTT boxplots for mobile greedy TCP throughput downlink measurements, where at least 90 MHz of carrier bandwidth is available. Each measurement is 20 s long, during the run, the congestion control was alternated. The measurement tour was more than 95 min long, with at least 95 samples per congestion control.

ference of 5 Mbps in median. Also, the 25 and 75 % quantiles for BBRv3 are slightly higher. But when it comes to sRTT 75 % of ROCCET's sRTTs are below the median of BBRv3. In addition, the maximum of ROCCET is slightly higher than the 75 % quantile for BBRv3. This can be explained through the variance of latency in cellular networks. Changes in latency are not only correlated with bottleneck link capacity. Therefore, BBRv3 causes some bufferbloat (but less than CUBIC), which leads to higher data rates and latency.

VII. Mobile Measurements Results

Since we built ROCCET with the main intention of being used in cellular networks, we conduct mobile measurements in a commercial NR-NSA network. As for stationary measurements, we measured greedy TCP throughput. During the measurement ride, we performed 285 measurements. Each measurement is 20 s long, and during each run, the congestion control was alternated. The results are filtered so that at least 90 MHz of carrier bandwidth is available during transmission. We did this for a better comparison because data rates differ a lot if no 5G carrier is available during a measurement. The reached goodput and the respective sRTTs are shown in Fig. 6. First of all, when we compare ROCCET with CUBIC, we can see that we have improved the sRTT. The median and 75 % quantile of sRTTs caused by ROCCET are lower than the 75 % quantile for CUBIC. In addition, the median of ROCCET is also lower than for CUBIC. As for our stationary measurements, the goodput with CUBIC is higher than with ROCCET. The reason for that is the same as in the previous Sec. VII, CUBIC is overfilling the pipe, causing queuing. Another result is that ROCCET and BBRv3 nearly have the same goodput distribution, with the exception that ROCCET causes slightly less latency than BBRv3. The 75 % quantile and maximum of ROCCET are lower than for BBRv3. On the other hand, the 25 % quantile and median for both are nearly the same.

Given these points, we significantly improved CUBIC with our extension ROCCET. In addition, ROCCET has a performance that is at least as good as that of BBRv3.

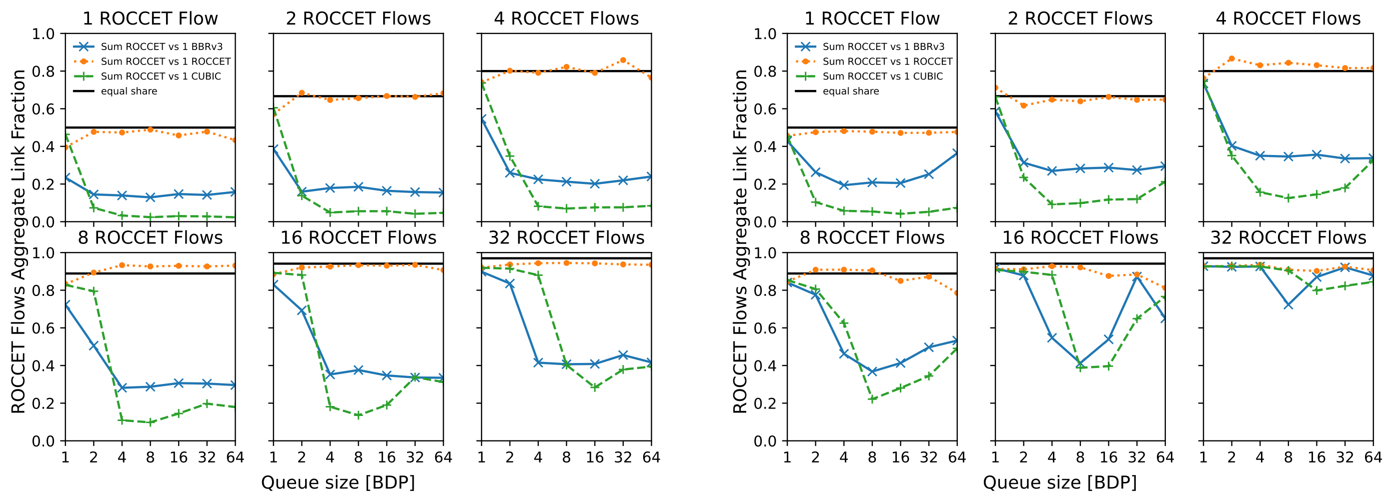
VIII. TCP Bandwidth Share

In this section, we evaluate the bandwidth share of TCP ROCCET in competition with BBRv3, CUBIC, and ROCCET itself. First, we evaluate the bandwidth share between multiple ROCCET flows competing with single BBRv3 and CUBIC flows. Afterwards, we analyse the inter- and intra-bandwidth share of single BBRv3 and CUBIC flows competing with ROCCET. We conducted these experiments based on the work of Ware et al. [15]. With this evaluation, we want to show that ROCCET counts as a deployable congestion control algorithm. Therefore, we conducted our measurements with the link characteristics of *50 Mbps* $\times$ *30 ms* and *10 Mbps* $\times$ *40 ms*, using an Emulab [16] link emulation. We evaluated the bandwidth share at a bottleneck shared link in a dumbbell topology. All links in this topology are wired; therefore, there is no influence on fairness by scheduling.

A. Multi-Flow bandwidth share

For both link scenarios, we measure one competing TCP flow against 1 to 32 ROCCET flows, with a bottleneck link buffer size from 1 to 64 BDP. Since the buffers are deep, we set the receiver's receive window to a larger value than the buffer size. For each configuration, we did five measurements with a duration of two minutes. The results are shown in Fig. 7. Each line represents the summed throughput of 1 to 32 flows from the same sender, which compete with a single BBRv3, CUBIC, or ROCCET flow from a different sender. First of all can be seen that ROCCET has nearly an equal throughput share with other ROCCET flows, for both of our scenarios.

For the scenario *50 Mbps* $\times$ *30 ms*: BBRv3 and CUBIC are getting more bandwidth than the equal share with ROCCET. The bandwidth share with ROCCET is getting worse with the increase in the buffer size. Additionally, CUBIC also gets



(a) *50 Mbps* × *30 ms* (b) *10 Mbps* × *40 ms*

Fig. 7: Bandwidth share of 1 to 32 ROCCET flows competing with a single BBRv3, CUBIC, or ROCCET flow. Link characteristics are *10 Mbps* × *40 ms* and *50 Mbps* × *30 ms* with a bottleneck link buffer size of 1 to 32 BDP.

more bandwidth than BBRv3 when competing with ROCCET flows. One explanation for this is that ROCCET is reacting to the bandwidth probing phase of BBRv3, detecting increasing RTTs and reducing the cwnd. Such behavior is typical for congestion control algorithms that observe changes in RTTs like TCP Vegas [17]. For CUBIC, the share is worse since CUBIC always tries to fill the buffer until a loss occurs. In this case, ROCCET reacts to queuing caused by CUBIC. If the buffer size increases further, the equal share with CUBIC gets better for 8 to 32 competing ROCCET flows.

For the scenario *10 Mbps* × *40 ms*: Nearly the same behavior can be observed as for the previous scenario, with the addition of a better equal bandwidth share with BBRv3 and CUBIC. Also, the bandwidth share is increasing more with the buffer size than for the previous scenario. The conclusion from this is that for a smaller BDP and a larger buffer, the equal share increases.

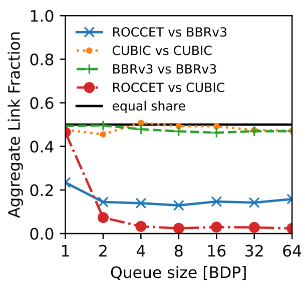
Nevertheless, our results show that ROCCET has nearly an equal bandwidth share with other ROCCET flows for different link characteristics and small to large buffers. On the other hand, BBRv3 and CUBIC obtain a larger bandwidth share than ROCCET when the buffer size exceeds one BDP. This effect increases with the size of the buffer. For cellular networks like 5G, this is not a problem since the network is scheduled; therefore, the bandwidth share depends on the implementation of the scheduler. Furthermore, depending on the implementation of the RLC layer, it is not even necessary that TCP flows for different receivers share the same RLC layer buffer. In addition, it is also possible that on higher layers of the RAN buffers are per single TCP flow. An example of such an implementation can be found in OpenAirInterface [18].

B. Inter- and intra-bandwidth share

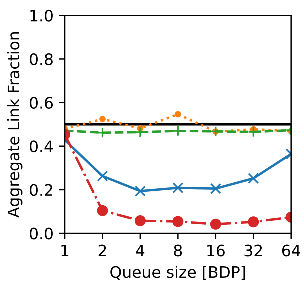
In addition to these bandwidth share experiments we have analyzed in Sec. VIII-A, we conducted detailed analyses of the inter- and intra-bandwidth share behavior for single flows. To this end, we measured the bandwidth share for CUBIC with CUBIC and BBRv3 with BBRv3 and compared the share with a single competing ROCCET flow. We evaluated these measurements for the same link characteristics as before, with a buffer size of 1 to 64 BDP. The bandwidth shares are shown in Fig. 8. Overall, it can be seen that BBRv3 and CUBIC provide a nearly equal share when competing with equal congestion control. For a link of *50 Mbps* × *30 ms*, the equal share of CUBIC with itself is nearly the same as for CUBIC and ROCCET if the buffer has a size of one BDP. Otherwise, ROCCET gets pushed out of the buffer by CUBIC, and the bandwidth share gets significantly worse. For BBRv3, the bandwidth share of ROCCET is always less than 30 % of the link capacity, decreasing with increasing buffer size, but the bandwidth share between ROCCET and BBRv3 is always better than for ROCCET and CUBIC for buffer sizes above one BDP. In addition, the decrease in bandwidth share for one and two BPD-sized buffers is less with BBRv3 than with CUBIC.

For the link characteristic of *10 Mbps* × *40 ms* we observe a better sharing behavior. First of all, BBRv3 and CUBIC have the same bandwidth share with ROCCET as with each other at a buffer size of one BDP. As for our other measurements, the bandwidth share decreases with the increase in the buffer size. The behavior changes at a buffer size of 16 BDP. There, we observe an increase in the bandwidth share with increasing the buffer size.

Our results show that ROCCET is a defensive congestion



(a) *50 Mbps* × *30 ms*



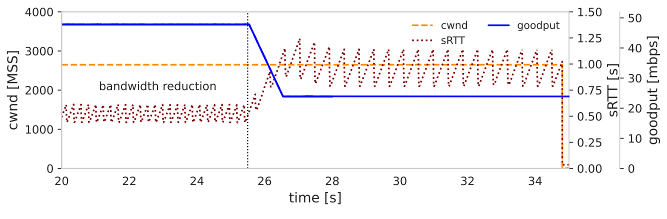
(b) *10 Mbps* × *40 ms*

Fig. 8: Single flow inter- and intra-bandwidth share for BBRv3 and CUBIC with ROCCET for link charakeristics of *50 Mbps* × *30 ms* and *10 Mbps* × *40 ms*.

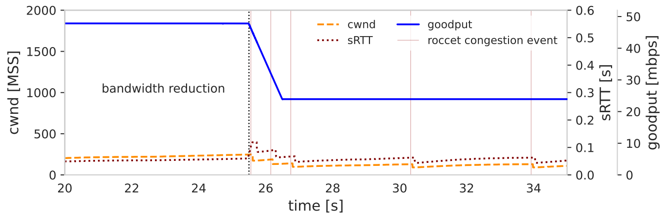
control algorithm that allocates less than an equal bandwidth share if competing with CUBIC or BBRv3 flows in buffers larger than one BDP. On the other hand, ROCCET can provide an equal or less intra-bandwidth share. These points lead us to conclude that ROCCET can be considered as a deployable congestion control algorithm since it causes less harm to other congestion control algorithms than other congestion control to themselves [15].

IX. Reaction to Bandwidth Changes

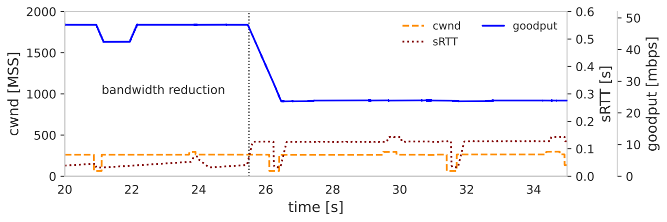
Our last in-depth analysis of ROCCET compared to BBRv3 and CUBIC is the reaction to bandwidth reductions. For this, we have performed greedy TCP throughput measurements for 35 s with a link capacity reduction to half of the original link capacity. In our scenario, we choose a link characteristic of *50 Mbps* × *30 ms* with an oversized buffer of multiple BDP, which are usually found in cellular networks. The characteristic transmissions for BBRv3, CUBIC, and ROCCET are shown in Fig. 9. For CUBIC, we have doubled the scaling of the axis for cwnd and nearly tripled the axis for the sRTT. First of all, for CUBIC, we observe the unwanted side effect of the Linux kernel implementation, which we described in Sec. III,i.e., CUBIC does not react to changes in bandwidth. Instead, the goodput gets halved and the sRTT increases. This behavior gets solved by ROCCET. As we can see in Fig. 9b, ROCCET generates several CEs after the bandwidth of the link is halved. In addition, ROCCET also generates CEs after the bandwidth reduction while bandwidth probing around 30 and 34 s of the transmission. When we compare the BBRv3 transmission (Fig. 9c) with ROCCET (Fig. 9b), we observe that both algorithms provide a nearly equal-sized cwnd before the bandwidth reduction. After the bandwidth reduction, this behavior changes. BBRv3 does not react to the halved bandwidth, which leads to higher sRTTs compared to ROCCET. Furthermore, BBRv3 does not detect the bandwidth reduction until the end of the transmission, even though it enters the min RTT probing state for two times. Some similar performance issues were also observed by [19]. If the available



(a) CUBIC



(b) ROCCET



(c) BBRv3

Fig. 9: Characteristic reaction of BBRv3, ROCCET, and CUBIC to bandwidth reduction from 50 Mbps to 25 Mbps with a RTT of 40 ms and a queue size of multiple BDP. Note: Different axis scaling for cwnd and sRTT in the transmission plot for CUBIC.

bandwidth gets reduced, BBRv3 has substantial delays in cwnd adoptions.

X. Implementation

In this section, we briefly describe the implementation of TCP ROCCET and all additional software we provide for development and debugging the kernel module. Our implementation can be found in a Git repository [20]. Since we have implemented ROCCET as a pluggable kernel module, there is no need to build a new kernel. This means that TCP ROCCET can be loaded while the operating system is running. Our implementation generally extends the Linux kernel code for TCP CUBIC. Most of our code was added to the TCP *.cong_avoid* function. In addition, we added functions to update internal parameters such as RTT_{min} and *ack_rate*. Our implementation contains three files: *tcp_roccet.c*, containing the extension of CUBIC, *tcp_roccet.h*, which contains the internal struct needed for the algorithm, and *roccet_kprobe.c*, an additional kernel module to provide live data of a ROCCET socket during a transmission. The ROCCET kprobe module works similarly to the already existing in the kernel TCP debugging tools.

Our results were generated using the ROCCET algorithm described in Sec. IV. Our implementation offers the possibility to adjust several of ROCCET's parameters. Making the algorithm less aggressive by detecting an increase in sRTT, leading to less CEs generated by ROCCET. The two parameters are the tolerance of the received ACK count and the upper bound for the srRTT, at which an ROCCET CE is generated. By increasing these parameters, ROCCET gets more defensive if the sRTT increases. Furthermore, there is the possibility to ignore loss as a CE generally, and, in addition, a recalculation of the RTT_{min}. The recalculation works as follows: If the last update of RTT_{min} is more than ten seconds ago, ROCCET calculates a new RTT_{min} by using an EWMA, with RTT_{min} and the current RTT as parameters. For a scenario where RTTs increase slightly over a longer time interval of several seconds, this recalculation of the RTT_{min} causes a steady state of srRTT. This can lead to poorer detection of increasing RTTs. Nevertheless, we provide these configuration options to adjust the behavior of ROCCET if they are needed for special link characteristics, for example, lossy links. Since ROCCET is an extension of CUBIC, all inherited CUBIC parameters can be adjusted.

XI. Conclusion

In this paper, we presented a new TCP congestion control algorithm suited for current cellular 5G NR networks. It extends the Linux kernel default congestion control CUBIC and improves its performance, and additionally solves the unwanted side effects of CUBIC's implementation, which we also discuss in this paper. We have described our algorithm in detail and demonstrated its performance through both stationary and mobile measurements in two different commercial networks. Taking our measurement results into account, ROCCET significantly improves the performance of CUBIC by reducing the latency. In addition, we have shown that ROCCET is able to achieve the same goodput as BBRv3 but with less latency. Furthermore, we have performed bandwidth share experiments comparing ROCCET with CUBIC and BBRv3. We provide an in-depth analysis of multi- and single-flow bandwidth share, respecting the work from Ware et al. [15], which consequently classifies our congestion control as deployable, since ROCCET's inter-bandwidth share with BBRv3 and CUBIC is at least as good as the intra-bandwidth share of BBRv3 and CUBIC. Afterwards, we have analyzed the reaction of TCP ROCCET to bandwidth changes compared to BBRv3 and CUBIC. Our analysis shows that TCP ROCCET provides a better performance than the compared congestion control. Furthermore, our kernel module offers several configuration options that can be changed. For future work, we will take a look at ROCCET's performance on different kinds of networks and further optimize its behavior. Our implementation of ROCCET can be found here [20] as a kernel module. As a long-term goal, we will add the code to the official Linux repository. Additionally, a debug module is available in our repository, providing insight into ROCCET's internal variables, such as srRTT.

References

[1] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 64–74, 2008.

[2] L. Prause and M. Akselrod, "TCP Congestion Control Performance Issues in Non-Standalone 5G NR Networks," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, 2023.

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, September-October, pp. 20 – 53, 2016.

[4] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 329–342.

[5] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[6] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," Internet Requests for Comments, RFC Editor, RFC 8289, January 2018.

[7] Google, "BBRv3," https://github.com/google/bbr/releases/tag/bbrv3-2025-03-18, 2025.

[8] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *IEEE INFOCOM 2004*, vol. 4, 2004, pp. 2514–2524 vol.4.

[9] S. Ha and I. Rhee, "Taming the elephants: New TCP slow start," *Computer Networks*, vol. 55, no. 9, pp. 2092–2110, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128611000363

[10] E. Atxutegi, F. Liberal, K.-J. Grinnemo, A. Brunstrom, A. Arvidsson, and R. Robert, "TCP behaviour in LTE: Impact of flow start-up and mobility," in *2016 9th IFIP Wireless and Mobile Networking Conference (WMNC)*, 2016, pp. 73–80.

[11] M. Akselrod and M. Fidler, "TCP Congestion Control Performance on a Highway in a Live LTE Network," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020, pp. 1–7.

[12] L. Prause and M. Akselrod, "A Measurement Study on the Application-level Performance of NSA-NR," in *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*, 2022.

[13] Semtech (formerly Sierra Wireless), "EM9293 5G NR Sub-6 GHz Module," https://www.sierrawireless.com/iot-modules/5g-modules/em9293/, 2025, [Online; accessed 30-September-2025].

[14] The Regents of the University of California through Lawrence Berkeley National Laboratory, "iPerf3," https://github.com/esnet/iperf, 2021.

[15] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Beyond Jain's Fairness Index: Setting the Bar For The Deployment of Congestion Control Algorithms," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, ser. HotNets '19. Association for Computing Machinery, 2019, p. 1724. [Online]. Available: https://doi.org/10.1145/3365609.3365855

[16] The University of Utah, "Emulab," https://www.emulab.net, 2025.

[17] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proceedings of the Conference on Communications Architectures, Protocols and Applications*, ser. SIGCOMM '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 2435.

[18] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, p. 3338, Oct. 2014. [Online]. Available: https://doi.org/10.1145/2677046.2677053

[19] D. Zeynali, E. N. Weyulu, S. Fathalli, B. Chandrasekaran, and A. Feldmann, "Promises and Potential of BBRv3," in *Passive and Active Measurement: 25th International Conference, PAM 2024, Virtual Event, March 1113, 2024, Proceedings, Part II.* Berlin, Heidelberg: Springer-Verlag, 2024, p. 249272. [Online]. Available: https://doi.org/10.1007/978-3-031-56252-5_12

[20] L. Prause and T. Füchsel, "TCP ROCCET Git Repository," https://gitlab.uni-hannover.de/lukas.prause/tcp-roccet-kernel-module/, 2025, [Online; accessed 30-September-2025].