Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 13 (2021) 114–124

DOI: 10.2478/auseme-2021-0009

Comparison of TCP SIAD and TCP BBR Congestion Control in Simulated 5G Networks

Donát Scharnitzky[1], Zsolt Krämer[2], Sándor Molnár[3]

Budapest University of Technology and Economics,
Faculty of Electrical Engineering and Informatics,
Department of Telecommunications and Media Informatics, Budapest, Hungary
e-mail: [1] scharnitzky@tmit.bme.hu, [2] kramer@tmit.bme.hu, [3]molnar@tmit.bme.hu

Manuscript received November 19, 2021; revised December 06, 2021

Abstract: 5G cellular networks have introduced a completely novel air interface called New Radio (NR). This technology delivers numerous benefits compared to previous generations, including significantly higher peak data rates. However, due to the propagation properties of the frequencies used in NR, the volatility of the available downlink capacity also increases. In this paper, we study two TCP congestion control algorithms which are designed to be able to quickly utilize sudden increases in available capacity. We present an implementation of TCP SIAD in the ns-3 open source network simulator and compare its performance with TCP BBR using the mmWave module of the simulator.

Keywords: TCP BBR, TCP SIAD, mmWave, 5G, ns-3

1. Introduction

The commercial deployment of 5G mobile networks is still ongoing, however, the benefits provided by the new air interface are already well established. New Radio (NR) is able to provide lower delay, better energy efficiency, and higher data rates compared to previous generations. Meanwhile, transport layer protocols and mechanisms also continue to evolve. Congestion control is a classic problem in transport layer performance optimization, and Cardwell et al. proposed a new algorithm in 2017 called Bottleneck Bandwidth and RTT (BBR) [1], which promised significant improvements. BBR has been compared to the current default congestion control algorithm in the Linux kernel, CUBIC in a number of recent studies.

In this paper, we study the interplay between congestion control algorithms and 5G mmWave environments. We present the implementation of TCP SIAD

(Scalable Increase, Adaptive Decrease), an algorithm with similar design goals as BBR in the ns-3 open source network simulator. Then we present the first performance comparison between the two algorithms in a scenario involving transitions between line of sight (LOS) and non-line-of-sight (NLOS) states.

The paper is organized as follows. In Section 2 we discuss the related work (namely TCP BBR, TCP SIAD and transport layer performance in 5G mmWave networks), then in Section 3 we describe the implementation of TCP SIAD in ns-3. Section 4 presents the simulation environment and the performance results, and Section 5 concludes the paper.

2. Related work

A. TCP BBR

BBR was introduced by Google in 2017 in [1]. Compared to loss-based and delay-based congestion control algorithms, it represents a new approach, which can be called model-based, where the algorithm tries to maintain an estimation of the bottleneck bandwidth and RTT by active probing. The probing of bandwidth and RTT happens in separate phases, as probing for bandwidth increases latency and probing for RTT drains the queue, thus decreasing throughput. The other two phases of BBR's operation are called startup - and drain phases. The startup behavior is similar to the slow start in CUBIC. The drain phase after the startup tries to drain the additional queue at the bottleneck before the probing for bandwidth can begin.

A formal analysis and measurement study has been presented in [2] on the performance of BBR, which confirmed the intended behavior, however, also identified some cases where excess packet loss and fairness problems may occur. The authors of [3] compared the performance of BBR and CUBIC in real LTE networks on the highway, finding that while the achieved throughput was similar, BBR operated at significantly lower latency.

B. TCP SIAD

TCP Scalable Increase Adaptive Decrease [4] is a congestion control algorithm proposed by Kühlewind with the aim of having low delay while maintaning high utilization in different network conditions. The rate of feedback is independant of the bandwidth and can be manually set via a control point. To achieve this, the increase rate of the congestion window at congestion events is calculated in a way that the next congestion event is expected to happen after the same time as the previous one. This time is measured in the number of RTTs, and this is the configurable control point. The *epoch* is the time between two congestion events (congestion events in the same RTT are

regarded as the same). The buffers in the network have an impact on the delay, since congestions are needed for high utilization and they require the buffers to fill. TCP SIAD tries to empty the buffer every RTT when the delay is too large, which it can achive with reducing the congestion window. After the reduction the increase rate is recalculated to keep high utilization and congestion to happen for feedback (after the expected time since the last congestion event). The algorithm adapts to network environment changes, it detect new bandwidth with a similar method as Slow Start.

ISPs in a lot of cases set the buffer sizes to high values to reach high utilization [4], which leads to high delays and standing queues in case of packet loss based congestion control algorithms because the fillment of buffers. TCP SIAD is packet-loss based, but it also takes into account the delay, it avoids standing queues and reaches high utilizaton with small buffer sizes as well, thus the ISPs can lower the buffer sizes. The configurable control point allows external entities to control how aggressively the algorithm behaves, which makes it possible to apply higher level flow control mechanisms.

TCP SIAD consist of two main parts [4]: Scalable Increase and Adaptive Decrease. Scalable Increase calculates *alpha*, the rate of increase, in every *epoch* such that the *epoch*s will have the same length, but it does not modify *alpha* in an *epoch*. Adaptive Decrease calculates *beta*, the rate of decrease, based on the estimation of the number of packets in queue. TCP SIAD contains three additional algorithms: Fast Increase (to allocate new bandwidth), Additional Decrease (to empty buffers) and Trend Calculation (to improve convergence).

C. Transport layer performance in mmWave environments

The potential and challenges that 5G mmWave networks bring to the transport layer have motivated studies on the interplay between transport layer mechanisms and the characteristics of NR. A list of possible challenges are identified by the authors of [5], including rate adaptation, link quality judgement, bufferbloat, and beam misalignment. A more detailed analysis in [6] studies two deployment scenarios specified by 3GPP: high speed train and dense urban. The authors consider numerous factors that influence transport layer performance and thus different congestion control algorithms, TCP segment sizes, RLC (Radio Link Control) buffer sizes and server locations are compared. Regarding the server location, it is argued that the volatility of 5G networks increase the benefits of a shorter control loop for TCP. This is analyzed in detail in [7], presenting a comprehensive performance evaluation of transparent performance enhancing proxies in 5G mmWave networks. Regarding the different end-to-end congestion control algorithms, [6] finds that BBR significantly outperforms CUBIC, NewReno and HighSpeed in terms of

goodput, especially in the case of smaller RLC buffers. This holds true in both the remote server and the edge deployment cases.

One particularly interesting case in a NR environment is the one involving transitions between LOS and NLOS states. The performance of different TCP variants have been studied in great detail under these conditions. [8] shows that many TCP congestion control algorithms struggle to recover after the NLOS state, especially if the RTT is higher than 5 ms. The more aggressive algorithms (e.g., Scalable) achieved significantly higher throughput compared to CUBIC and Reno in these cases. [9] also investigates LOS-NLOS transitions in an urban deployment, and finds that in most configurations, BBR is able to outperform other TCP algorithms in terms of throughput and latency, and it was the only variant that could benefit from small buffers. In [10] the authors argue that for a 28GHz mmWave deployment, a 7MB RLC buffer results in optimal transport layer performance. A detailed investigation showed the effects of blockage on the different TCP variants, where BBR achieved lower latency than the loss-based algorithms.

All the aforementioned studies used the mmWave module of the ns-3 simulator, and assumed a 28 GHz frequency. An emulation-based measurement study presented in [11] assumed a 60 GHz frequency. Short blockages and long blockages were both studied and BBR avoided the large latency spikes experienced by CUBIC in both cases.

3. Implementing TCP SIAD in ns3

Network Simulator 3 (ns-3) is an open source discrete time network simulation tool [12]. We used an extended version of ns-3, which contains a module for mmWave (5G) that can be used to add mmWave EPC, User Equipment, buildings, etc., to scenarios [13]. We used version 2.0 of the mmWave module, with the BBR implementation (and Internet module) from ns-3.35. Our implementation of TCP SIAD and the complete simulation setup can be found in a public repository [14].

A. Class hierarchy

TCP SIAD is implemented in the Linux kernel [15], we used it to guide our own implementation in ns-3. To add a congestion control algorithm to ns-3, one has to subclass the TcpCongestionOps class. Similarly how most of the congestion control algorithms use TcpNewReno as base class (which is a specialization of TcpCongestionOps) [16] we subclass TcpNewReno as can be seen in *Fig 1*. We added the new TCP SIAD class to the TypeId system for ease of access from scenarios.

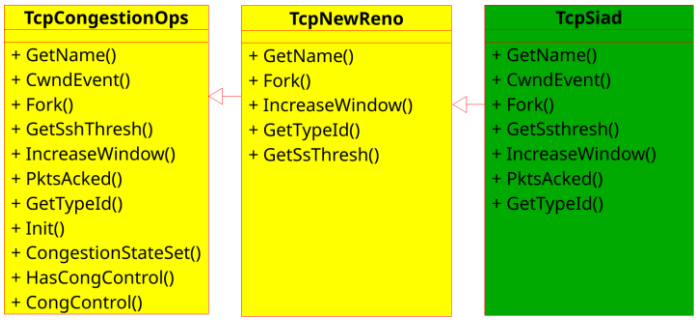

Figure 1: Class inheritance of congestion control, with the added TcpSiad class (green background)

B. Implementation details

We added a header and a source file to ns3 (these contain the SIAD implementation), other changes were not required. The algorithm is realized with four functions:

- GetSsThresh: calculates the new congestion window on a congestion event.
- IncreaseWindow: approximately called on ACKs, increases the congestion window.
- CwndEvent: resets the delay-related variables.
- PktsAcked: called after every ACK, determines the delay.

The variables are stored as class member variables, since the socket object that the functions get doesn't contain as much information as the Linux kernel's socket struct. *config_num_rtt* and *configNumMs* are registered by *TypeId* function, these variables provide the control loopback of TCP SIAD. The following two subchapters describe some implementation details.

C. Scalable Increase

The connection starts in Slow Start. If the congestion window is larger than *incthresh*, it needs to be grown via Fast Increase (Formula 1.) starting from the value 1 [4]:

$$alpha = alpha + \frac{alpha}{cwnd}, every\ ACK, alpha \leq \frac{cwnd}{2} \tag{1}$$

If it is lesser than *ssthresh*, then via Slow Start. In these cases, there is no Additional Decrease, and this is indicated by setting *min_delay_seen* to true.

Additional Decrease can happen only after Adaptive Decrease or Additional Decrease (with approximately one RTT offset) in the linear increase phase.

If Additional Decrease is not needed, the congestion window is increased. *snd_cwnd_cnt* counts the segments, the counter is set to 0 when an increase happens, or when decrease happens (somewhere else).

alpha needs to be modified in the following four cases:

- We just entered linear increase phase after Slow Start (before the increase the congestion window is lesser than *ssthresh*, after that it is greater). *alpha* needs to be calculated according to Formula 2. [4]:

$$alpha = \frac{incthresh - ssthresh}{Num_{RTT}}, 1 < alpha < ssthresh \tag{2}$$

- If we just left Slow Start, but there is no valid *incthresh* yet (*incthresh* < *sshthresh*) or we just left *incthresh* we switch to Fast Increase, thus *alpha* needs to be set to 1.
- We are in Fast Increase (congestion window is greater than *incthresh*). We add inc to *alpha* if it is less than half of the congestion window.
- We are in Slow Start (congestion window is lesser than *ssthresh*). *alpha* will always be the congestion window, with which we achieve to double the window.

D. Additional Decrease

Additional Decrease: *snd_cwnd_cnt* is set to 0 (see below). The congestion window is reduced according to Formula 3. [4]:

$$cwnd = \frac{RTT_{min}}{RTT_{curr}} ssthersh - 1 \tag{3}$$

Then, if the congestion window is greater than the minimum congestion window, *alpha_new* (Formula 4.) and *red* (Formula 5.) is calculated [4]:

$$alpha_{new} = \frac{incthresh - cwnd}{Num_{RTT} - cnt_{dec} - 1} \tag{4}$$

$$red = \frac{cwnd}{Num_{RTT} - cnt_{dec}} \tag{5}$$

If *red* is greater than *alpha_new*, *alpha* is recalculated and the congestion window is reduced by *red*. If *red* is not greater that *alpha_new*, then *alpha* is set to *alpha_new* and is substracted from the congestion window. If the congestion window was the minimum congestion window, *alpha* has to be calculated

again. If at any point the congestion window becames minimal, *min_delay_seen* is set to true to disable the running of more Additional Decrease. If *alpha* is greater than the congestion window, then no more Additional Decreases happen (but *alpha* is not restricted, otherwise *incthresh* would not be reached in time).

4. Performance evaluation

A. Simulation environment

We created a scenario to test how TCP BBR and TCP SIAD behave in a volatile environment. It contains an EPC network which is connected to a remote host, this simulates a connection via the Internet. The topology has one eNodeB that is connected to one User Equipment (UE). The UE has a constant linear movement, after 5 seconds a building blocking its LOS to the eNodeB and then after another 5 seconds it goes back in LOS again.

We measured the received bytes and the RTT. *Table 1* shows the parameters that we changed from the default values. We compared TCP BBR and TCP SIAD congestion control algorithms in the same environment, where the environment was constant except for the RLC buffer size (and a random seed parameter).

Table 1: ns3 configuration parameters

Parameter	**Value**
Internet link RTT	25 ms
Internet link bandwidth	100 Gbps
Internet link MTU	1500 byte
TcpSocket segment size	10000 byte
TcpSocket min RTO	1000 ms
TcpSocket send buffer size	131072 * 50 byte
TcpSocket receive buffer size	131072 * 50 byte
AQM	Disabled
HARQ	Enabled
Center Frequency	28 GHz
Path loss model	BuildingsObstaclePropagationLossModel
Scheduler type	MmWaveFlexTtiMacScheduler
Simulation time	15 s
RLC mode	AM
RCL buffer size	[4, 7, 20, 40] MB

B. Performance results

Table 2 shows the Average Throughputs and RTTs for the different RLC buffer sizes and algorithms. At high buffers (7MB or greater), the congestion control algorithms behave virtually the same (they have a difference of less than 1%), thus having the same throughput and delay, which concludes that using more buffer than 7MB has no effect. *Fig 2* shows the utilization and fast adaptation of both algorithms in this case. Lowering the buffer however can negatively impact the throughput, especially in case of TCP BBR. TCP SIAD could still keep 90% of its utilization, and with this it achieved a 41% higher throughput than TCP BBR, while having the same delay.

At 4MB buffer the delay was 37% smaller compared to at 7MB buffers, which can justify using smaller buffers for delay sensitive applications.

Table 2: Average throughput and RTT for the different RLC buffer sizes

RLC buffer	**Average Throughput (Mbps)**		**Average RTT (ms)**	
	SIAD	**BBR**	**SIAD**	**BBR**
4 MB	1157.87	819.84	28.84	28.69
7 MB	1289.20	1295.10	46.03	45.81
20 MB	1290.57	1301.14	46.05	45.76
40 MB	1290.57	1301.14	46.05	45.76

When the UE moves behind the building, the throughput decreases and the delay increases significantly. After going in LOS again with the eNodeB, in case of higher buffer values, the throughput and delay get back to the previous values, as can be seen in *Fig 2*. In case of 4MB RLC buffer, after going back in LOS, TCP SIAD is able to utilize the bandwidth as well as before going in NLOS, however TCP BBR struggles to do so, as depicted in *Fig 3*.

It is worth highlighting that we have used an increased TCP MSS of 10KB based on previous studies [6] that showed how using larger segments can enable TCP to achieve higher throughput in mmWave environments. BBR however does not benefit from this due to its fundamentally different design.

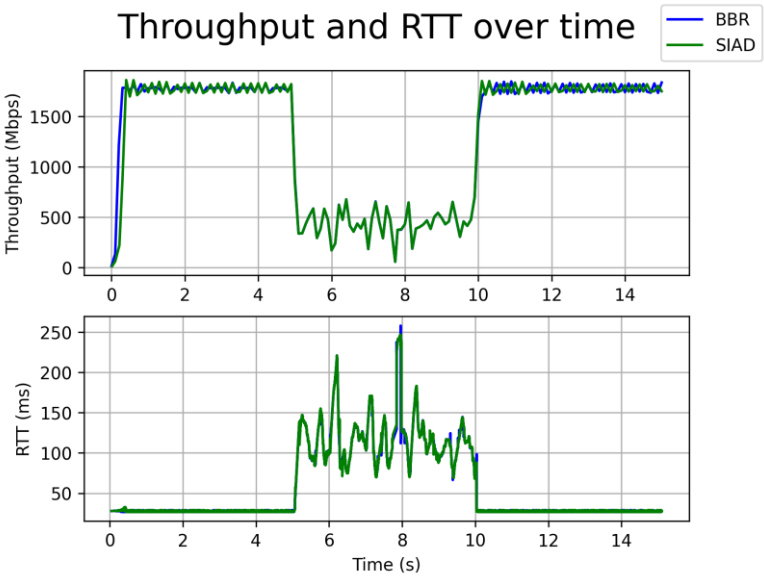

Figure 2: Throughput and delay of TCP BBR and TCP SIAD at 7MB RLC buffer

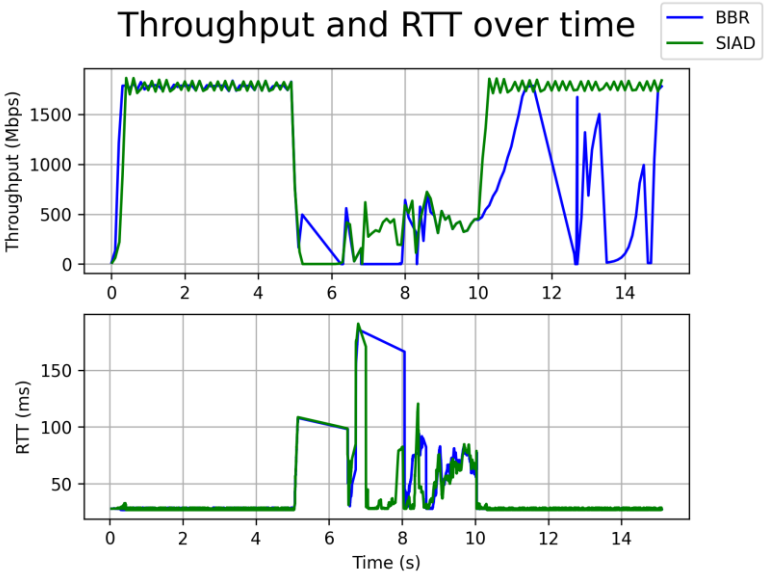

Figure 3: Throughput and delay of TCP BBR and TCP SIAD at 4MB RLC buffer

5. Conclusion

The interplay between transport layer mechanisms and 5G mmWave network dynamics are complex and provide opportunities for optimizations. In this paper we have implemented TCP SIAD, an algorithm designed to quickly adapt to increased available bandwidth while keeping the delay low, and compared it to TCP BBR in a 5G mmWave environment using ns-3. We have shown that both algorithms perform well in terms of utilization and adaptation for RLC buffer sizes 7MB and higher, but at 4MB they start to struggle (especially TCP BBR). For low latency, however, 4MB buffer with TCP SIAD is a better choice, since it has only a small reduction in link utilization.

Our future work includes extending this analysis to different segment sizes and internet link RTTs.

References

[1] Cardwell, N., Cheng, Y., Stephen Gunn, C., Yeganeh, S. H., and Jacobson, V., "BBR: Congestion-Based Congestion Control", *Communications of the ACM* 60.2, 2017, pp. 58–66.

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

[3] Feng, L., Chung, J. W., Jiang, X., and Claypool, M., "TCP CUBIC Versus BBR on the Highway", *International Conference on Passive and Active Network Measurement. Springer, Cham*, 2018.

[4] Kühlewind, M., "TCP SIAD: Congestion Control Supporting High Speed and Low Latency", arXiv preprint arXiv:1612.07947, 2016.

[5] Ren, Y., Yang, W., Zhou, X., Chen, H., Liu, B., "A Survey on TCP Over mmWave.", *Computer Communications*, 2021.

[6] Zhang, M., Polese, M., Mezzavilla, M., Zhu, J., Rangan, S., Panwar, S., Zorzi, M., "Will TCP Work in mmWave 5G Cellular Networks?", *IEEE Communications Magazine* 57.1, pp. 65–71, 2019.

[7] Hayes, D. A., Ros, D., and Alay, Ö., "On the Importance of TCP Splitting Proxies for Future 5G mmWave Communications", *IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium, IEEE*, 2019.

[8] Pieska, M., and Kassler, A., "TCP Performance over 5G mmWave Links—Tradeoff Between Capacity and Latency", *IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE*, 2017.

[9] Poorzare, R., and Calveras Augé, A., "How Sufficient is TCP When Deployed in 5G mmWave Networks Over the Urban Deployment?", *IEEE Access 9*, 2021, 36342–36355.

[10] Mateo, P. J., Fiandrino, C., and Widmer, J., "Analysis of TCP Performance in 5G mm-wave Mobile Networks.", *ICC IEEE International Conference on Communications (ICC). IEEE*, 2019.

[11] Srivastava, A., Fund, F., and Panwar, S. S., "An Experimental Evaluation of Low Latency Congestion Control for mmwave Links", *IEEE INFOCOM Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE*, 2020.

[12] https://www.nsnam.org/ [Online, last checked: 2021.11.18]

[13] Mezzavilla, M., Zhang, M., Polese, M., Ford, R., Dutta, S., Rangan, S., Zorzi, M., "End-to-End Simulation of 5G mmWave Networks," in *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2237–2263, 2018.
[14] https://github.com/dscharnitzky/mmwave-siad-bbr [Online, last checked: 2021.11.18]
[15] http://mirja.kuehlewind.net/src/tcp_siad.c [Online, last checked: 2021.11.18]
[16] https://www.nsnam.org/docs/release/3.35/doxygen/classns3_1_1_tcp_congestion_ops.html [Online, last checked: 2021.11.18]