Online Monitoring of TCP Throughput Limitations

Simon Bauer, Kilian Holzinger, Benedikt Jaeger, Paul Emmerich, Georg Carle
Technical University of Munich (TUM), Department of Informatics
Chair for Network Architectures and Services
{bauersi | holzingk | jaeger | emmericp | carle }@net.in.tum.de

***Abstract*—Monitoring and optimizing network performance is essential for data center operators and service providers. To better understand network performance, the throughput limiting factor of TCP connections can be analyzed. Previous research introduces approaches for the offline analysis of root causes of TCP throughput based on previously captured traffic. With increasing computational power and packet processing capabilities on commodity hardware, previously existing analysis limitations get overcome nowadays.**

This paper presents a scalable tool to analyze the throughput limitation of TCP flows in real-time, i.e., while the tool observes traffic on a network interface. We describe the adaption of existing approaches for TCP throughput limitation analysis and required passive capacity estimation to satisfy online analysis requirements.

We evaluate the effectiveness and accuracy of our implementation with a generated data set for different TCP congestion control algorithms and varying network parameters. Furthermore, we survey the performance of our implementation with thousands of concurrent flows and discuss trade-offs and limitations of such sophisticated analysis in real-time.

We provide our code as free and open-source.

***Index Terms*—Traffic monitoring, TCP, throughput, root cause analysis**

I. Introduction

Providing reliable and fast data transmission is a primary interest of service providers and data center operators. The performance of network traffic often relates to the performance of the Layer 4 protocol TCP, which provides reliable data transmission to higher layers and their applications. When talking about TCP performance, throughput, also referred to as data rate, is of primary interest.

As higher throughput indicates faster data transmission and therefore influences user experience, it is essential to understand the causes behind achieved data rates. Understanding and analyzing such causes allows optimizations of network infrastructure, operating systems, and applications and helps to detect network incidents or anomalies.

Especially the latter one motivates to perform the analysis of throughput limitations online, i.e., during traffic is observed. The causes behind TCP throughput were studied in previous research and referred to as TCP root cause analysis (RCA). Siekkinen et al. [1] introduce a TCP RCA toolkit that estimates the reason behind a network limited data transfer. While such toolkit relies on full traffic captures of network traffic, it is limited to offline analysis. This paper closes the gap between sophisticated offline analysis of TCP characteristics and online monitoring of such characteristics.

978-1-7281-4973-820$31.00

We contribute a scalable tool to run TCP RCA in online. The tool is capable of analyzing hundreds of Mbit/s on commodity off-the-shelf (COTS) hardware. We describe modifications of the TCP RCA algorithm presented by Siekkinen et al. [1], [2] concerning online analysis. Furthermore, we survey design decisions and trade-offs of our implementation and evaluate the effectiveness, accuracy, and performance of our implementation with a generated data set. Our tool is available as free and open-source.

The remainder of this paper covers the following: First, we provide background knowledge regarding TCP throughput and previous research on the before-mentioned TCP RCA toolkit. Next, we describe the design and implementation of our tool. Further, we describe our test setup and data set generation before evaluating our tool. We conclude with determined trade-offs of our approach.

II. Background

In this section, we provide background knowledge regarding TCP throughput, TCP congestion control algorithms, the potential limitations of TCP throughput, and the analysis of such limitations.

A. TCP Throughput and Congestion Control

The transmission control protocol (TCP), as originally specified in RFC 793 [3], provides reliable data transfer in unreliable network environments. TCP takes care of retransmitting data in case of packet loss or packet damage. Packet loss is detected by the absence of an expected acknowledgment packet or by repeating acknowledgments for previous packets.

TCP uses a window field to enable the receiving endpoint to limit the sending rate of the sender, if necessary. The sender is not allowed to send more unacknowledged data than the size of the receiver advertised window. As the receiver window field is limited to 65535 B by its length of 16 bit, a window scaling option can be used to increase the receiver window up to 1 GB [4].

TCP congestion control is purposed to optimize the sending rate on the sender side. I.e., the sending rate

should converge to the available bandwidth of the bottleneck link of the path of a connection. Different algorithms exist to determine an optimal congestion window size as close as possible. While algorithms differ significantly in details, congestion control is typically based on different phases which define the increase or decrease of the congestion window. Indicators for these increases or decreases are connection characteristics like packet loss or the round trip time (RTT). We consider the congestion algorithms Reno [5], CUBIC [6], and BBR [7] as the most wide-spread and relevant approaches.

B. TCP Throughput Limitation Analysis

While TCP tries to maximize a connection's throughput up to the available bandwidth, different factors can limit throughput. Siekkinen et al. [1], [2] surveyed TCP throughput and its limitations in details.

Authors differ between two types of limitations: application limitation and network limitation. Next to application limited periods (ALPs), Siekkinnen et al. introduce bulk transfer periods (BTPs) and short transfer periods (STPs). STPs are defined as transfer periods with less than 130 packets which are not application-limited. To determine the network limiting factor of a BTP, authors introduce a decision tree. The decision tree differs between four different root causes: unshared and shared bottleneck links, the advertised receiver window, and the transport layer itself.

Unshared and shared bottlenecks imply that the throughput of a connection cannot be increased due to the limited capacity of a network link. This bottleneck link is either fully utilized by a single flow (unshared bottleneck) or the bottleneck link is shared by several concurrent flows that fully saturate the link (shared bottleneck). A flow is referred to as receiver limited, if the receiver advertised window prevents the sender to send out more packets while no other limitation prevents the increase of throughput. The transport layer limitation relates to TCP's congestion avoidance (CA). CA may limits throughput when the sender cannot fully exploit the receiver advertised window, while no limitation by the network path occurs. Figure 1 shows an illustration of the decision tree introduced by Siekkinen et al. [1]. So-called limitation scores indicate the described root causes.

The scores used to estimate a root cause are deducted as follows: The dispersion score indicates whether an unshared bottleneck limits a flow. Such indication is done by analyzing the ratio between achieved throughput and maximum physical bandwidth on the whole path. Hereafter, we refer to the maximum physical bandwidth of a network path as the capacity of the path.

The retransmission score compares the amount of retransmitted data to the amount of total transmitted data. A high frequency of retransmitted data indicates an overloaded shared bottleneck link. As packets get dropped

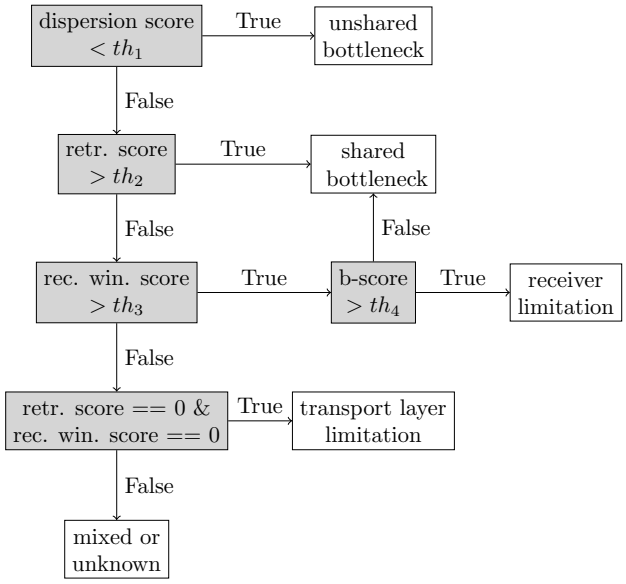

Fig. 1. Decision tree for TCP throughput limitations according to Siekkinen et al. [1].

at the bottleneck due to overload and therefore require retransmissions.

The receiver advertised window score analyzes whether the receiver advertised window allows the sender to send more data without acknowledgments or not. In the second case, the receiver advertised window prevents higher throughput.

The burstiness score or b-score, is purposed to filter out connections, that are limited by a shared bottleneck but do not suffer from packet loss. Shared bottleneck links with large buffers can cause such observation. Large buffers avoid packet drops and therefore avoid retransmissions.

Siekkinen et al. limit the scope of their RCA toolkit to long-lived TCP connections, e.g., connections that transmit around 100 KB - 150 KB. An additional requirement of the tool is FIFO queuing at all passed routers, as required for the used passive capacity estimation approach. The passive capacity estimation of the original toolkit is based on the PPrate algorithm introduced by En-Najjary et al. [8] which works on packet dispersion.

III. Design and Architecture for Online Analysis

To implement our online throughput limitation analysis tool, we build on the network capture tool FlowScope due to its efficient packet processing capabilities. FlowScope uses the Data Plane Development Kit (DPDK) for packet I/O and is highly scalable. Benchmarks show FlowScope's capabilities to process over a million flows concurrently, and to achieve high analysis rates, up to 100 Gbit/s [9], [10].

FlowScope uses a hashtable indexed with `flowkey`s to keep track of flows and to store flow specific information. Dynamically re-sizing of hashtable entries is too expensive

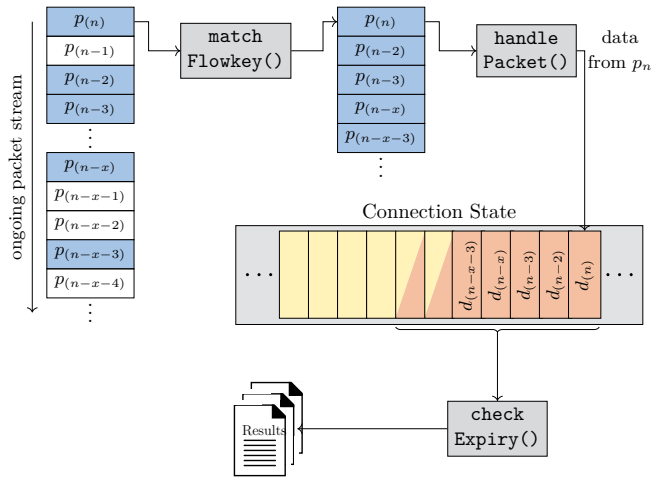

Fig. 2. Illustration of processing steps and connection state handling.

for online analysis. Therefore, we decide to use static hashtable entry sizes. While this avoids expensive resizing, the state possibly stored for each flow gets limited.

As we require both directions of a flow for our analysis, we rely on bidirectional flow tracking. We do this based on the `flowkey`. We calculate the `flowkey` as a hash of a sorted connection identifier, based on the source and destination addresses, respectively ports. To keep the association of the sender, respectively the receiver, to the corresponding IP address, we store such information in the connection state of each flow.

As mentioned above, we decide to use a fixed connection state size. Therefore, we have to choose a sufficiently large hashtable entry size to provide enough space for different data collected over time, that is required for the analysis. We choose a connection state size of 156 KB. Tests show that this size is sufficiently large, while memory requirements are not too extensive. I.e., a connection state size of 156 KB implies an approximated memory need of 160 GB for 1 million concurrent flows. This relation seems reasonable considering modern commodity-off-the-shelf (COTS) hardware.

We split analysis tasks in different modules to provide flexibility and maintainability. In order to achieve better performance, we decide to separate the purpose of each module in two steps: First, we extract features of newly arrived packets and store them in the connection state. Feature extraction is done by a `handlePacket()` function each module provides. Second, a periodically called function calculates the final results of a particular module based on the previously stored data. This step is implemented in the `checkExpiry()` function of each module. In the following, we refer to a call of this function as the periodical calculation step. As a default, we set the interval between calculation steps to one second.

We orchestrate and call the different modules in the main module. I.e., the main module invokes packet handling by the other modules, runs the calculation steps, and collects return values. The main module enables to configure the execution order of the analysis modules and the frequency of their calculations. The different modules and their purpose are described in details in Section IV.

A further design decision is the use of the available connection state. In contrast to the original offline implementation of the TCP RCA toolkit, we only have limited capabilities to store data per analyzed flow. We use the connection state as a ring buffer to handle overrunning connection states. The ring buffer enables us to maintain a sliding window of currently considered data that was previously extracted per packet.

Figure 2 shows the mentioned processing steps and their interplay with the connection state. As illustrated, data in the connection state can be used several times by different calculation steps, depending on the rate of arriving packets.

IV. Implementation of Analysis Modules

In this section, we describe the implementation and functionality of the single analysis modules. In total, the tool consists of 10 different modules, including the aforementioned main module.

a) Handling Connection Establishments: We assume that the first packet of a connection is a TCP SYN packet from the client to the server. If the first observed packet of a connection is not an `SYN` packet, we ignore the flow for further analysis. During the TCP Three-Way Handshake, the module extracts the maximum segment size (MSS) and the TCP window scaling options for both directions.

b) Position Estimation: To estimate the measurement point's position on the network path of a connection, we observe time intervals during the Three-Way Handshake. I.e., the module analyses the relation of the time interval between the initial `SYN` packet and the corresponding `SYN + ACK` packet and the time interval between the `SYN + ACK` packet and the following `ACK` packet. These time intervals are referred to as δt_2 respectively δt_1 in Figure 3. We calculate the measurement position by $\frac{\delta t_2}{\delta t_1}$

The position estimation is purposed to decide whether it is necessary to shift certain packet timestamps or not. Shifting might be required for the time-series of receiver advertised windows and outstanding bytes. Shifting is not applied if we measure traffic near the sender, in our terminology the server, but for receiver-side measurements. Based on δt_2 and δt_1 the shifting value is determined by $\frac{\delta t_2}{\delta t_1} \cdot \frac{RTT}{2}$

c) ALP Detection: If a packet belongs to an ALP, it is ignored for further network limitation analysis. We detect ALPs according to the approach proposed by Siekkinen et al. [2]. I.e., we determine the current period as application limited if we observe an idle time larger than $\frac{RTT}{2}$ or a too-high share of packets smaller than the MSS. An ALP ends, if we observe three consecutive packets with a size equal to the MSS. Our tool keeps track of ALPs by labeling

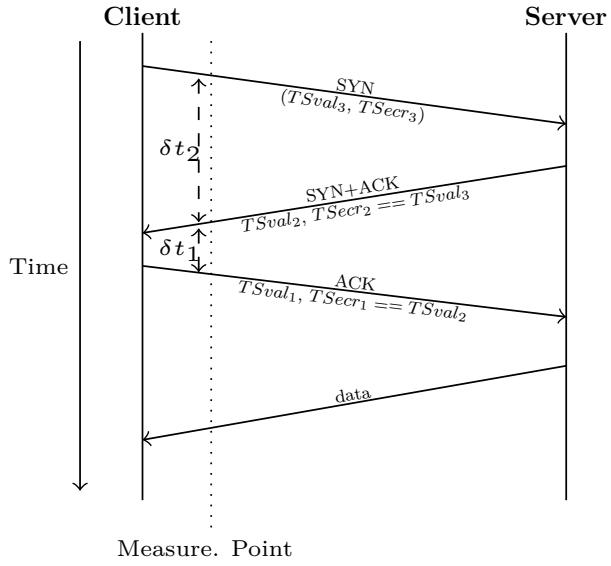

Fig. 3. Position and RTT estimation during the Three-Way Handshake.

a packet received during an ALP and calculates the share of ALPs of the whole flow. While isolating ALPs from BTPs is feasible, merging of BTPs separated by small ALPs as described by Siekkinen et al. [2] cannot be done online. Merging such periods requires the whole history of a flow, a requirement we cannot guarantee due to limited connection state.

d) RTT Estimation: We estimate the current RTT with the TCP timestamp option. TCP timestamps are widely spread nowadays and allow to estimate RTTs straightforward. The RTT estimation module stores the TCP timestamp value (`TSval`) and TCP timestamp echo reply value (`TSecr`) for each packet. During the periodical calculations, the estimation module takes the latest packet whose `TSval` value can be matched to the `TSecr` value of an already observed packet. The time interval between these both packets is referred to as δt_1, as shown in Figure 3. Next, the tool takes the `TSecr` value of the first packet and matches it to the packet with the corresponding `TSval` value. We refer to this time interval as δt_2 and estimate the RTT as $\delta t_1 + \delta t_2$.

e) Capacity Estimation: The capacity estimation module is based on the PPrate algorithm published by En-Najjary et al. [8]. The module can estimate the capacity for both directions of the connection based on the stream of data packets or based on the stream of `ACK` packets. The better-suited method depends on the measurement position.

After choosing the estimation method, we calculate capacity samples for each received packet based on packet size and the inter-arrival times of packets. Capacity samples get stored in the connection state.

During the periodical calculations, the module performs statistical analysis of the before calculated capacity samples.

We filter and process the stored capacity samples according to the PPrate algorithm. The algorithm requires to estimate capacity by the strongest and narrowest mode higher than the so-called asymptotic dispersion rate, which is derived during statistical processing. As the determination of the strongest and narrowest mode is not further defined, we use an own function to find the strongest and narrowest mode. The function determines the correct mode based on mode significance and the count of samples next to the actual mode.

To avoid unreliable estimates, the module only considers flows for which more than a configurable count of packets have been observed. The corresponding sliding window size can be configured. The sliding window size parameter is purposed to limit the possible input size for the further calculations. This is motivated by performance considerations and is not expected to impact accuracy, as the sliding window is configured to consider a sufficient number of packets.

f) Dispersion Score: The dispersion score is calculated by the ratio between the average throughput of a bulk transfer and the capacity of the connection path. The ratio then is subtracted from 1. The average throughput is calculated based on the last n packets of the connection, while the sliding window size n can be configured.

g) Retransmission Score: The purpose of the retransmission score is to detect shared bottlenecks due to a high share of retransmitted data packets. The score calculates the relation between retransmitted data and the transmitted data in total. We identify retransmitted packets by observing sequence numbers. If we observe a sequence number smaller or equal to the highest sequence number observed previously, we classify a packet as a retransmit.

During the periodical calculation step, we sum up the observed data and calculate the final retransmission score.

h) Receiver Window Score: The receiver window score requires time series data of the receiver advertised window and of the number of outstanding bytes.As mentioned above, time-series data might requires to be shifted depending on the position of the measurement point. For each packet we shift timestamps and keep track of the highest seen `ACK` and `SEQ` number as time-series. Based on both time-series we periodically calculate the number of outstanding bytes. If the difference between the receiver advertised window and the number of outstanding bytes is larger than three times the MSS, the module appends a 1 to a boolean vector. Otherwise, the module appends a 0. Finally, the receiver window score is the average of all vector elements in the sliding window.

i) Burstiness Score: In contrast to the original RCA toolkit [1], we decide to use the burstiness score and not the proposed b-score. This decision is reasonable, as our framework always keeps track of the current RTT and capacity. As all necessary per-packet data is already extracted by other modules, we can directly calculate

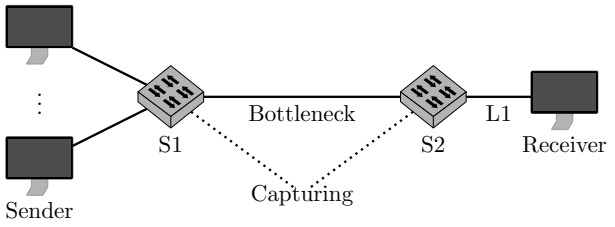

Fig. 4. Setup of the used measurement-framework.

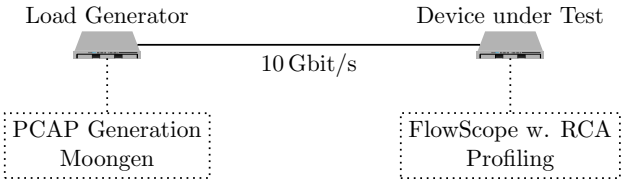

Fig. 5. Setup for measurements in the Testbed.

the burstiness score during the periodic calculations. The needed average of the receiver advertised window size is determined during the periodic step, too.

V. Evaluation

For the evaluation of our tool we consider two aspects: the effectiveness of the implemented modules and overall performance. First, we describe the generation of our test data in Section V-A. Afterwards we evaluate effectiveness in Section V-B and performance in Section V-C

A. Test Setup and Test Data Sets

We generate test data sets with the measurement framework introduced by Jaeger et al. [11] based on the network emulation tool Mininet [12]. The measurement framework allows to configure setups with various parameters, like bandwidth, TCP algorithm, packet delays, or loss rate.

We use a basic topology consisting of several sending nodes, representing the clients, and a receiver node, representing the server. As the framework generates TCP traffic with `iperf3`, the client corresponds to the sender in this setup. Senders are connected to the receiver via two switches that are interconnected with a link. Both switches serve as capturing points. This enables us to capture traffic on both sides of the bottleneck link. Figure 4 illustrates this basic setup.

Generated PCAPs are then replayed with the packet generator MoonGen [13] between two physical hosts. This setup, as illustrated in Figure 5, is similar to a monitoring host that listens on mirror link.

For the evaluation of the accuracy of implemented capacity estimation module, we consider different test cases.

We survey the estimation accuracy for the considered TCP congestion algorithms Reno, CUBIC, and BBR. Additionally, we are interested in the impact of packet loss on the estimation accuracy. Therefore, we generate flows suffering under different probabilities of packet loss at the bottleneck link. To study accuracy for different bottleneck capacities, we generate PCAPs with different bottleneck capacities. To inspect the impact of several concurrent flows sharing the bottleneck link, we produce a set of PCAPs with an increasing number of concurrently established TCP connections.

Table I summarizes all configurations of the measurement framework for different test cases. By default, we configure a capacity of 10 Mbit/s. For each configuration we produce 10 PCAPs.

To evaluate the effectiveness of the implemented limitation scores we generate traffic consisting of a bulk transfer limited by a specific root cause. We vary parameters like congestion control algorithm, and round trip time for each root cause. Bottleneck capacity is set to 10 Mbit/s. In total, we generate 360 test PCAPs for each of the four root causes.

To capture flows limited by an unshared bottleneck, we establish a single TCP connection. The only limiting factor is the limited capacity of the bottleneck link. For shared bottleneck limited traffic, we extend the setup of unshared bottlenecks by an additionally established connection. This way, two flows limit each other on the shared bottleneck link. To reproduce receiver window limited flows, we disable the TCP receiver window scaling option in the Linux kernel of the receiving host before establishing the connection. To capture congestion window limited data transfer, we establish connections that are terminated after 1 second. For such short connections we expect the TCP slow start to mainly limit throughput.

To evaluate performance, we generate PCAPs with varying number of concurrent flows. As the measurement framework is not purposed for high numbers of concurrent flows with high bandwidth, we generate traffic between two physical servers. Several `iperf3` servers are started in parallel and respond to several clients on different ports.

B. Evaluation of Module Effectiveness

This section presents results of measurements of the implemented capacity estimation module and the modules responsible for score calculation.

a) Capacity Estimation: We evaluate the accuracy of the passive capacity estimation for different tests cases as described in Section V-A. To assess accuracy, we define different thresholds for the relative error of a capacity estimate. Measurements show that the results for both methods are quite stable. Results captured on switch S2 are estimated with the data packet-based method. For captures received from switch S1 we apply the `ACK`-based method.

Our analysis shows that for the most test cases less than 50 % of all estimates satisfy an accuracy threshold of 3 %. For an relative error acceptance of 5 % accuracy increases significantly for measurements with different congestion control algorithms and several concurrent flows. We find that there is no significant impact by the TCP algorithm

TABLE I
RESULTS OF THE EVALUATION OF THE CAPACITY ESTIMATION MODULE FOR DIFFERENT ACCURACY THRESHOLDS.

Test Case	Varied Values	Data packet-based Error < 3 %	< 5 %	< 10 %	ACK-based Error < 3 %	< 5 %	< 10 %	Both methods Error < 50 %
Cubic only	-	32 %	96 %	99 %	28 %	98 %	100 %	100 %
TCP Cong. Control	Reno, CUBIC, BBR	66 %	97 %	97 %	73 %	90 %	93 %	100 %
Packet Loss	0 % - 25 %	48 %	73 %	78 %	47 %	71 %	77 %	78 %
Concurrent Flows	1 - 25	37 %	95 %	98 %	36 %	95 %	98 %	98 %
Capacity	5 Mbit/s - 100 Mbit/s	48 %	59 %	70 %	47 %	68 %	81 %	90 %

on the accuracy, while TCP CUBIC results in even higher accuracy.

As expected, the accuracy of estimates decreases with increasing packet loss. Results' correctness drops significantly for probes suffering from loss higher than 2 %. However, such high loss rates are expected to have a heavy impact on network communication and should not occur in real-life networks.

Furthermore, we find that our implementation results in less accurate estimates with increasing bottleneck capacity. While we observe an accuracy near 100 % for bandwidths between 5 Mbit/s and 30 Mbit/s, we only observe an accuracy about 60 % for bandwidths between 5 Mbit/s and 100 Mbit/s with an error tolerance of 5 %. Further analysis and debugging is required to understand the causes of such higher deviations.

Table I shows a summary of the results for all different test cases and different error tolerances. The table shows the share of estimates that satisfy a certain accuracy threshold for all test cases and both estimation methods, i.e., data packet-based and ACK-based.

b) Score Calculation: Regarding the effectiveness of the tool's limitation score modules, we study the cumulative distribution functions (CDF) for each score for the different test cases. We expect that scores allow separating the different root causes. For each test run we get several score values due to several periodical calculations. We calculate the CDF with the averages of all received score values of a single test run.

For the dispersion score, we expect scores of unshared bottlenecks to be significantly smaller compared to others. Figure 6 shows the cumulative distribution function of the dispersion score calculated for all test cases. Nearly 80% of flows limited by an unshared bottleneck result in a dispersion score of less than 0.1, while the scores calculated for other test cases have a significant share of higher dispersion scores. As we calculate the dispersion score by $1 - \frac{throughput}{capacity}$, negative score values are unexpected. Such values can be traced back on underestimated capacities.

After filtering unshared bottlenecks due to the dispersion score, the classification scheme uses the retransmission score to separate limitations by shared bottlenecks from receiver window or transport layer limitations. We expect the retransmission score of receiver window or transport layer limited flows to be significantly smaller, as shared bottlenecks cause retransmissions of packets dropped at the shared bottleneck link. Figure 6 shows the retransmission score of the mentioned test cases. We can differ receiver window limited flows from others, while transport layer-limited flows show similar retransmission scores as shared bottleneck limitations. This observation is caused by our test data. We generate transport layer limited flows by capturing flows during the TCP slow start phase, which also suffers under packet loss and, therefore includes retransmissions.

The receiver window score is purposed to separate flows that tend to be receiver limited from flows that are limited by the transport layer. Figure 6 shows the distribution of calculated receiver window scores for both of the corresponding data sets. The receiver score recognizes receiver window limited flows due to high values. The scores calculated for shared bottleneck and transport layer limited flows are mostly zero.

The RCA classification scheme considers that there are cases of shared bottleneck limitations that show high receiver window scores. This situation might be the case if a packet passes larger buffers that delay the transmission of outstanding bytes. However, as our data set does not consider such a scenario, the receiver window limitation scores for shared bottlenecks are quite small.

The burstiness score is purposed to filter receiver window limitations from shared bottlenecks. CDFs for both limitations differ significantly as shown in Figure 6 and allow separating both cases from each other.

c) Summary: Our implementation of the passive capacity estimation module works sufficiently accurate for most test cases. An artifact to mitigate is the increasing relative error for higher capacities. However, the inaccuracy of the capacity estimation did not affect the evaluation of the implemented limitation scores significantly, as measurements of score modules are done with a capacity of 10 Mbit/s.

The results of the limitation score modules satisfy our expectations regarding the correlations between the labeled root cause of test case and the calculated score values. Except the inaccurate test cases for transport layer limited flows due to retransmissions during the slow start phase, we did not observe unexpected scores. The plotted distributions show the variance of our generated test cases and indicate potential threshold values required to determine the actual throughput limiting factor of a flow.

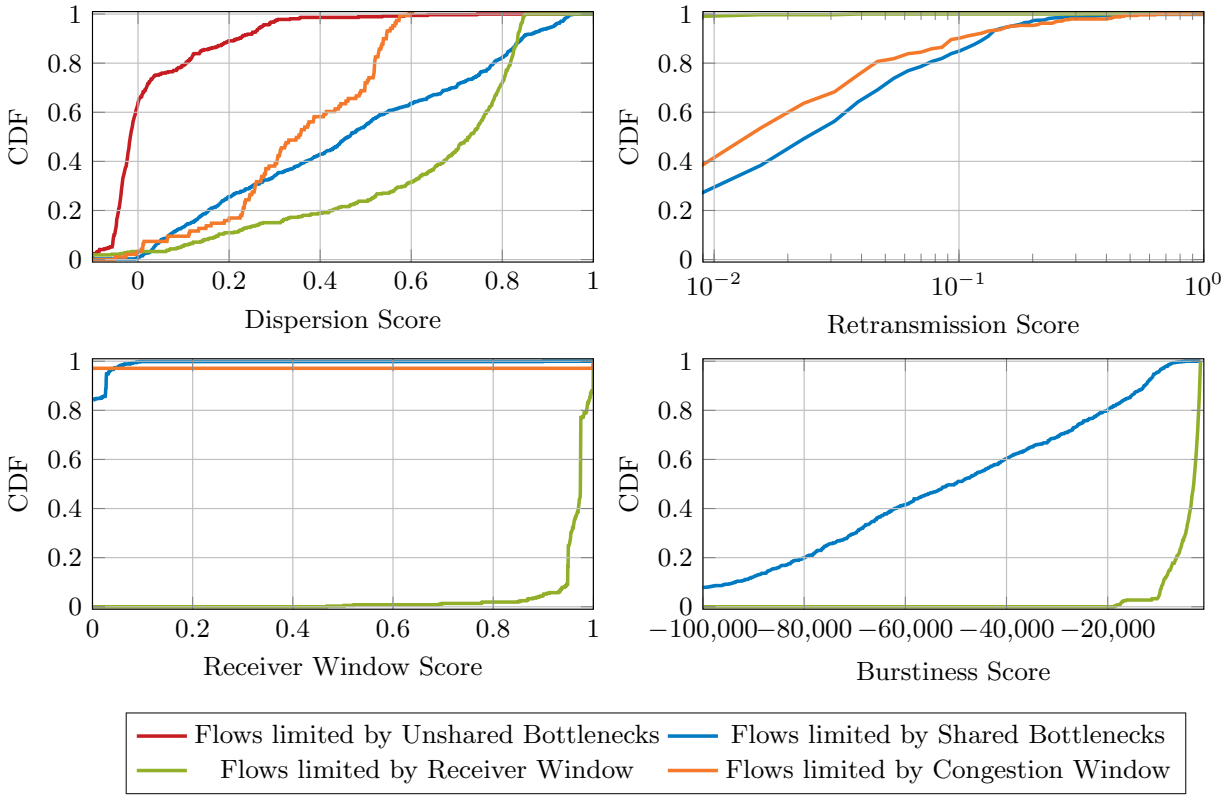

Fig. 6. CDFs of score values for generated flows limited by a specific root cause.

C. Performance Measurements

We evaluate the performance of our implementation to assess its scalability and to determine performance bottlenecks. The used Device under Test (DuT) runs an Intel Xeon CPU E5-2620 processor with 2.00 GHz clock rate and possesses 128 GiB memory. Traffic is generated as described in Section V-A. We replay the captures at a rate of 2 Gbit/s to overload the DuT. We only run a single analyzer thread, i.e., the analysis is done on a single core.

According to our design, we try to do as few calculations as possible on a per-packet base and to offload such expensive calculations to the periodical calculation checks. This circumstance motivates to optimize the performance of single modules and to analyze the costs of each module.

We measure the run time of the periodical calculations for all different modules with a single flow. As shown in Table II, most modules take less or around 20 ns. However, three modules consume significantly more run-time - especially the capacity estimation module with over 15 ms, and the dispersion score calculation with over 2.5 ms. The calculation of the dispersion score requires to work on time-series data for the receiver advertised window and the number of outstanding bytes. Working on time-series data probably causes expensive lookups.

The long run time of the capacity estimation can be traced back to expensive operations like the differentiation of histograms and operations on a sliding window of previously observed data. The module was configured to work on a sliding window of 1000 packets.

Such a long run time of single modules can cause a limitation regarding the number of concurrently processed flows. Assuming a load of 70 flows and a capacity estimation module runtime of up to 15 ms, might result in a run time of over one second for a single calculation run. Therefore, the periodical calculations would be too slow for an interval of one second. While this might be mitigated by increasing the time between check intervals, it is still a trade-off between scalability and analysis effectiveness.

Another potential performance limitation of our analysis tool is the analyzer throughput. To avoid influences by too extensive run times, we disable the capacity estimation module for the benchmarking of the analyzer throughput. We start measuring analyzer throughput for a single flow and achieve a data rate of 620 Mbit/s.

In further measurements, we analyze the impact of the number of concurrently analyzed flows and increase the number of current flows up to 4096. Figure 7 shows the analyzer throughput and the aggregated run time of all modules. For up to 32 concurrent flows throughput decreases continuously. Afterward, throughput decreases significantly for the next increments of the flow count. We trace this observation back to the Last Level Cache (LLC) of the CPU. Significant performance decrease occurs when the aggregated size of all connection states approaches the size of the LLC. When the LLC is full, connection states have to be loaded from main memory, what makes packet analysis more expensive. In our measurements, the effect of the LLC is observed between a number of 32 and 64

TABLE II
LIST OF INCLUDED MODULES AND THE RUNTIME OF ONE PERIODICAL CALCULATION.

Module	Runtime
Connection identification	21 ns
ALP detection	6 ns
Position estimation	2 ns
RTT estimation	9 ns
Capacity estimation	15167 ns
Dispersion score calculation	94 ns
Retransmission score calculation	13 ns
Receiver window score calculation	2536 ns
Burstiness score calculation	21 ns

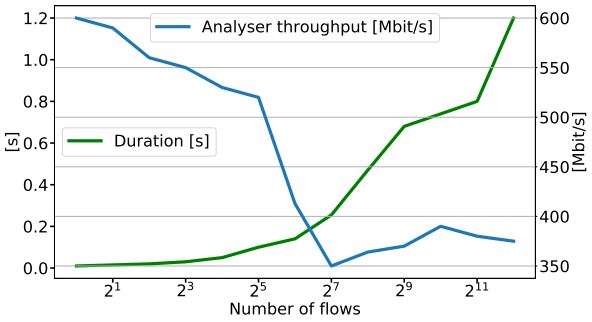

Fig. 7. Analyzer throughput and aggregated run time of periodical calculations.

flows. This matches our assumption, as 64 flows have an aggregated connection state size of about 11 MB with the used connection state size of 156 KB. The DuT has an LLC size of 12 MB. After the drop due to the filled LLC, throughput remains constant for increasing flow counts. As expected, the aggregated module run time increases linearly with the number of concurrent flows.

VI. Related Work

As already mentioned, this work is highly related to the TCP RCA toolkit proposed by Siekkinen et al. [1], [2]. In this paper, we present an approach to perform the described root cause analysis in real-time. While Siekkinen et al. focus on long-lived TCP connections, further research surveyed the performance of shorter TCP connections, like Barakat et al. [14] and Zhang et al. [15]. Such approaches might extend the capabilities of our online RCA tool in future work. An analysis of flow rates on the Internet is introduced by Zhang et al. [16] in 2002. One outcome of this research is the TCP rate analysis tool T-RAT purposed to determine the cause behind observed flow rates.

However, the analysis of TCP flows and their performance, in general, are a frequently studied topic. Bak et al. [17] introduce a root cause analysis approach considering root causes like packet loss, anomalies, or network limited throughput. A review of bottlenecks in content delivery networks was published by Peng et al. [18] who introduce a tool to monitor and analyze TCP statistics. Similar work was done by Jaiswal et al. [19] who analyzed TCP characteristics based on congestion window sizes and RTT. Hagos et al. [20], [21] recently studied the field of TCP state monitoring and prediction with the help of machine learning. The impact of BBR on loss-based congestion control algorithm was recently surveyed by Ware et al. [22].

For the implementation of our passive capacity estimation tool, we mainly follow the algorithm from the PPrate tool published by En-Najjary et al [8]. PPrate relies on the packet dispersion technique. Further research studied capacity estimation based on packet dispersion. For example, Katti et al. [23] presented the capacity estimation and bottleneck detection tool Multiq and Dovrolis et al. [24] survey packet dispersion techniques in more details.

VII. Conclusion and Future Work

In this paper, we describe an implementation of TCP throughput limitation analysis in real-time and survey design decisions for such sophisticated online traffic analysis. We adapt existing approaches for online analysis requirements and evaluate our implementation regarding effectiveness and performance. Our evaluation shows, that online throughput limitation analysis is feasible in real-time with reasonable performance on commodity hardware, i.e., with a rate of over 350 Mbit/s for 4096 parallel flows on a single analyzer core.

We find several trade-offs between the accuracy and responsiveness of the tool and performance. The analysis of sliding windows requires long analysis run times, while run time is limited due to periodical calculations. Our analysis shows, that memory concerns are not crucial for modern commodity hardware. Our tool requires approximately 160 GB memory for one million concurrent flows with a connection state size of 156 KB. This justifies fixed state size per flow, which is reasonable to avoid expensive resizing of hash table entries.

For future work, we plan the parallelization of our tool across several CPU cores. Such parallelization is expected to enable an analysis of 10 Gbit/s links on COTS hardware. Additionally, the optimization of the capacity estimation module regarding larger capacities and methods to automatically derive reasonable thresholds for the limitation factor classification are considered for further research.

We provide our code as free and open-source [25].

Acknowledgments

This work has been supported by the German Federal Ministry of Education and Research, project X-Check (16KIS0530), the DFG project MoDaNet (CA595/11-1), and the German-French Academy for the Industry of the Future.

References

[1] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and D. Collange, "A root cause analysis toolkit for tcp," *Comput. Netw.*, vol. 52, no. 9, pp. 1846–1858, Jun. 2008.

[2] M. Siekkinen, G. Urvoy-Keller, and E. W. Biersack, "On the interaction between internet applications and tcp," in *Proceedings of the 20th International Teletraffic Conference on Managing Traffic Performance in Converged Networks*, ser. ITC20'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 962–973.

[3] "Transmission Control Protocol," RFC 793, Sep. 1981.

[4] V. Jacobson, R. Braden, and D. Borman, "Tcp extensions for high performance," United States, 1992.

[5] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Sep. 2009.

[6] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant." *Operating Systems Review*, vol. 42, no. 5, 2008.

[7] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a deeper understanding of tcp bbr congestion control," in *IFIP Networking 2018*, May 2018.

[8] T. En-Najjary and G. Urvoy-Keller, "Pprate: A passive capacity estimation tool," in *2006 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, April 2006, pp. 82–89.

[9] P. Emmerich, M. Pudelko, S. Gallenmüller, and G. Carle, "Flowscope: Efficient packet capture and storage in 100 gbit/s networks," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, June 2017, pp. 1–9.

[10] P. Emmerich, M. Pudelko, Q. Scheitle, and G. Carle, "Efficient dynamic flow tracking for packet analyzers," 10 2018, pp. 1–6.

[11] B. Jaeger, D. Scholz, D. Raumer, F. Geyer, and G. Carle, "Reproducible Measurements of TCP BBR Congestion Control," *Computer Communications*, May 2019.

[12] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.

[13] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: ACM, 2015.

[14] C. Barakat and E. Altman, "Performance of short tcp transfers," in *Proceedings of the IFIP-TC6 / European Commission International Conference on Broadband Communications, High Performance Networking, and Performance of Communication Networks*, ser. NETWORKING '00, 2000.

[15] Y. Zhang, L. Qiu, and K. Srinivasan, "Speeding up short data transfers: Theory, architectural support, andsimulation results," Cornell University, Tech. Rep., 2000.

[16] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates," in *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '02. New York, NY, USA: ACM, 2002.

[17] A. Bąk, P. Gajowniczek, and M. Zagożdżon, *Analysis of TCP Connection Performance Using Emulation of TCP State*, 12 2017, vol. 461.

[18] P. Sun, M. Yu, M. Freedman, and J. Rexford, "Identifying performance bottlenecks in cdns through tcp-level monitoring," 08 2011.

[19] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring tcp connection characteristics through passive measurements," in *IEEE INFOCOM 2004*, March 2004.

[20] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure, "A machine learning approach to tcp state monitoring from passive measurements," in *2018 Wireless Days (WD)*, April 2018.

[21] D. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure, "Recurrent neural network-based prediction of tcp transmission states from passive measurements," 11 2018, pp. 1–10.

[22] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Modeling bbr's interactions with loss-based congestion control," in *Proceedings of the Internet Measurement Conference*. ACM, 2019.

[23] S. Katti, D. Katabi, C. Blake, E. Kohler, and J. Strauss, "Multiq: Automated detection of multiple bottleneck capacities along a path," 01 2004, pp. 245–250.

[24] C. Dovrolis, P. Ramanathan, and D. Moore, "Packet-dispersion techniques and a capacity-estimation methodology," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 963–977, 2004.

[25] S. Bauer, K. Holzinger, F. Wiedner, B. Jaeger, and P. Emmerich, "Online tcp limitation monitor," https://github.com/bauersi/online-TCP-limitation-monitor, 2020.