Scalable TCP Throughput Limitation Monitoring

Simon Bauer, Florian Wiedner, Benedikt Jaeger, Paul Emmerich, Georg Carle
Technical University of Munich (TUM), Department of Informatics
Chair for Network Architectures and Services, Garching b. München, Germany
{bauer, wiedner, jaeger, emmericp, carle}@net.in.tum.de

***Abstract*—The analysis of TCP throughput limitations is purposed to determine the limitation preventing a TCP connection to increase its throughput. Monitoring throughput limitations of TCP connections enables detecting network misconfiguration, overload, and other anomalies while connections are still active in the monitored network. Therefore, TCP throughput limitation monitoring enables reactions to such incidents in real-time to improve per-flow and network-level performance.**

This paper presents a multi-threaded TCP throughput limitation monitoring framework providing scalability due to fully parallelized analysis pipelines. We benchmark our framework's performance and use our implementation to conduct a case study on real-world Internet traffic captured on 10 Gbit/s links.

Our framework shows to scale up linearly and to be capable of monitoring workloads of several Gbit/s distributed on several ten thousands of concurrent flows on commodity hardware. We use a real-world Internet trace set including over 9 million TCP flows to assess the share of flows suitable for our monitoring approach, interpret distributions of measured limitation scores, and estimate the throughput limitations of the analyzed TCP connections.

***Index Terms*—TCP, throughput limitations, traffic monitoring, high-performance**

I. Introduction

The Transmission Control Protocol (TCP) is the dominant transport layer protocol in use. Due to providing reliable data transmission, TCP is the used Layer 4 protocol for many types of network traffic like web applications or file transmission protocols. This fact significantly increases interest in the analysis of the throughput and the throughput limitation of TCP flows. Analyzing TCP throughput limitations supports revealing network entities' misconfiguration, network debugging, and other administrative tasks like network resource planning. So far, previous studies introduced approaches to determine the throughput limitation of TCP flows also referred to as TCP root cause analysis [1]–[3]. These approaches expect network traffic captures as input and, therefore, only allow the offline analysis of captured traffic. In previous research [4] we introduced an approach to make the online monitoring of TCP throughput limitations feasible. We evaluated our implementation based on synthetically generated traffic in a controlled test environment. Our proof of concept implementation showed to be limited by computational resources restricting analysis throughput to about 500 Mbit/s on commercial off-the-shelf hardware.

In this paper, we survey the use of our approach to analyze TCP throughput limitations in real-time in productive environments considering traffic rates of several Gbit/s on commodity hardware. We survey how to implement scalable and productively usable TCP throughput limitation monitoring and survey the performance limitations of such an implementation. Besides, we examine the question of what limits TCP throughput in real-world Internet traffic.

We present a linearly scalable framework to monitor TCP throughput limitations, that is capable of analyzing several Gbit/s of TCP traffic consisting of millions of TCP flows in real-time. Based on this framework, we conduct a case study on real-world Internet traffic traces provided by the MAWI Working Group Traffic Archive [5] and contribute a detailed analysis of measurement results.

The remainder of this paper is structured as follows: First, we briefly introduce background and terminology on offline and online analysis of TCP throughput limitations. Next, we describe the optimization of our framework regarding new requirements for large-scale traffic monitoring. Afterward, we evaluate the performance of our framework and present results of measurements on real-world Internet traffic. We conclude our work with an overview of related work and progress in TCP performance analysis and high-performance network monitoring.

II. Background

This section introduces terminology and background knowledge regarding approaches to determine TCP throughput limitations and to monitor them in real-time.

A. Offline Analysis of TCP Throughput Limitations

In 2007 and 2008, Siekkinen et al. published approaches to determine the limiting factor of the throughput of TCP connections for captured traffic [2], [3]. The authors' approach considers two steps. The first step uses the Isolate and Merge algorithm [2] to separate a flow into three types of periods. Such periods are application limited periods (ALPs), short transfer periods (STPs), and bulk transfer periods (BTPs). During ALPs, the throughput does not increase since not enough data is ready to be transmitted. Periods containing less than 130 packets are referred to as STPs. They are not considered for further analysis as they are assumed to be dominated by congestion control algorithms [3]. For bulk transfer periods, there are five different limitations of throughput: unshared bottleneck links, shared bottleneck links, the TCP receiver window, the transport protocol, i.e., TCP itself, or causes that can not be clearly identified.

978-3-903176-32-4

To differentiate between throughput limitations of bulk transfer periods, Siekkinen et al. [3] introduce four limitation scores to assess the impact of a potential root cause. These scores are later used to determine the limiting factor of a connection based on a decision tree that compares score values to predefined threshold values.

The first limitation score is the dispersion score, which compares the throughput of a flow to the capacity of the path of the flow. The dispersion score is purposed to detect limitations by an unshared bottleneck link. An unshared bottleneck link is wholly utilized by a single flow and, therefore, limits further throughput increase. The second score is the retransmission score proposed to detect limitations by shared bottleneck links. This score compares the amount of retransmitted data to the amount of totally transmitted data for a certain bulk transfer period. This way, the retransmission score determines the impact of packets getting dropped at so-called shared bottleneck links. Shared bottleneck links limit the throughput of flows as they are thoroughly utilized by several concurrent flows competing for available bandwidth. Next, Siekkinen et al. introduce the receiver window score that estimates the data that can be sent until the receiver window is wholly used and prevents higher throughput rates. To detect shared bottleneck links that do not cause retransmitted packets due to large buffers that prevent packets from getting dropped, Siekkinen et al. introduce the burstiness score. The burstiness score uses the relation between the average advertised receiver window, the maximum segment size (MSS), capacity, and round trip time (RTT) to assess traffic burstiness within a certain period.

B. Online Monitoring TCP Throughput Limitations

In previous research [4], we introduced an approach to analyzing throughput limitations of TCP flows online. We rely on the limitation score-based approach of Siekkinen et al. and described adaptations to the original algorithms to make calculations of limitation scores feasible in real-time. We contributed a prototype implementation of our approach based on the high-performance traffic capture and analysis tool FlowScope [6], [7] and evaluated our prototype with a synthetically generated labeled data set.

Our approach implies several adaptations to the original approach to satisfy online analysis requirements. For instance, we abandon the Isolate and Merge algorithm's merge step as this step requires several iterations over previously detected transfer periods. As this step is not feasible in real-time, we renounce to merge transfer periods. Instead, we use sliding windows to track previously extracted packet information for each flow and calculate the limitation scores based on such windows. The sliding windows of a flow containing the extracted packet data are stored in a flow's connection state.

Our approach considers two steps of packet data processing to reduce the computational load per packet. First, we extract the necessary information per packet and store it to the flow's connection state. Second, the computationally expensive calculation of limitation scores is carried out in fixed time intervals, referred to as periodic result calculations. Our prototype stores connection states as hash table entries. To avoid expensive resizing of hash table entries, we use a static connection state size of 156 kB, which is an upper bound to the aggregated sliding window sizes per flow.

III. Framework Implementation for Large-scale Analysis

This section introduces optimizations to the existing prototype implementation [4] to enable scalable monitoring capabilities and describes their implementation.

As our framework is purposed to monitor network links with data rates of several Gbit/s, we require efficient use of computational resources and a scalable analysis design. Our prototype implementation already considers efficient packet analysis [4]. Efficiency is achieved by reducing computational load per packet by offloading expensive calculations, e.g., calculating limitation scores, to periodically performed result calculations. However, to scale up monitoring capabilities, we consider the parallelization of the whole traffic analysis pipeline.

A. Multi-threaded Traffic Analysis

One primary goal of our framework is to scale up monitoring capabilities to achieve analysis throughputs of several Gbit/s. Therefore, we implement the parallelization of the complete analysis pipeline used for the existing prototype implementation.

Our framework uses the packet capturing and analysis tool Flowscope [6], [7] as a base for packet processing and analysis. FlowScope natively supports multi-threaded packet analysis based on Receive Side Scaling (RSS). RSS enables NICs to distribute packets to several RSS queues, while each RSS queue is mapped to a specific CPU core. The distribution of packets to RSS queues relies on hardware-based hashing of packet header data. So far, FlowScope only supports the multi-threaded analysis of unidirectional traffic, as RSS natively does not ensure that packets of both directions of a TCP flow are mapped to the same RSS queue. However, our monitoring approach requires that the same analysis pipeline analyzes both directions of a flow. To meet the bidirectional RSS requirement, we extend FlowScopes RSS capabilities to support symmetric hashing of packet data. We use a symmetric hash key introduced by Woo et al. [8]. To fully exploit speed up by RSS, we parallelize all further steps of the analysis pipeline. Each RSS queue provides received packets to a pipeline consisting of four threads:

- First, a thread responsible for copying packets from the RSS queue to FlowScope's packet buffer.
- A thread that reads packets from the packet buffer to perform per-packet analysis and feature extraction.
- A third thread to run periodic result calculations based on data in the connection states and to detect expired flows.
- A thread to finally expire flows by deleting connection states.

B. Flow Handling

Appropriate flow handling is a central aspect of our monitoring approach that impacts performance and our framework's applicability. We refer to a flow as packets with the same 5-tuple consisting of server and client IP addresses, ports, and the used L4 protocol. From an analysis point of view, we can only consider flows for that we observe the complete TCP three-way handshake. Observing the handshake is required to extract data necessary to run the TCP throughput limitation analysis, such as the MSS, sender and receiver information, and the measurement position.

To optimize the memory consumption of our framework, we take care of proper flow expiration. As soon as a flow is expired, its connection state is no longer needed, and there is no need to consider the flow for periodic result calculation further. Therefore, flow expiration contributes to efficient memory and CPU utilization. A flow is expired if we do not observe a packet with the corresponding 5-tuple for a specific time interval. For our measurements, we use a time interval of 180 seconds.

IV. Evaluation

A. Test setup

To evaluate our implementation's performance and to conduct a case study on real-world Internet traffic captures, we run measurements in a test setup consisting of three physical hosts. We replay packet captures from a dedicated host referred to as Load Generator. The Load Generator runs the packet generator MoonGen, capable of replaying packet captures at rates of several Gbit/s [9]. Our monitoring framework runs on a second host referred to as Device under Test (DuT). A third host provides an ElasticSearch database backend to store results for further analysis. The LG and the DuT are connected with 10 GE cabling. The DuT is connected to the backend host using 1 GE cabling. The LG is equipped with two AMD EPYC 7601 CPUs, each capable of 32 physical cores with 2.2 GHz clock frequency, 1 TB of DDR4 main memory, and an Intel X550T 10 GE NIC connected to the DuT. The DuT is capable of two Intel Xeon Gold 6130 CPUs with 16 physical cores per socket with 2.2 GHz clock frequency, 395 GB DDR4 main memory, and two Intel X722 10 GE NICs. As hyper-threading is enabled, each socket provides 32 virtual cores. The backend host provides the same hardware configuration as the DuT. All hosts run Debian stretch.

B. Performance Measurements

In previous research [4], we found two significant performance limitations of our prototype implementation: a) computational resources and b) memory consumption. To survey our optimized and extended monitoring framework's analysis capabilities, we benchmark our tool with a synthetically generated data set and packet traces taken from the MAWI Working Group Traffic Archive [5].

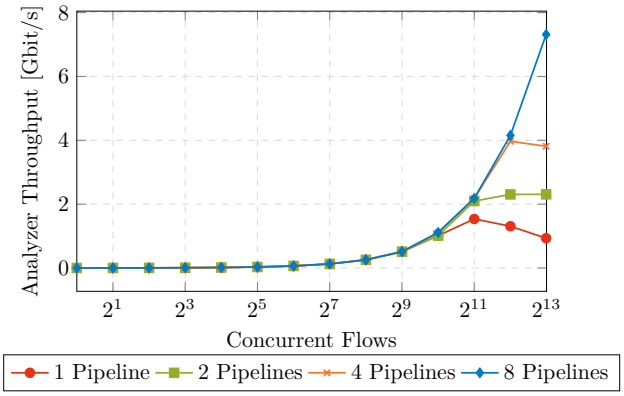

Fig. 1. Analyzer throughput for varying numbers of concurrent flows.

1) Synthetic Data Set for Performance Evaluation: To benchmark our monitoring framework's analysis throughput, we generate a data set consisting of synthetically generated TCP traffic. We use the traffic generator iperf to generate TCP flows with 1500 B packet size, a data rate of 1 Mbit/s, and a duration of 10 seconds. iperf is limited in the number of concurrently generated flows as too many concurrent flows result in inaccurate data rates per flow. We observe such inaccuracies for flow counts larger than 128. Therefore, we capture several traces with 128 concurrent flows and merge them to achieve larger numbers of concurrent flows. The resulting data set provides traces from 1 flow up to 8192 concurrent flows. As we generate flows at 1 Mbit/s data rate, the maximum expected data rate in our data set is around 8 Gbit/s corresponding to about 1.2 million packets per second. However, we found that the achieved average data rate for the trace with 8192 flows is a little lower than expected, i.e., about 7.5 Gbit/s, due to inaccuracies in the merged PCAPs.

2) Performance Evaluation: We measure the achievable throughput of the analysis pipelines with the generated data set. We find that a single pipeline can analyze 1024 concurrent flows without dropping packets or ignoring flows for the analysis. With increasing load, i.e., more than 1024 concurrent flows, packets get dropped due to fully exploited computational resources, and the maximum achieved analysis throughput is around 1.5 Gbit/s for 2048 flows. We observe that throughput additionally decreases for measurements with a single pipeline and flow counts larger than 2048 down to 1 Gbit/s. For setups with more than one single pipeline, we find a throughput of approximately 1 Gbit/s per pipeline. This observation holds for all setups, even for eight parallel pipelines and an offered load of 8192 concurrent flows. The measured analyzer throughput for all pipeline setups is shown in Figure 1. We conclude that these observations indicate the linear scalability of our framework due to independent analysis pipelines.

Next, we want to assess the memory consumption of our framework. Therefore, we conduct measurements with traces taken from the MAWI Working Group Traffic Archive [5]. Choosing such traces for evaluating memory consumption

TABLE I
USED MAWI TRACES

Date, Time	Avg. Data Rate	Packets	TCP Flows
2019/01/02, 13:59-14:14	1.532 Gbit/s	136 M	526 K
2019/01/09, 14:00-14:15	2.684 Gbit/s	300 M	1,634 K
2019/01/16, 14:00-14:15	2.781 Gbit/s	308 M	1,684 K
2019/01/23, 14:00-14:15	2.823 Gbit/s	280 M	1,319 K
2019/01/30, 14:00-14:15	3.033 Gbit/s	278 M	1,208 K
2019/02/06, 14:00-14:15	2.214 Gbit/s	211 M	1,047 K
2019/02/13, 14:00-14:15	2.601 Gbit/s	245 M	931 K
2019/02/20, 14:00-14:15	1.967 Gbit/s	205 M	1,130 K
2019/02/27, 14:00-14:15	2.831 Gbit/s	260 M	1,001 K

is motivated by the expectation that these real-world traces carry larger numbers of concurrent flows than our synthetically generated dataset. We run measurements with six, respectively 16, parallel analysis pipelines and observe the used memory during the analysis. The setup with six pipelines cannot analyze all packets of some traces due to computational limitations. However, the setup with 16 pipelines is capable of analyzing all traces without reaching computational limits. We find a maximum memory consumption of 65 GB for six parallel pipelines, respectively 160 GB for the setup with 16 pipelines. In addition to the connection state size allocated in memory for each analyzed flow, our tool consumes significant amounts of memory per pipeline to provide the ring buffer QQ that Flowscope uses to buffer received traffic before processing it to further analysis steps. By default, our framework allocates 8 GB per analysis pipeline to maintain the QQ buffer. For measurements with six analysis pipelines, we observe a maximum memory consumption of about 65 GB, while 48 GB are allocated by the QQ instances. During the measurements with 16 pipelines, we find a maximum memory consumption of 160 GB, while 128 GB are allocated by QQ instances. These numbers indicate that the setup with 16 pipelines allocates connection states up to 32 GB during the measurements with the used trace set. Observed values vary slightly for each dataset as maximum memory consumption depends on the highest amount of concurrent flows in a particular input data period. We observe that our framework can analyze all considered traces with 16 parallel analysis pipelines without any packets dropped due to overload, while traces include traffic peaks up to 4 Gbit/s.

C. Case Study

To show our framework's analysis capabilities, we conduct measurements with real-world Internet traffic taken from the MAWI Working Group Traffic Archive [5]. In this section, we survey the applicability of our framework outside controlled test environments and analyze the TCP throughput limitations of real-world Internet traffic.

1) Data Set Characteristics: We use packet captures published by the Measurement and Analysis on the WIDE Internet (MAWI) Working Group for our measurements. The MAWI Working Group provides traffic traces captured on different sample points on the WIDE backbone network. We choose nine traces captured on Samplepoint G in January and February 2019. Each trace provides 15 minutes of captured traffic. Such traces carry packet counts between 136 million and 308 million. Per trace we find TCP flow counts between 500k and over 1,600k and average data rates between 1.5 Gbit/s and 3 Gbit/s. All used traces are listed in Table I.

Our framework requires a minimum amount of data packets per flow. Such a minimum is necessary to provide enough input data to the analysis modules, such as the capacity estimation module, which is implemented according to the PPrate algorithm [10]. We observe that 300 packets are a suitable lower bound to avoid more considerable inaccuracies of capacity estimates in controlled test environments. Therefore, we configure our framework only to consider flows carrying more than 300 data packets for further analysis. This significantly decreases the number of considered flows per trace. We find that the vast majority of flows in our data set consist of fewer than 300 packets considering both directions of a flow. The 97th percentile of packets per flow lies between 150 and 340 packets for all traces. The 99th percentile of packets per flow significantly exceeds 1000 packets for the most traces. This observation implies that only between 1 % and 2 % of all observed TCP flows are relevant for our analysis. We observe an exponential distribution of packets per flow. I.e., we find large numbers of very short flows and tiny numbers of large flows up to several million packets per flow. Next to a sufficient number of packets per flow, a captured flow is required to provide all required handshake information to be considered for further analysis. This implies that flows that began before traffic capturing started can not be analyzed. In total, our framework was able to successfully estimate the capacity for 29K flows that provide full handshake information across all analyzed MAWI traces.

Further, we analyze the distribution of server ports across the analyzed traces. We observe that nearly 50 % of all flows are addressed to well-known server ports, i.e., port 1 to 1024. For port numbers higher than 1024, flows are distributed relatively equally. Furthermore, our server ports analysis reveals that the significantly dominating ports are port 80 and port 443. Due to the dominant share of web traffic, we decide to focus on web traffic for the detailed analysis of throughput limitations next to the remaining well-known ports.

2) Capacity, Throughput, and Application Limitation: As explained above, we require a successful capacity estimate to analyze a flow's throughput limitation. Therefore, we consider successful capacity estimation as a prerequisite for further analysis of a flow. We find that 50 % of all estimated capacities are lower than 2 Gbit/s and only observe minor deviations between the distributions of capacities for port 80, port 443, and the remaining well-known TCP ports. We measure an average throughput of less than 10 Mbit/s for over 60 % of all score calculation intervals. For over 90 % of calculation intervals, we measure an average throughput lower than 100 Mbit/s. Beside, we observe outliers with throughputs higher than 1 Gbit/s. As expected, the cumulative distribution of measured capacities is an upper bound to the cumulative distribution of measured

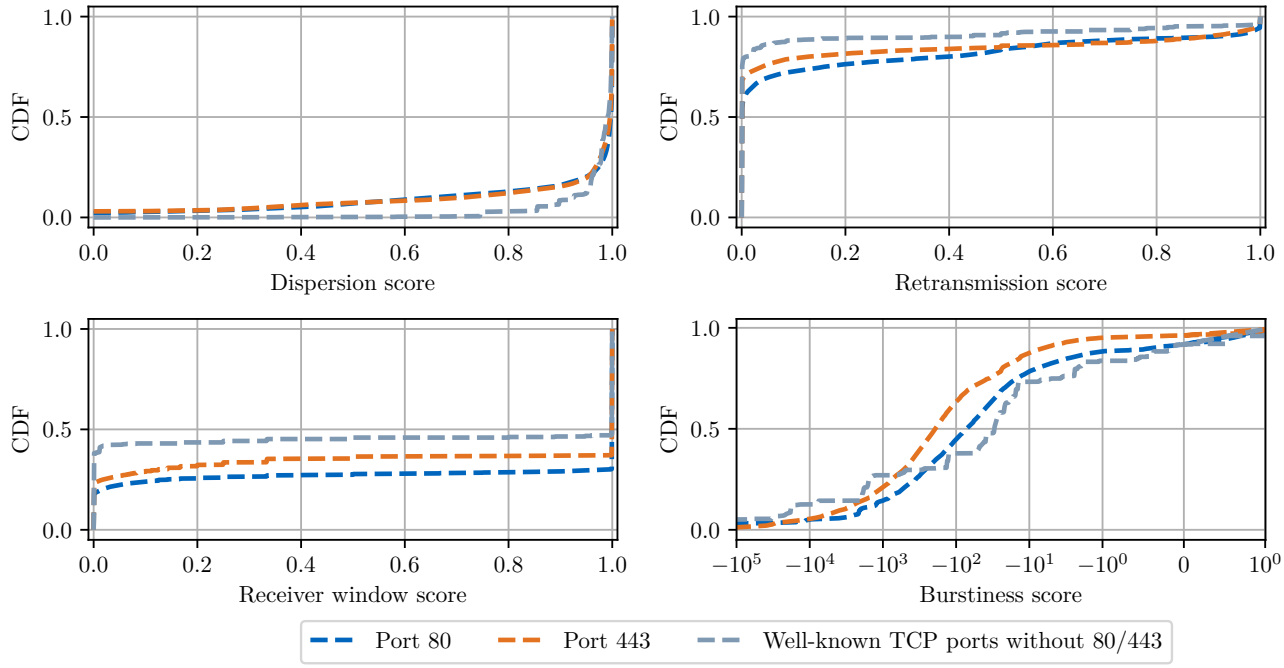

Fig. 2. Cumulative distributions of measured score values.

average throughput.

Before evaluating measured limitation scores, we survey the share of application limitations. Application limited periods significantly impact our approach as a) limitation scores are only calculated during bulk transfer periods and b) we reset the sliding windows containing extracted packet data of a flow after each ALP. We are interested in the time flows spend in application limited periods and find that the share of application limitation is near zero for nearly all flows in the analyzed MAWI traces.

3) Limitation Scores: In the following, we present measurement results for the limitation scores used to determine the throughput limitation of a TCP connection. The formulas for the limitation scores are specified by Siekkinen et al. [3]. Figure 2 shows the cumulative distribution for measured limitation scores. As our framework calculates results periodically based on a defined time interval, we likely receive several calculated limitation score vectors per flow. We configure the framework to calculate scores at an interval of three seconds. On average, each flow with successfully estimated capacity is considered around eight times for the periodic score calculations. This implies that flows may contribute several results to our score analysis depending on the flow duration.

a) Dispersion Score: The dispersion score compares the average throughput of a flow during the current bulk transfers period, respectively, the current sliding window, to the capacity estimate of the flow. Small dispersion score values indicate a limitation by an unshared bottleneck link. In our measurements, less than 5 % of all calculated dispersion scores are smaller than 0.8.This observation indicates that most flows utilize less than 20 % of the corresponding network path capacity.

b) Retransmission Score: The retransmission score is an indicator of shared bottleneck links. The score compares the amount of retransmitted data to the total amount of transmitted data to assess the impact of packets being dropped due to fully utilized links. We determine a packet to carry retransmitted data if the packet's sequence number is smaller than the largest sequence number observed so far.

We find that 60 % of all calculated retransmission scores are zero for flows on server port 80. Regarding the remaining well-known ports we measure a retransmission score equal to zero for 80 % of all calculation periods. At the same time, we observe a small share of relatively large retransmission scores near one. Such high score values indicate that nearly all packets are classified as retransmitted data. We explain such large values with the missing implementation of packet reordering. If packets are not received in order, our framework assumes such packets to be retransmissions. This assumption indicates that our framework potentially suffers from many false-positives during the detection of retransmitted data. We consider analyzing the impact of such unordered packets on the retransmission score for future research.

c) Receiver Window Score: The next score is the receiver window score used to assess whether the receiver window size limits further throughput increase. A receiver window score close to 1 indicates that the connection uses a large share of the receiver window size and, therefore, tends to be limitted by the receiver window. The distribution of measured score values reveals that the majority of receiver window scores are either 0 or 1 and that only 5 % to 10 % of the scores show a value unequal 0 and 1. We find that between 20 % and 30 %

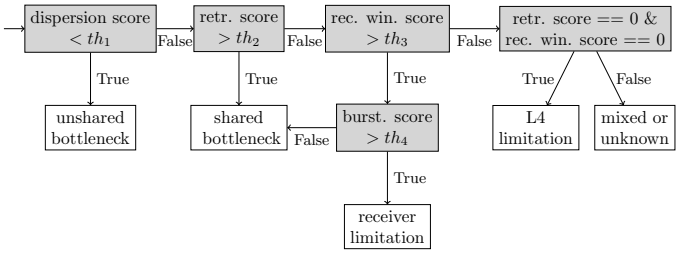

Fig. 3. Decision tree to determine throughput limitations by Siekkinen et al. [3].

of all calculated score values equal 0 and that around 50 % and 60 % of all calculated score values are equal to 1.

d) Burstiness Score: The burstiness score is used to assess a flow's burstiness, which indicates whether a connection is affected by large buffers that prevent shared bottleneck links from dropping packets. Larger score values indicate no impact by buffered packets, while smaller values imply more significant bursts inside the analyzed flow. We measure burstiness scores between -10^7 and 1. Over 75 % of all calculated scores are greater than -1000. Based on previous studies [4] of the burstiness score with a synthetic data set we assess values greater than -1000 as relatively large. Therefore, we expect no significant burstiness for the majority of the analyzed flows.

4) Root Cause Estimation: To determine the actual limitation of the analyzed traffic, we apply the decision tree-based analysis of limitation score vectors as purposed by Siekkinen et al. [3]. The decision tree, as shown in Figure 3, compares predefined threshold values to measured scores. Siekkienen et al. propose threshold values for the dispersion score ($th_1 = 0.2$), retransmission score ($th_2 = 0.01$), and receiver window score ($th_3 = 0.5$). There is no purposed threshold value for the burstiness score as the original approach relies on the b-score to approximate the actual burstiness score. Therefore, we define the corresponding threshold value $th_4 = -1000$ based on experiences with a synthetically generated data set.

We determine that shared bottleneck links limit the throughput of over 35.1 % of all analyzed flow intervals. The second most common throughput limitation is the receiver window size, which was determined for over 32.1 % of all intervals. The remaining shares of estimated root causes are 12.2 % for L4 limitations, 5.3 % for unshared bottlenecks links, and 15.1 % of measured score vectors result in mixed, respectively unknown, throughput limitations.

V. Related Work

The analysis of TCP throughput limitations has a lengthy history in the field of network traffic analysis. Early publications work out potential limitations and approaches to identify them. As mentioned before, our approach is based on the TCP root cause analysis approach introduced by Siekkinen et al. [2], [3], while Zhang et al. [1] initially introduced potential TCP throughput limitations. Bak et al. [11] present a throughput degradation analysis approach based on a decision tree similar to Siekkinen et al. and evaluate their approach with 4G wireless Internet traffic. While the mentioned approaches focus on flows carrying large amounts of data, several publications surveyed the performance of short TCP connections [12]–[14]. Arajo et al. [15] survey the impact of the network, hosts, and applications on TCP throughput rates in a longitudinal study for MAWI traces from 2006 to 2011. Authors show, that the mentioned entities constraint over 50 % of analyzed TCP traffic. Regarding the real-time analysis of TCP performance, Ghasemi et al. [16] present a P4 prototype called Dapper, that integrates the analysis of TCP performance to the data plane. The integration of TCP analysis capabilities to softwarized networks were surveyed by Singh et al. [17] and Kagami et al. [18]. Next to the analysis of TCP throughput limitations, recent publications survey the performance of congestion control algorithms [19], [20], or focus on the optimization of TCP performance indicators like throughput [21] or latency [22]. Besides analyzing TCP performance and limitations in general, approaches exist to assess and analyze traffic in more specific scenarios, such as Loh et al. [23] who study video stalls in encrypted video streaming traffic to support QoE analysis. Other researchers survey the modeling of service performance [24] and automation of monitoring deployments [25].

Regarding the performance of traffic analysis and monitoring frameworks, several recent publications survey the suitability of DPDK-based tools for such tasks. Ren et al. [26] introduce the high-performance packet processing and analysis framework PacketUsher. Zhang et al. [27] present a DPDK-based tool to monitor flows at line rate of several Gbit/s, while Trevisan et al. [28] presented an implementation of statistical traffic analysis up to 40 Gbit/s.

VI. Conclusion

We presented an implementation of a linear scalable TCP throughput limitation monitoring framework. Our implementation is capable to monitor several Gbit/s of traffic and several thousands of concurrent flows on commodity hardware. We find that the use of 16 analysis pipelines, corresponding to 64 CPU threads, and 160 GB of memory is sufficient to monitor replayed packet traces with data rate peaks up to 4 Gbit/s in real-time. We conclude that TCP throughput limitation monitoring is feasible in large-scale environments.

In a case study on captured Internet traffic we found shared bottleneck links and the receiver window's utilization as the dominant throughput limitations. We observe that only 1 % up to 2 % of all TCP flows are relevant for our analysis due to the number of packets per flow in the considered MAWI data set.

Acknowledgments

This work was supported by the German Research Foundation (project ModANet under grant no. CA595/11-1) and by the German-French Academy for the Industry of the Future.

REFERENCES

[1] 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.

[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.

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

[4] S. Bauer, K. Holzinger, B. Jaeger, P. Emmerich, and G. Carle, "Online Monitoring of TCP Throughput Limitations," in *2020 IEEE/IFIP Network Operations and Management Symposium (NOMS 2020)*, Budapest, Hungary, 2020.

[5] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the wide project," ser. USENIX 2000 FREENIX Track. USENIX, 2000.

[6] P. Emmerich, M. Pudelko, S. Gallenmller, 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.

[7] P. Emmerich, M. Pudelko, Q. Scheitle, and G. Carle, "Efficient Dynamic Flow Tracking for Packet Analyzers," in *CloudNet*, Tokyo, Japan, Oct. 2018.

[8] S. Woo and K. Park, "Scalable tcp session monitoring with symmetric receive-side scaling," KAIST, Daejeon, Korea, Tech. Rep., 2012.

[9] 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.

[10] 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.

[11] A. Bak, P. Gajowniczek, and M. Zagodon, *Analysis of TCP Connection Performance Using Emulation of TCP State*, 12 2017, vol. 461.

[12] 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.

[13] N. Cardwell, S. Savage, and T. Anderson, "Modeling the performance of short tcp connections," 1998.

[14] A. Hafsaoui, D. Collange, and G. Urvoy-Keller, "Revisiting the performance of short tcp transfers," vol. 5550, 05 2009.

[15] J. T. Araújo, R. Landa, R. G. Clegg, G. Pavlou, and K. Fukuda, "A longitudinal analysis of internet rate limitations," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014.

[16] M. Ghasemi, T. Benson, and J. Rexford, "Dapper: Data plane performance diagnosis of TCP," *CoRR*, vol. abs/1611.01529, 2016.

[17] M. Singh, N. Varyani, J. Singh, and K. Haribabu, "Estimation of end-to-end available bandwidth and link capacity in sdn," in *International Conference on Ubiquitous Communications and Network Computing*. Springer, 2017.

[18] N. S. Kagami, R. I. T. da Costa Filho, and L. P. Gaspary, "Capest: Offloading network capacity and available bandwidth estimation to programmable data planes," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, 2019.

[19] B. Jaeger, D. Scholz, D. Raumer, F. Geyer, and G. Carle, "Reproducible Measurements of TCP BBR Congestion Control," *Computer Communications*, vol. 144, May 2019.

[20] S. Patel, Y. Shukla, N. Kumar, T. Sharma, and K. Singh, "A comparative performance analysis of tcp congestion control algorithms: Newreno, westwood, veno, bic, and cubic," in *2020 6th International Conference on Signal Processing and Communication (ICSC)*, 2020.

[21] R. Kumar, A. Koutsaftis, F. Fund, G. Naik, P. Liu, Y. Liu, and S. Panwar, "Tcp bbr for ultra-low latency networking: challenges, analysis, and solutions," in *2019 IFIP Networking Conference (IFIP Networking)*. IEEE, 2019.

[22] I. Ali, T. Hussain, F. Perviz, and A. Hussain, "Analysis of tcp congestion control queuing mechanism and investigation for high throughput and low queuing delay," EasyChair, Tech. Rep., 2020.

[23] F. Loh, F. Wamser, C. Moldovan, B. Zeidler, D. Tsilimantos, S. Valentin, and T. Hofeld, "Is the uplink enough? estimating video stalls from encrypted network traffic," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020.

[24] F. Moradi, R. Stadler, and A. Johnsson, "Performance prediction in dynamic clouds using transfer learning," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019.

[25] F. Moradi, C. Flinta, A. Johnsson, and C. Meirosu, "Conmon: An automated container based network performance monitoring system," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017.

[26] Q. Ren, L. Zhou, Z. Xu, Y. Zhang, and L. Zhang, "Packetusher: Exploiting dpdk to accelerate compute-intensive packet processing," *Computer Communications*, vol. 161, 2020.

[27] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "Flowmon-dpdk: Parsimonious per-flow software monitoring at line rate," in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, 2018.

[28] M. Trevisan, A. Finamore, M. Mellia, M. Munafò, and D. Rossi, "Dpdk-stat: 40gbps statistical traffic analysis with off-the-shelf hardware," *Telecom, Paris, France, Tech. Rep*, vol. 318627, 2016.