Towards the Classification of TCP Throughput Changes

Simon Bauer, Benedikt Jaeger, Max Reimann, Jonas Fromm, Georg Carle
Technical University of Munich (TUM), Department of Informatics
Chair for Network Architectures and Services, Garching b. München, Germany
{bauer | jaeger | reimann | fromm | carle}@net.in.tum.de

***Abstract*—Analyzing throughput limitations of TCP connections has been a frequently studied topic. While existing approaches determine throughput limitations for whole connections or specific segments of connections, this paper surveys whether such approaches can also be used to classify individual changes in the throughput of a connection.**

In this paper, we introduce an approach to classify changes in TCP throughput based on their coincide with changes between TCP transfer periods. We evaluate different change point detection methods with generated TCP traffic providing ground truth data regarding throughput changes. Further, we survey the matching between throughput change points and transfer periods in passively captured Internet traffic. We conclude that the classification of TCP throughput changes with TCP transfer periods is feasible for a significant share of changes and observe significant differences in matching results depending on the used change point detection method.

***Keywords*—Change point detection, TCP throughput analysis, TCP transfer periods**

I. Introduction

Analyzing throughput limitations of TCP connections is purposed to support service and infrastructure providers to increase understanding of the performance of their networks, services, or customers. Such analysis, also referred to as TCP root cause analysis enables insights into the utilization of network resources and supports tasks like network resource planning or anomaly and incident detection. So far, TCP throughput limitation analysis determines the primary throughput limitation for a whole connection [8] or segments of a connection [1], [10].

This paper pursues whether approaches from the field of TCP throughput limitation analysis can be utilized to classify individual changes in the throughput of a TCP connection. As a first step towards the classification of TCP throughput changes, we use transfer periods according to Siekkinen et al. [1], which segment a connection into bulk transfers and application limited periods.

Accordingly, this paper examines the proportion of throughput changes that coincide with changes between transfer periods. For this purpose, we investigate the suitability of different change point detection methods for detecting changes in TCP throughput. We introduce a measurement framework that enables the evaluation of change point detection results with ground truth data and conduct passive measurements on Internet traffic to investigate how many of the detected changes can be classified using our approach. We find that the sensitivity of the change point detection method is crucial and survey the trade-off between the share of classifiable changes and matched period changes.

The outline of this paper is structured as follows: We introduce background regarding TCP transfer periods and change point detection methods in Section II. Section III describes our approach to the classification of changes in TCP throughput and the evaluation of different change point detection methods. In Section IV we describe the implementation of used traffic analysis and traffic generation tools. The evaluation of change point detection methods based on our generated data set is presented in Section V. Section VI presents results of measurements conducted on captured Internet traffic. We provide an outlook on related work in Section VII and conclude this paper with a summary of our findings and future work in Section VIII.

II. Background

This section introduces background on TCP throughput limitations, transfer periods as determined by the Isolate & Merge (I&M) algorithm, and change point detection (CPD) methods.

A. TCP Throughput and Throughput Limitations

The transmission control protocol (TCP) is purposed to determine fair data transmission rates for single connections to avoid overloading the passed network path and the receiver of sent packets. In order to determine a fair transmission rate, the sender of a TCP connection increases its transmission rate as far as there are no indicators for overload at other entities. The manner of determining the sending rate depends on the applied congestion control algorithm [2] that increases and decreases the congestion window for a connection based on different indicators [3]–[6]. We refer to a connection's data transmission rate as throughput in the following.

Besides congestion control, a TCP connection's throughput can be limited for different reasons. The analysis of such reasons is referred to as TCP root cause analysis or throughput limitation analysis and has been studied in several studies [7]–[10]. Examples of throughput limitations are bottleneck links on the network path, a slow receiving entity, or TCP itself, e.g., due to a congestion window that does not increase sufficiently

978-1-6654-0601-7/22/$31.00

fast. A further limitation to TCP throughput is a sending application with limited performance, i.e., when a sender does not produce enough data to exploit available network resources.

B. The Isolate & Merge Algorithm and TCP Transfer Periods

To determine time intervals in which the sending application limits a connection, Siekkinen et al. [1] introduce the Isolate & Merge (I&M) algorithm that segments a flow in three different kinds of periods:

- Application Limited Periods (ALP): the sending application does not provide enough data to fully utilize available network resources.
- Bulk Transfer Periods (BTP): the sender produces enough data to continuously send data and fully utilize available network or receiver resources.
- Short Transfer Periods (STP): short bulk transfers that contain less than 130 packets and are mainly dominated by congestion control.

First, the I&M algorithm isolates ALPs, BTPs, and STPs from each other. Therefore, the Isolate procedure processes all packets of a connection according to their order. Initially, the algorithm starts to account packets to an ALP. If three consecutive data packets as large as the Maximum Segment Size (MSS) are observed, the Isolate procedure starts a BTP. Further packets are assigned to such BTP as long as two conditions are satisfied: a) the share of packets smaller than the MSS within the last ten packets is not larger than a predefined threshold value (th) and b) the inter-arrival time (IAT) between two successive packets is not larger than $\frac{RTT}{2}$. If such conditions are no longer satisfied, the Isolate procedure assigns packets to an ALP and terminates the current BTP. After isolating transfer periods, the Merge procedure is purposed to merge ALPs with adjacent BTPs as far as the impact of the ALP is insignificant regarding the periods' average throughput. An ALP gets merged to the adjacent transfer periods if merging the periods does not result in a significantly lower average throughput of the new period than the average throughput of each BTP before merging. Merging relies on a predefined threshold value ($drop$) that decides whether the throughput of a merged period is significantly lower or not.

C. Change Point Detection

Change point detection (CPD) is purposed to detect changes in a signal, respectively, in a sequence of values. Detected changes are referred to as change points (CP). CPD is applied in different contexts considering different characteristics of the applied method. While some approaches require the number of expected changes within a signal before the estimation, other methods do not require such information beforehand. A further differentiation exists between online and offline detection. Online CPD aims to provide fast detection of changes, while offline CPD allows processing the signal without constraints regarding the execution time. According to Truong et al. [11] a change point detection method is defined by a cost function, a search method, and different constraints depending on the method. The cost function is purposed to assess the homogeneity of a signal. Therefore, it decides how likely a change is to occur in a given signal. The search method determines the processing of the signal to detect changes.

III. Approach

First, this section describes the classification of changes in TCP throughput based on transfer periods. Second, this section introduces metrics used to evaluate the suitability of CPD methods for changes in TCP throughput.

A. Classifying TCP Throughput Changes

In this paper, we pursue whether transfer periods are suitable for classifying changes in the throughput of a TCP connection.

We define four types of change points according to their relation to transfer periods as determined by the I&M algorithm: The first type is referred to as a *matching* change point, defined as CPs that match a border between transfer periods. If a connection changes from application limitation to bulk transfer, it is expected that the connection's throughput will increase. Another way around, we expect throughput to decrease significantly when a bulk transfer ends and an ALP begins. Next to matching change points, there might be changes that do not coincide with borders of transfer periods. We refer to such CPs as *unmatched*. Unmatched CPs are subdivided according to the transfer period surrounding the CP. Correspondingly, the four types of CPs are matched, unmatched in an ALP, unmatched in a BTP, and unmatched in an STP. Figure 1 shows an example of the throughput of a TCP connection over time. In this example, the flow includes two BTPs separated by an ALP. The changes between the transfer periods imply throughput changes that are marked as change points CP_1 and CP_4. During the ALP, throughput changes again. The corresponding unmatched change points CP_2 and CP_3 can be traced back to the application limited behavior of the sending application. There are expected changes in throughput when an application switches between sending small packets and completely idling during an ALP. As illustrated, the throughput during the second BTP increases resulting in CP_5 and CP_6, for example, as there is temporarily more available bandwidth on the bottleneck link.

B. Evaluating Change Point Detection Methods

To the best of our knowledge, the application of CPD on throughput changes of TCP connections has not been studied so far. Therefore, we evaluate the suitability of different change point detection methods for our problem. The metrics used for evaluating CPD methods are described in this section. In order to evaluate different CPD methods, we require the generation of TCP connections with intended changes in throughput. The implementation of the generation process of labeled TCP connections is described in Section IV-D.

The first metric we consider is the Annotation Error [11]. The Annotation Error is defined as the difference between the number of actual changes (derived from ground truth labeling) and the detected change points. A significant Annotation Error

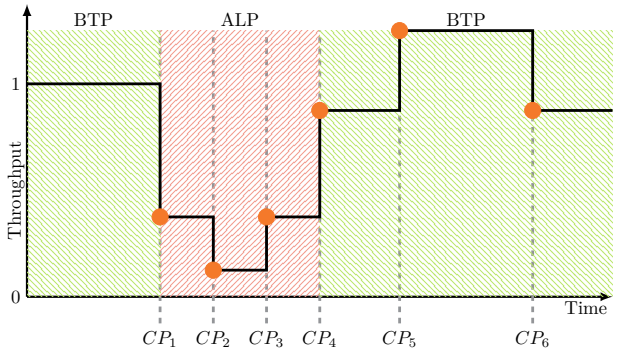


Fig. 1: Illustration of the matching between transfer periods and different types of change points.

implies a large number of undetected changes (false negatives) or a large number of detected changes without an actual change (false positives).

A further indicator for the accuracy of a CPD estimation is the F1-score according to Truong et al. [11]. The F1-score is calculated as the harmonic mean of precision and recall, which assess the impact of false negatives, respectively false positives on the detection results. Precision is calculated by the ratio of detected change points that match an actual signal change. Therefore, strong over-segmentation by the detected change points results in low precision due to many false positives. Recall is the ratio of actual change points matched by a detected change point. The definition of recall indicates that many false negatives, i.e., not detected changes, result in low values. Whether an actual change and an estimated CP are referred to as matched depends on the error margin, also referred to as tolerance interval, used for the score calculation. As the harmonic mean of precision and recall, the F1-score is normalized between 0 and 1, while larger F1-scores indicate better change point detection results.

IV. Implementation

We use two frameworks to implement traffic analysis and traffic generation. First, we implement a traffic analysis framework to parse captured network traffic, conduct CPD, and run the I&M algorithm. The second framework is purposed to generate TCP connections with intended throughput changes while tracking ground truth information for each connection.

A. Flow Analysis and Feature Extraction

The introduced traffic analysis framework consists of a pipeline of analysis components, each interacting with a database backend to read and write results. The first analysis step is extracting packet information from a PCAP containing captured traffic, and aggregating packets to connections, also referred to as flows in the following. The flow aggregation and feature extraction component is written in Go and relies on the *gopacket* library [12] to read packets. We extract packet timestamps, payload sizes, the MSS, TCP flags, sequence numbers, and TCP options for each packet and compose packets to a flow based on the IP5-tuple considering packets in both directions.

Further, the framework calculates a sequence of throughput for each flow. Calculating throughput is done by segmenting the flow duration in intervals of fixed size and aggregating transmitted bytes for each interval. By default, we use a resolution of throughput calculation of $100\,\mathrm{ms}$.

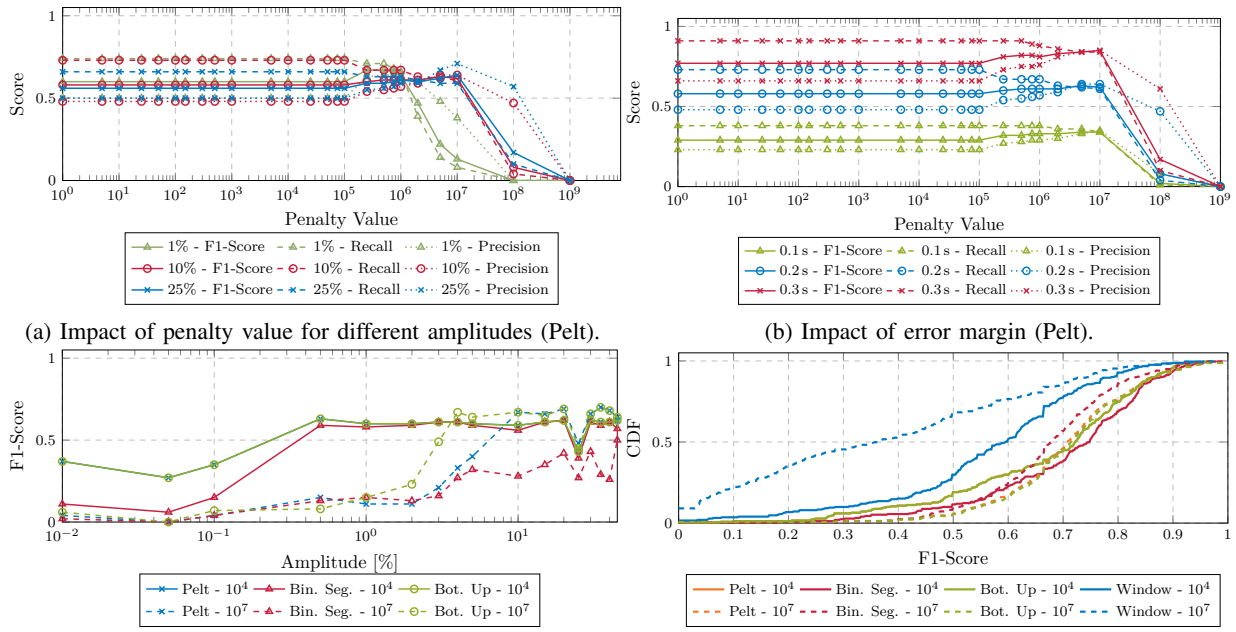
B. Change Point Detection

For the implementation of change point detection, we rely on the Python library *ruptures* introduced by Truong et al. [13].

Ruptures enables users to compose CPD methods from over eight different cost functions and four search methods applicable to our use case, i.e., offline CPD for an unknown number of changes. For our measurements, we choose the cost function that assesses the homogeneity of a signal based on the least absolute deviation. The least absolute deviation is purposed to detect shifts in the median of the analyzed signal while it is expected to provide outlier robust change point detection [16]. The considered search methods are the optimal method *Pelt* [17] and the three approximate search methods *Binary Segmentation* (Bin. Seg.) [18], *Bottom Up Segmentation* (Bott. Up.) [19], and *Window-based CPD* (Win.-based). Further, *ruptures* uses the penalty value to influence the sensitivity of applied change point detection. In the following, we refer to a CPD method as a tuple of cost function and search method. In order to estimate changes, the analysis framework reads a sequence of measured throughput for each connection and conducts change point detection. Afterward, the CPD component writes detected changes to the framework's backend. A further component, responsible for the validation of estimated change points, receives ground truth data by processing configuration files of the traffic generation framework introduced in Section IV-D. Such configuration files include the point when an intended throughout change took place. Based on the set of intended changes and the set of estimated change points, the analysis framework calculates scores to assess CPD results for generated traffic as introduced in Section III-B.

C. Transfer Period Analysis

A further analysis component takes care of identifying transport periods for each flow according to the I&M algorithm introduced by Siekkinen et al. [1] as described in Section II-B. The implemented I&M component calculates inter-arrival times (IAT) and round trip time (RTT) estimates for each connection. Further, we determine the distance between the capturing point and the sender as required for the I&M algorithm. Our framework relies on two approaches to derive RTT estimates of a flow. First, we calculate the initial RTT during the TCP Three-Way-Handshake. As the sequence of packets during the handshake is well-known, initial RTT can be estimated by the intervals between the SYN packet and the consecutive $SYN + ACK$ packet, referred to as d_1, and between the $SYN + ACK$ packet and the following ACK packet, referred to as d_2. Next to estimating the initial RTT,



(a) Impact of penalty value for different amplitudes (Pelt).

(b) Impact of error margin (Pelt).

(c) Impact of change amplitude for different penalties.

(d) Traffic with random parameters for different penalties.

Fig. 2: Measurement results of CPD method evaluation with different generated data sets.

we calculate RTT estimates based on the TCP timestamp option if a connection uses it. The timestamp option-based approach identifies triples of packets carrying matching TCP timestamp values and calculates the intervals between such packets. Afterward, we calculate the average of all estimated RTTs to be used by the algorithm. As the capturing point might not be close to the sender, Siekkinen et al. [1] propose shifting of packet timestamps in order to calculate IATs between a received ACK and a sent data packet. Shifting is based on the relative distance between the capturing point and the sender. The intervals d_1 and d_2, as introduced above, are used to estimate the relative distance between sender and receiver as $\frac{d_1}{d_1+d_2}$. Based on the extracted data, our framework determines the segmentation of each connection into different transfer periods as specified by Siekkinen et al. [1].

D. Labelled Traffic Generation

In order to evaluate different CPD methods, we require to generate TCP traffic with intended throughput changes. We rely on the TCP measurement framework introduced by Jaeger et al. [14], based on the network emulation tool Mininet [15]. The measurement framework by Jaeger et al. enables network and host parameters to be dynamically reconfigured based on user-defined schedules. According to the configured schedule, the framework orchestrates network entities to generate traffic and adapts parameters like link bandwidth, link delays, or loss. TCP traffic is generated by transmitting data with the *netcat* utility. A simple topology consisting of two hosts connected via a switch is sufficient for our traffic generation, as we only aim to modify the throughput of a specific connection. Further, the measurement framework already satisfies our requirement to dynamically change a flow's throughput, as the schedule-based configuration can configure the bandwidth of a link, which we use as an upper limit of throughput.

V. Evaluation

To evaluate different CPD methods, we conduct measurements on generated TCP traffic, including intended throughput changes. As described in Section IV we implement change point detection with the Python library *ruptures* and use the least absolute deviation as cost function. Measurements are conducted with the search methods Pelt, Bin. Seg., Bott. Up, and Win.-based CPD. We calculate the Annotation Error, F1-score, Recall, and Precision as defined in Section III-B. Our evaluation is purposed to determine how reliable changes are detected with different search methods, penalty values, and amplitudes of changes. We generate three traces for each traffic configuration to ensure results are robust to measurement artifacts. Plotted results show the average of the different measurement runs per configuration.

First, we assess the impact of different penalty values on the accuracy of conducted change point detection. We generate connections with a duration of 30 s and 30 equidistantly distributed changes, and configure change amplitudes of 1 %, 10 %, and 25 %. We observe that very high penalties imply worse accuracy depending on the amplitude of changes as CPD gets less sensitive. This can be seen in Figure 2 a) for measurements conducted with the search method Pelt, different amplitudes, and an error margin of 200 ms. We observe constant F1-scores, recall values, and precision values for smaller penalties. Further, we note that F1-scores slightly increase for specific penalties before penalties become too high and no changes are detected. This observation can be traced back to fewer false positives for larger penalties. We find a

significant impact by the error margin used for the F1-score calculation. For example, increasing the error margin from 200 ms to 300 ms increases the F1-score for measurements conducted with Pelt from 0.7 to 0.9 for most penalties as shown in Figure 2 b) for an amplitude of 10%. The same patterns are observed for other search methods. With an error margin of 300 ms we observe large recall values for Pelt, Bin. Seg., and Bottom Up. This observation indicates that nearly all intentionally generated throughput changes are detected. A significantly lower precision value implies that the CPD methods detect more changes as intended. For Win.-based CPD, we observe larger precision values than recall values. This indicates that fewer intended changes are detected, while the false positives share is significantly lower than for other search methods. However, Pelt, Bin. Seg., and Bottom Up result in significantly larger F1-scores and, therefore, overall accuracy. To assess the impact of changes' amplitude, we run measurements with varying amplitudes. Again, we generate connections of 30 seconds, apply an error margin of 200 ms, and consider several penalty values. Figure 2 c) shows F1-scores for considered amplitudes and penalties of 10^4 and 10^7 for Pelt, Bin. Seg., and Bottom Up. We observe pretty small F1-scores for all methods and very small amplitudes for both penalties. As expected, smaller penalty values enable to detect changes with smaller amplitudes. We find constant accuracy after the amplitude exceeds a certain threshold for both penalties. Such observations of the F1-scores can be explained by the according AnnotationErrors. For small amplitudes, too few changes are detected. With increasing amplitudes, the number of false negatives decreases. To further examine differences between search methods, we generate 250 connections with randomly selected amplitudes for each change, randomly selected numbers of changes per connection, and randomized intervals between two changes. The cumulative distribution of measured F1-scores for an error margin of 200 ms reveals that 50 % of measured F1-scores are larger than 0.7 for Pelt, Bin. Seg., and Bottom Up with a penalty of 10^4, as shown in Figure 2 d). Win.-based CPD shows significantly lower accuracy. The accuracy with a penalty of 10^7 is slightly smaller compared to accuracy with a penalty of 10^4 for Pelt, Bin. Seg., and Bottom Up, and significantly smaller for Window-based CPD. We find that Pelt and Bottom Up Segmentation result in the nearly identical F1-scores for applied penalties. The same observation applies to measurements discussed above.

For our analysis of Internet traffic, we conclude that Pelt, Bott. Up, and Bin. Seg. are expected to provide more accurate results than Window-based CPD. The measured recall values for such search methods imply that it is more likely to detect false positives than not detecting an intended change.

VI. Analysis of Captured Internet Traffic

To evaluate the suitability of our approach to classify changes in TCP throughput, we conduct measurements on captured Internet traffic. We compare measurement results for different CPD methods and penalty values. Note that measurements with Pelt were not feasible due to Pelt's computational complexity. To survey the results of our matching analysis between period changes (PC) and change points (CP), we calculate two shares: We define the *CP share* as the share of detected change points matching to a period change and the *PC share* as the share of period changes that match to a detected change point. The *CP share* represents the share of matched change points and also implies the share of unmatched change points as defined in Section III. Therefore, a large CP share indicates that we are able to classify a large share of detected change points as matching to a period change. The *PC share* indicates how successful period changes were matched to detected change points and whether period changes do not coincide with an estimated throughput change.

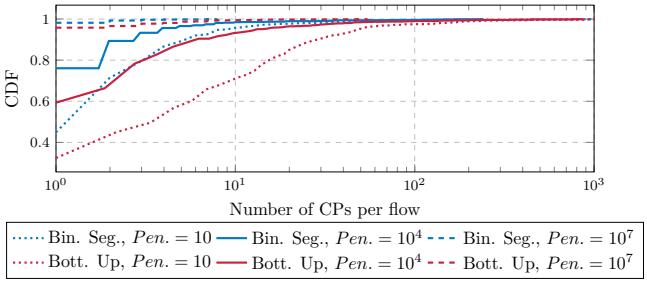


Fig. 3: Number of CPs per connection.

For our study, we analyze a traffic capture provided by the Measurement and Analysis on the WIDE Internet (MAWI) Working Group [20]. The trace taken on the 31st of July 2021 includes 15 minutes of Internet traffic captured at MAWI's Samplepoint-F. We filter out TCP connections consisting of less than 130 packets, which is the minimum packet count for an STP, as such connections are not relevant for our analysis of matching between CPs and PCs. Further, we only consider IPv4 traffic. Next to packet count and IPv4, we constraint analyzed connections to those who provide a fully captured TCP Three-Way-Handshake as this is required for the I&M algorithm. In total, we find 1649 connections that fulfill all analysis requirements in the analyzed traffic capture. We choose a matching margin of 300 ms. This means that we refer to a change point as matched if the difference between the timestamp of the change point and the nearest period change is smaller 300 ms. We calculate throughput in intervals of 100 ms. For measurements with the I&M algorithm, we choose a *drop* value of 0.9 and a *th* value of 3 as proposed by Siekkinen et al. [1].

To assess the distribution of change points on connections, we conduct measurements with different change point detection configurations and calculate the cumulative distribution function (CDF) of the number of change points per connection as shown in Figure 3. We observe that different penalty values result in significantly different numbers of detected change points per flow for both search methods. Further, we find that the number of changes decreases significantly with increasing penalty values. Bottom Up segmentation results in significantly more detected change points than Binary Segmentation. Independent of the search method and the penalty, we find

TABLE I: Total number and share of matched period changes (PC) and change point (CP) types.

Method	Penalty	Total CPs	Matched CPs	Unmatched	in BTP	in ALP	in STP	Matched PCs
Bin. Seg.	10^1	4502	683 (15.17 %)	3819 (84.83 %)	3491 (77.54 %)	286 (6.35 %)	42 (0.93 %)	31.41 %
	10^4	2065	446 (21.60 %)	1619 (78.40 %)	1527 (73.95 %)	69 (3.34 %)	23 (1.11 %)	20.51 %
	10^7	510	286 (56.08 %)	224 (43.92 %)	209 (40.98 %)	0 (0.0 %)	15 (2.94 %)	12.32 %
Bottom Up	10^1	8991	1195 (13.29 %)	7796 (86.71 %)	6128 (68.16 %)	1562 (17.37 %)	106 (1.18 %)	54.97 %
	10^4	5578	878 (15.74 %)	4700 (84.26 %)	4071 (72.98 %)	567 (10.16 %)	62 (1.11 %)	40.39 %
	10^7	819	301 (36.75 %)	518 (63.25 %)	502 (61.29 %)	1 (0.12 %)	15 (1.84 %)	13.85 %

that the vast majority of connections include less than 10 estimated change points while observing a very long tail of detected change points per flow. Within the 1649 analyzed connections, we find 1100 connections that only consist of one single ALP. We remove such connections from our data set as they are not relevant for further analysis of throughput change classification. The I&M algorithm detects 2174 period changes for the considered TCP connections.

Regarding the results of our matching analysis between CPs and PCs, as shown in Table I, we observe that the share of matched change points increases with increasing penalties. At the same time, the absolute number of matched CPs decreases, such as the total number of detected CPs. In contrast, the share of matched period changes increases with decreasing penalty. This pattern can be explained by fewer change points for larger penalties and more change points for smaller penalties. For changes detected by Bin. Seg. and a penalty of 10^7 we observe a CP share of 56 %, i.e., a period change can explain 56 % of estimated change points. However, at the same time, the mentioned CPD configuration only has a PC share of 12.3 %. This indicates that over 85 % of period changes are not matched by a change point. With Bott. Up segmentation and a penalty of 10^4, we observe a PC share of over 40 %, respectively 54.9 % for a penalty of 10. Such large PC shares can be explained by a larger number of detected change points, also implying large numbers of unmatched changes. We conclude that the penalty value can be used to either increase the CP share at the cost of a worse PC share or vice versa. An exemplary analysis of measured shares for Bottom Up segmentation reveals that nearly 40 % of flows show an PC share equal to 1 with a penalty value of 10. At the same time CP shares for the same penalty show large shares of unmatched change points. Contrary to such observation, a penalty value of 10^7 results in significantly larger CP shares than PC shares. This implies a trade-off between increasing CP and PC shares depending on the penalty value.

VII. Related Work

The application of change point detection in network traffic analysis has been studied before. For example, change point detection is considered to detect anomalies and network intrusion in real-time with a focus on efficient and fast detection of changes in traffic parameters, as for instance caused by denial of service attacks [21]–[23]. Besides its application to anomaly detection, change point detection has been considered to optimize TCP congestion control in combination with the application of deep learning by Li et al. [24].

Next to different applications of change point detection our work is closely related to the field of TCP throughput limitation analysis, also referred to as TCP root cause analysis. Starting with Zhang et al. [10], researchers presented different approaches to detect different limitations of TCP throughput [1], [7]–[9]. While we evaluated the classification of throughput changes with transfer period changes, further throughput limitations could be considered for the classification of further changes. Thereby, changes occurring during bulk transfer periods are of major interest as the question arises whether the root cause of throughput limitation changes when significant changes in throughput are observed. In previous research, we introduced an approach to the online monitoring of TCP throughput limitations [26], [27] which also could be complemented with online change point detection in future work.

VIII. Conclusion

This paper introduces an approach to classify TCP throughput changes with TCP transfer periods. We evaluate change point detection methods against ground truth data and conduct measurements on captured Internet traffic. We find that, depending on the change point detection configuration, over 50 % of detected changes can be matched to a change between transfer periods. High rates of matching change points coincide with low shares of matched period changes, i.e., not all period changes can be matched to an estimated change of throughput. Our measurements also reveal large numbers of unmatched change points that mainly occur during bulk transfers.

Our observations motivate further studies on the coincides between throughput changes and transfer periods, like the optimization of CPD configuration. Throughput changes within bulk transfers are considered to be of interest for further analysis regarding their coincide with changes in the throughput limitation of a connection. Such refinement requires the extension of our approach with further concepts from throughput limitation analysis.

Acknowledgments

This work was supported in part by the German Research Foundation, project ModANet (CA595/11-1) and by the German Federal Ministry of Education and Research, projects PRIMEnet and 6G-live.

References

[1] 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*, ITC20'07, (Berlin, Heidelberg), Springer-Verlag, 2007.

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

[3] A. Gurtov, T. Henderson, S. Floyd, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm." RFC 6582, Apr. 2012.

[4] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[5] 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*, (Zurich, Switzerland), May 2018.

[6] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[7] M. Siekkinen, G. Urvoy-Keller, E. W. Biersack, and D. Collange, "A root cause analysis toolkit for tcp," *Comput. Netw.*, vol. 52, June 2008.

[8] M. Timmer, P.-T. de Boer, and A. Pras, "How to identify the speed limiting factor of a tcp flow," in *2006 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services*, pp. 17–24, IEEE, 2006.

[9] A. Bak, P. Gajowniczek, and M. Zagozdzon, "Measurement methodology of tcp performance bottlenecks," in *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 1149–1156, IEEE, 2015.

[10] 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*, SIGCOMM '02, (New York, NY, USA), pp. 309–322, ACM, 2002.

[11] C. Truong, L. Oudre, and N. Vayatis, "Selective review of offline change point detection methods," *Signal Processing*, vol. 167, p. 107299, 2020.

[12] "google/gopacket: Provides packet processing capabilities for go."

[13] C. Truong, L. Oudre, and N. Vayatis, "ruptures: change point detection in python," *arXiv preprint arXiv:1801.00826*, 2018.

[14] B. Jaeger, D. Scholz, D. Raumer, F. Geyer, and G. Carle, "Reproducible Measurements of TCP BBR Congestion Control," *Computer Communications*, vol. 144, pp. 31–43, May 2019.

[15] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, pp. 139–42, 2014.

[16] J. Bai, "Least absolute deviation estimation of a shift," *Econometric Theory*, vol. 11, no. 3, pp. 403–436, 1995.

[17] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal detection of changepoints with a linear computational cost," *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1590–1598, 2012.

[18] P. Fryzlewicz, "Wild binary segmentation for multiple change-point detection," *The Annals of Statistics*, vol. 42, no. 6, pp. 2243–2281, 2014.

[19] P. Fryzlewicz, "Unbalanced haar technique for nonparametric function estimation," *Journal of the American Statistical Association*, vol. 102, no. 480, pp. 1318–1327, 2007.

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

[21] A. G. Tartakovsky, A. S. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, 2012.

[22] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blazek, and H. Kim, "A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods," *IEEE transactions on signal processing*, vol. 54, no. 9, pp. 3372–3382, 2006.

[23] M. Alkasassbeh, "A novel hybrid method for network anomaly detection based on traffic prediction and change point detection," *arXiv preprint arXiv:1801.05309*, 2018.

[24] W. Li, S. Gao, X. Li, Y. Xu, and S. Lu, "Tcp-neuroc: Neural adaptive tcp congestion control with online changepoint detection," *IEEE Journal on Selected Areas in Communications*, 2021.

[25] T. Iwata, K. Nakamura, Y. Tokusashi, and H. Matsutani, "Accelerating online change-point detection algorithm using 10 gbe fpga nic," in *European Conference on Parallel Processing*, pp. 506–517, Springer, 2018.

[26] S. Bauer, K. Holzinger, B. Jaeger, P. Emmerich, and G. Carle, "Online monitoring of tcp throughput limitations," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2020.

[27] S. Bauer, F. Wiedner, B. Jaeger, P. Emmerich, and G. Carle, "Scalable tcp throughput limitation monitoring," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 410–416, IEEE, 2021.