Troubleshooting Enhancement with Automated Slow-Start Detection

Ziad Tlaiss, Isabelle Hamchaoui, Isabel Amigo, Alexandre Ferrieux, Sandrine Vaton

Troubleshooting Enhancement with Automated Slow-Start Detection

Ziad Tlaiss[†*], Isabelle Hamchaoui[*], Isabel Amigo[†], Alexandre Ferrieux[*], Sandrine Vaton[†]
IMT Atlantique, Lab-STICC laboratory (Brest, France) [†]
Orange Labs Networks (Lannion, France)[*]
Emails: [†]{firstname.lastname}@imt-atlantique.fr, [*]{firstname.lastname}@orange.com

***Abstract*—Detecting anomalies in networks usually requires packet level traffic capturing and analysing. Indeed, the observation of emission patterns sheds some light on the kind of degradation experienced by a connection. In the case of reliable transport traffic where congestion control is performed, such as TCP and QUIC traffic, these patterns are the fruit of decisions made by the Congestion Control Algorithm (CCA), according to its own perception of network conditions. The CCA estimates the bottleneck's capacity via an exponential probing, during the so-called "Slow-Start" (SS) state. The bottleneck is considered as reached upon reception of congestion signs, typically lost packets or abnormal packet delays depending on the version of CCA used. The SS state duration is thus a key indicator for the diagnosis of faults; this indicator is estimated empirically by human experts today, which is time-consuming and a cumbersome task with large error margins.**

This paper proposes a method to automatically identify the Slow-Start state from actively and passively obtained bidirectional packet traces. It relies on an innovative timeless representation of the observed packets series. We implemented our method in our active and passive probes and tested it with CUBIC and BBR under different network conditions. We then picked a few real-life examples to illustrate the value of our representation for easy discrimination between typical faults.

Index Terms: troubleshooting, active measurement, passive measurement, Congestion Control algorithm, Slow-Start

I. Introduction

Quality of Experience (QoE) remains one of the most crucial competitive advantages for an Internet Service Provider as it directly impacts its brand image. However, room for improvement remains consistent for network operators as many networks are still impeded by crippling issues bitterly reported by customers and commented by media which affect operators reputation. Improving customers experience is a necessary yet delicate task, as it relies on continuous and pervasive monitoring of the network and, as soon as a degradation is detected, a quick identification and fixing of the root cause is demanded, that is troubleshooting.

Unfortunately many problems cannot be handled with mainstream monitoring tools: For example, excessive latency cannot be detected via flow-level tools such as Netflow [1], nor by equipment's counters [2] or traffic sampling methods [3]. Even if end-to-end active measurements can identify slow connections, they are of little help for locating latency bottlenecks on the path. Collecting and analysing exhaustive packet-level traces captured on network mid-points are still operators' best choice for anomaly root cause identification.

For decades, most operators' end-to-end diagnosis methods have been based on the observation of transport protocol (e.g. TCP, Transmission Control Protocol) packet headers. Hence, with a mere one-point traffic capture, passive probes can easily catch latencies and packet losses on both upstream (from sender to probe) and downstream (from probe to destination) segments, see [4] and [5]. Faulty nodes can further be located by moving the capture point (beam splitter, port mirroring, etc.). These captures together with the related analysis are typically performed by human experts upon network monitoring alarms or customers claims. This method remains the cornerstone of troubleshooting for many operators, however, it is jeopardized by the explosive growth of QUIC where transport headers are encrypted, making them unreadable for a midpoint observer. Although QUIC has an option which allows to monitor latencies [6], it is likely that it will not be implemented by client softwares (e.g. browsers, mobile applications) in the future. QUIC observability via passive probes will thus probably be lacking for a long time.

It is true that active probes, by generating their own traffic, are not impeded by encryption and can detect QoS degradation, however, contrarily to passive probes, their representativeness can be questioned for two reasons: First, test traffic is not real clients' traffic. Second, active probes typically see only a subset of the network. This last point can be balanced through a massive probes deployment, but as current troubleshooting is mainly based on human diagnosis, automation is certainly a key element for dealing with the data deluge collected via such a dense fleet of active probes. Beyond this scalability issue, the whole troubleshooting process should be revisited in the light of active measurement specificity. Indeed, even if active end-to-end measurements easily report QoS issues, they give no hints on their location. In this context, observing the source behaviour appears as a promising strategy: Source emission patterns derive from decisions of the Congestion Control Algorithm (CCA) embedded in TCP/QUIC stacks, and reflect network conditions rather accurately. The CCA is in charge of regulating the source emission to obtain a good throughput without flooding the network. It calms down as soon as it detects early signs of congestion, such as packet loss or delay. Tracking its behaviour thus reveals crucial hints for fault qualification and location.

Slow-Start (SS) is an important state of CCA as it aims at giving a first estimation of the path capacity at the beginning of

the connection life, or on resumption after a significant pause. Should it fail, then the connection will experience poor quality. Most importantly, the way it fails (with or without loss, etc.) is an excellent indication of the degradation root cause. In this work, we introduce an effective method to automatically detect the exit from SS state - or equivalent naming. For this purpose, we introduce a novel representation, that is the bytes-in-flight versus sequence number. We use then this representation in order to identify the last packet in the SS state of the CCA, by using a relation between the sequence number values and the bytes-in-flight values that is true only during SS phase. Due to the similarity of QUIC and TCP CCAs, this method applies to both. However, as it requires accessing transport headers information, it can be applied to QUIC traffic **only** with active measurements while it could be used on TCP traffic with both active and passive measurements. We also introduce how our SS detection method could be used as a powerful tool to easily discriminate between network typical fault types.

The remainder of this paper is organized as follows. In Section II, we demonstrate the significance of CCA behaviour for troubleshooting. In Section III, we describe the troubleshooting process from data collection to data analysis for root cause identification. Section IV presents our method to detect SS state exit. An evaluation of our method is presented in section V. An application of the method to network troubleshooting is presented in section VI. Section VII surveys the related work. Finally, a conclusion is given in Section VIII.

II. Congestion Control Algorithm and troubleshooting

A. Finite State Machine transitions series

The CCA role is to prevent congestion collapse while taking into account fairness and improving connection's performance [7]. Roughly, the CCA embedded in TCP/QUIC stacks aims at reaching, in a fair manner, the highest throughput safely tolerable by the network. Under its control, the emitted traffic falls back as soon as it detects signs of congestion, that is, lost packets or growing delays. This behaviour can be observed with an exhaustive capture of the flow's packets, but also, via the sequence of transitions of the CCA Finite State Machine (FSM). State transitions are typically triggered by degradation events such as detection of congestion signals. The FSM transitions series contains factually rich semantic information providing crucial elements for troubleshooting.

B. Invariants in CCA behaviour

Amongst the CCAs, the most commonly encountered are Cubic and BBR (Bottleneck Bandwidth and Round-trip propagation time)[8]. According to [9], they alone contribute to the vast majority of today's traffic, in particular, BBR traffic probably represents more than half of all internet traffic. Other CCAs can be observed, but in significantly lower proportion. However, the CCA landscape remains diverse as many flavours coexist for a given CCA, depending of the underlying implementation (e.g. Linux version on server side).

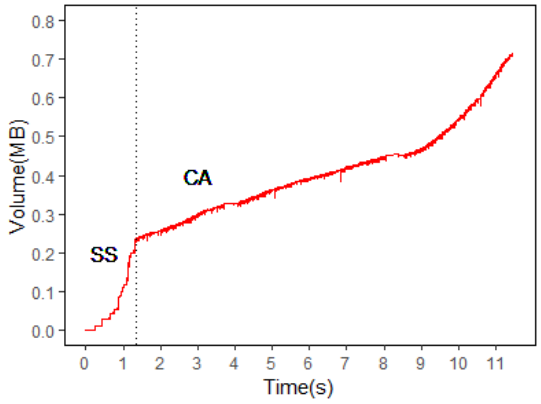

Figure 1: BIF over time evolution for a CUBIC capture showing SS & CA states, we can notice the exponential growth of the SS phase versus the linear growth during the CA phase

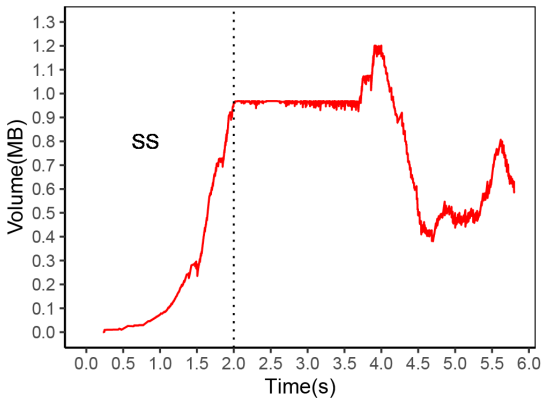

Figure 2: BIF over time evolution for a BBR capture showing SS state, we can notice the exponential growth of the SS phase versus the linear growth after

Even if these CCAs differ in major ways, they share a few common mechanisms, particularly at the beginning of the connection life. Should it be called "Slow Start" or whatever equivalent (e.g. Hystart or Hystart++, see [10]), the CCA behaviour in the first state is the same: an exponential rate growth until reaching the bottleneck capacity. An example of this exponential growth is shown in Figure 1 and Figure 2 for a CUBIC and BBR capture.

C. The (not so) Slow Start

Generally speaking, the CCA controls the amount of data being injected into the network. For this purpose, the sender typically maintains a variable called Congestion Window (cwnd) that determines the amount of data that can be transmitted before receiving an acknowledgement from the destination [11]. This cwnd is a variable internal to the sender stack and then is unknown from any mid-point observer, or from the destination. However, it can be inferred from observation of the Bytes-in-Flight (BIF), i.e. the number of bytes sent but

not yet acknowledged, as the BIF is, by definition, strictly bounded by the cwnd value at any given time.

During the SS, the CCA exponentially increases its Congestion Window (or similarly, its rate) to quickly reach the bottleneck capacity. The very first packets are emitted with respect to a so-called Initial Congestion Window, typically 10 packets [12]. Then, the sender keeps doubling the congestion window value every round trip time until a congestion signal is detected or a threshold (ssthresh) is reached [13]. This congestion signal is typically a loss for Cubic or an excessive delay for BBR [14]. Figure 1 shows the cwnd exponential growth during the SS state ending at around 1.4 sec before entering the Congestion Avoidance (CA) state.

D. Slow-Start exit time

When analysing a packet trace, the SS exit time is a key element for troubleshooting experts. If many state transitions suggest specific root causes, SS exit time has a particular significance. Indeed, in SS, the source estimates the value of the path's capacity by exponentially increasing its rate (binary search) until a congestion signal is received, then it exits the SS state to enter a new phase with a much lower rate growth.

Should the SS overestimate the bottleneck, then the source will exceed the bottleneck capacity and thus experience multiple packet losses, from which recovery is painful. On the contrary, if it underestimates the bottleneck capacity and triggers an early SS exit, the source will under-use the available bandwidth and possibly experience a poor throughput. While bottleneck overestimation is quite unusual, underestimation is a very common cause for low performance. It simply reveals that the SS has mistakenly detected a congestion signal. This is typically the case in presence of transmission loss, or excessive jitter related to radio mobile access - even underloaded. In both cases, limiting the rate will lead to a pitiful customer experience, without any benefit regarding a non-existent congestion. For example, as shown in Figure 1, the CCA exits SS after around 1.4 sec with a cwnd value around 0.2 MB, however, we can notice that the true bottleneck capacity is around 0.7 MB as the cwnd value keeps slowly increasing until reaching it. As the trigger signal is not obvious, this early exit should be investigated.

III. Troubleshooting: from data extraction to data analysis

In this section we present the metrics used for network troubleshooting, together with associated basic data processing.

A. Data collection

Capturing Packet traces can be made via many tools, such as Wireshark [15] and tcpdump [16]. These tools capture transport layer packet headers together with their arrival times.

B. Data processing

These captures should be processed, so as to derive significant timestamped indicators; The most significant ones are:

- Sequence number: it identifies the first data byte in a segment [17]. In the rest of this article, we denote by SEQ the *last* byte of the transmitted segment, that is, $sequence\,number + length$, which is better matched to acknowledgements.
- Acknowledgment (ACK): it represents the sequence number of the last byte received.
- Receiving window (RWIN) : it identifies the number of bytes that the receiver can accept.
- Bytes in flight (BIF): it represents the number of bytes sent by the source but not yet acknowledged. BIF is not included in packets header but can be deduced from SEQ and ACK values as shown in Figure 3.
- Round-Trip-Time (RTT): it represents the delay between the emission of a packet and the reception of the corresponding acknowledgment. It can be calculated using the SEQ and ACK arrival times as shown in Figure 3.

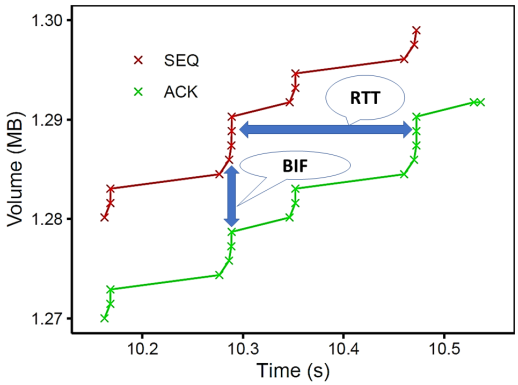

Figure 3: BIF and RTT calculation method

C. Data analysis

This final step typically consists in visually analyzing the temporal evolution of these indicators, e.g. with a tool as tcptrace [18]. For each trace, a human expert should visually detect the state transitions and QoS degradations affecting the connection.

To automate the analysis, we designed a method and developed a tool to automatically detect the SS exit time on collected traces. Combined with other indicators (presence of loss, etc.), this is a significant step toward full automation.

IV. Automatic Slow Start exit detection

In this section we introduce a new representation together with a method to automatically detect SS exit.[1]

A. Visual CCA states identification

As explained in section II, the FSM state series, and particularly the SS exit time gives crucial insight about degradation root causes. To get hold of these state series, the first idea that comes to mind is direct introspection in the sender stack. Unfortunately, this introspection requires cooperation from the sender's server, which is rather impractical, as many servers belong to third-party internet content providers, often reluctant

[1]Tool available online at https://193.252.113.227/cgi-bin/ats.cgi

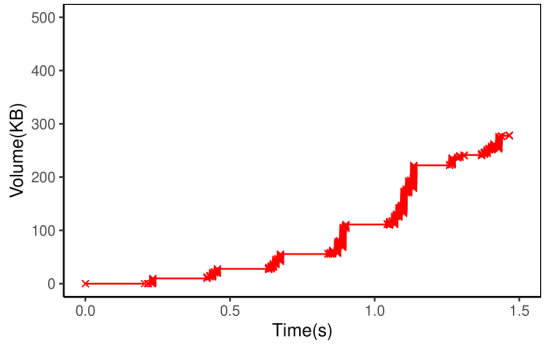

Figure 4: BIF against time during SS state, focus on the exponential growth

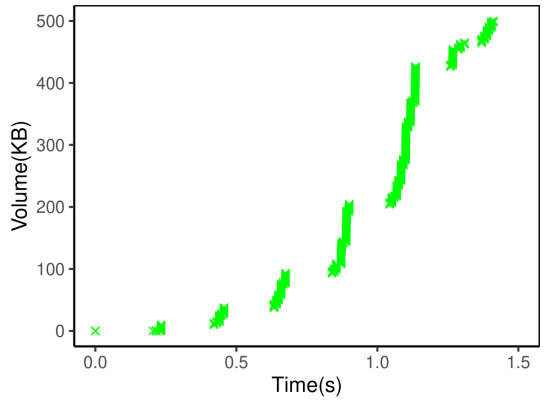

Figure 5: SEQ against time during SS state - packet burst doubled after each RTT

to open their infrastructures. As a consequence, sticking to measurements from active and passive probes is still operators' best choice to build these state series. Recall that passive probes can only handle TCP traffic, as active probes may monitor both TCP or QUIC flows.

In this context, troubleshooting experts are used to performing visual analysis of the BIF against time to detect the end of the exponential growth, namely the SS exit time. This method is highly time consuming and inaccurate. We can see in Figure 4 an exponential increase of the BIF against time until $t = 1.2$ sec, a telltale sign of the SS phase. Root cause analysis is then completed thanks to the SEQ against time graph (Figure 5) showing bursts of packets retransmission at this very same time, a typical effect of congestion loss [19]. The SS exit is then a legitimate reaction of the CCA to reaching the actual bottleneck.

B. Challenges towards automation

1) Noise: While manual analysis can easily handle noise and irregularities in the data, this is non trivial for an automated procedure. For example, some smoothing may be necessary in order to recognize the exponential growth in Figures 4 and 5. More generally, the graphs might need to

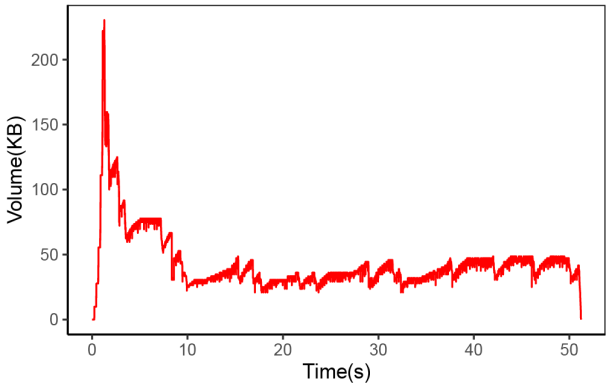

Figure 6: BIF against time - non stationarity and noise on BIF values

be segmented into a number of homogeneous regimes before any kind of pattern recognition can be applied.

2) Non-stationarity: The aforementioned time series are typically non-stationary, and not even piecewise stationary. This makes it impossible to define homogeneous areas, which invalidates most usual mathematical methods. An example about this non-stationarity can be observed in Figure 6 and Figure 7. Figure 6 shows the evolution of BIF over time, while Figure 7 represents the arrival times of the captured packets. The latter focuses on the so-called "on/off pattern" of packets arrival times which reflects the basic congestion window mechanism, waiting for ACKs before sending a new burst of packets. However, while this on/off pattern can be detected at the beginning of the connection (from 0 sec to 0.8 sec), it blurs over time, due to TCP's (intentional) tendency towards "ACK clocking" [11].

All previously mentioned challenges invalidate most methods like using regression and Markov Modulated Poisson Process (MMPP) that we have considered and tried when working on the automated detection of the SS state. A more promising approach yielding better initial results was "exponential regression", i.e. fitting the BIF-against-time with an exponential. However, it turns out that in case of *very* early SS exits (within 2 or 3 RTT), the exponential part is dwarfed by the subsequent evolution, making it impossible to detect the exponential part reliably. This is unfortunate, as we use the SS state detection to troubleshoot networks, where the most frequent cases of bad performance are correlated with a premature exit from SS. As it turns out, this fundamental problem is resolved using the new representation that we introduce in the next section.

C. Timeless packet series representation

In the light of our troubleshooting experience, it turns out that the main hurdle to automation lies in the on/off patterns of the source emissions. A natural way to get rid of them without any loss of information is to switch to a *timeless* representation. To this effect, we chose to represent BIF as a function of SEQ as shown in Figure 8. In essence, we

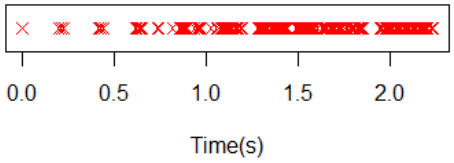

Figure 7: Packets arrival times - blurring of on/off pattern over time

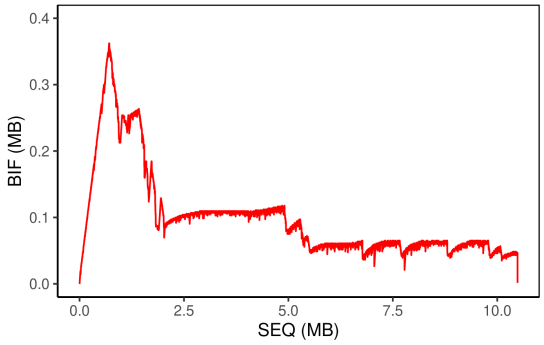

Figure 8: BIF against SEQ: new representation to detect the SS state exit time

replace the time axis with the sequence number progression: this naturally wipes out all burst, silence, or RTT variation effects, while preserving the important correlations between significant indicators, thus focusing on the CCA dynamics. To the best of our knowledge, such a representation was not described before in the state of the art.

D. Slow Start exit time: "slope 1/2" method

A few basic properties of the BIF vs SEQ representation can easily be derived analytically. To begin with, the shape of the graph is readily predictable during 2 phases:

(a) During burst emissions: in the absence of any acknowledgement, during this phase each sent packet increments both SEQ and BIF by an equal value, which is the segment's length.

(b) During the reception of burst acknowledgements: assuming all packets previously sent were received, the BIF quickly drops back to zero.

As a result of these 2 phases, every round-trip time, the graph is expected to display a triangular shape made of a slope 1 due to phase (a), followed by the vertical drop described in (b), as depicted in Figure 9. Furthermore, in an ideal SS state, the vertical extent of this triangular shape, which represents the cwnd, is expected to double every round-trip-time. Thus, the graph should display a *fractal* series of triangles, each one being twice the size of the one before. The position of the highest-SEQ point and highest-BIF point in the graph, after n round-trip-times, is thus expected to be:

$$SEQ = \sum_{i=0}^{n-1} a \times 2^i = a \times (2^n - 1)$$

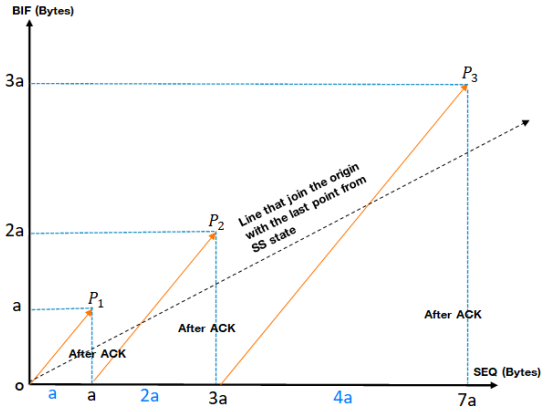

Figure 9: Theoretical Representation of the BIF vs SEQ evolution

$$BIF = a \times 2^{n-1}$$

The slope of the line from origin to this point is thus

$$BIF/SEQ = \frac{2^{n-1}}{2^n - 1}$$

And hence its limit

$$\lim_{n \to +\infty} BIF/SEQ = 1/2$$

It can further be seen that this asymptote $y = x/2$ is in fact "approached from above", as the top of each triangle satisfies.

$$BIF/SEQ = \frac{2^{n-1}}{2^n - 1} > 1/2$$

However, as soon as the SS state is exited, the exponential growth of the BIF stops, and no further point can stand above the $y = x/2$ line. This yields a very simple and practical criterion: the SS exit occurs immediately after the last point satisfying

$$BIF \geq \frac{SEQ}{2}$$

It should be stressed that the power of this method lies in its simplicity: no regression neither filtering are needed, a simple linear inequality suffices, once we are in the appropriate representation space.

E. Slope $\frac{1}{2}$ method details

While the critical state transition event is well characterized by the above criterion, some attention is due to properly interpret the earlier features of the representation. During the SS phase, as mentioned before, local slopes are typically 1, with a series of abrupt drops. As a result, the graph keeps crossing the asymptote, thus, a local decision is not appropriate, as it would readily generate false positives. Fortunately, the *global* criterion of the "last point above the asymptote" is more robust. This is fundamentally linked to the fact that after exiting SS, the CCA essentially takes very careful steps to refrain from going too fast, and by definition will never "catch up" to the exponential regime. The asymptote is never

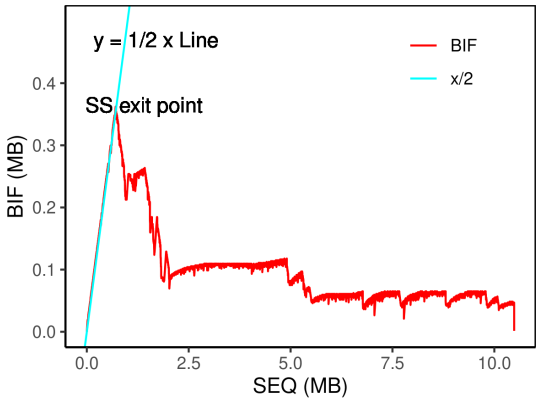

Figure 10: Automatically detecting SS state exit time with slope 1/2 method

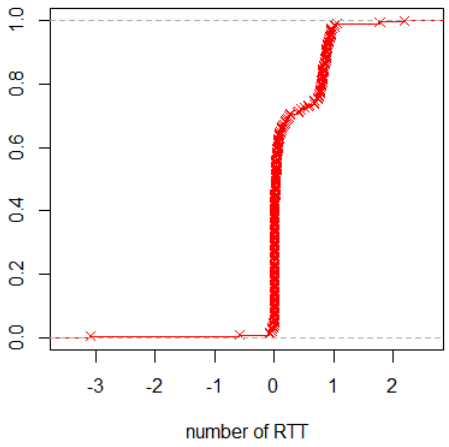

Figure 11: Cumulative distribution function of the difference in RTT units between the SS exit times from the slope 1/2 method and the one logged by the CUBIC server

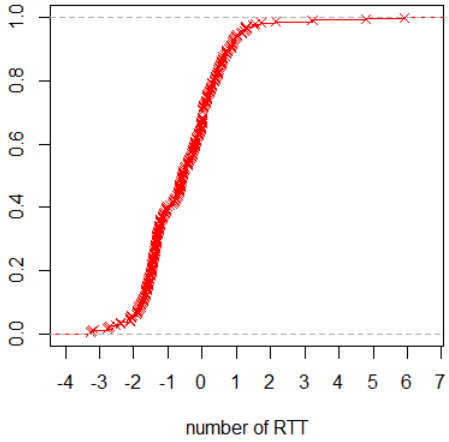

Figure 12: Cumulative distribution function of the difference in RTT units between the SS exit times from the slope 1/2 method and the one logged by the BBR server

to be crossed again. Figure 10 is an example of our slope 1/2 method application. We can see the BIF vs SEQ curve slightly exceeding the $y = \frac{1}{2}x$ line until it abruptly drifts below, marking the instant when the CCA has exited the SS state.

V. Evaluation of the slope 1/2 method

In this section we assess the accuracy of our method by comparing the SS exit time we obtain, against a "ground truth" which we define as the CCA state transition time recorded in the server stack logs. For this purpose, the assessment is performed on one of our servers, accessed by several active probes, through the public internet. We instrumented the Cubic and BBR TCP stacks on this server to generate CCA logs; then we performed active measurements with our probes by executing many downloads from our server using BBR and CUBIC CCA, and from these logs, we extracted the "ground truth" SS exit times. We note that in the Linux BBR implementation that we instrumented, the SS (binary search period) is very difficult to identify, especially in presence of massive loss, as it is not a discrete state but a region in parameter space.

For the sake of representativity, we locate our probes amongst 4 Orange affiliates. Depending on the country, the average RTT ranges from 20ms to 200ms with varied loss levels. Moreover, in each of these countries, our downloads were performed with various congestion levels (peak and off-peak hours). Last, we compute the difference between the *slope 1/2* exit time and this "ground truth". This time difference is represented in Figures 11 and 12, in RTT units.

Figure 11 shows the distribution of error in prediction for the CUBIC stack on 219 downloads. We can notice that this error is less than 1 RTT in more than 95% of cases. It is indeed the best accuracy that can be expected, since the typical time granularity of CCA decisions is precisely the RTT.

The same representation for BBR is shown in Figure 12 for 241 downloads. We can notice that 55 % of cases are bounded between -1 and 1 RTT. We attribute this larger deviation in prediction error more to the approximate ground truth (because of the aforementioned difficulties in the BBR implementation) than to real mis-predictions of our slope 1/2 method. We are currently investigating a better calculation for ground truth.

VI. Application to network troubleshooting

The slope 1/2 method, together with our new BIF/SEQ representation, proves to be a powerful tool in network troubleshooting. Indeed, with this representation, typical patterns appear, each of them pointing to a type of degradation. What is more, these graphical patterns could rather easily be identified using trivial criteria, thus leading to easy classification. We describe hereafter some typical faults together with their graphical representation.

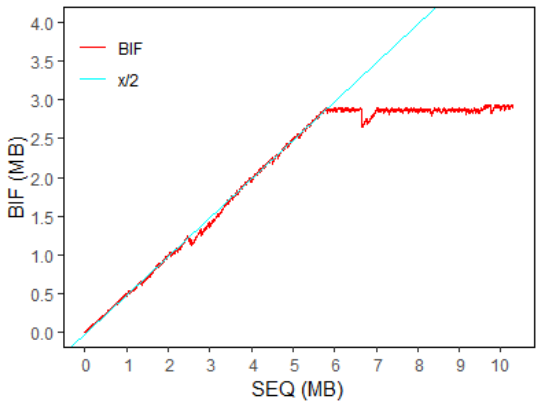

Figure 13: Constant BIF after SS exit: Good QoS

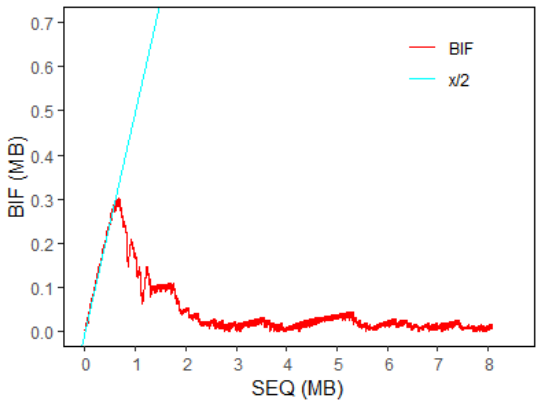

Figure 14: BIF plunge after SS exit: Loss

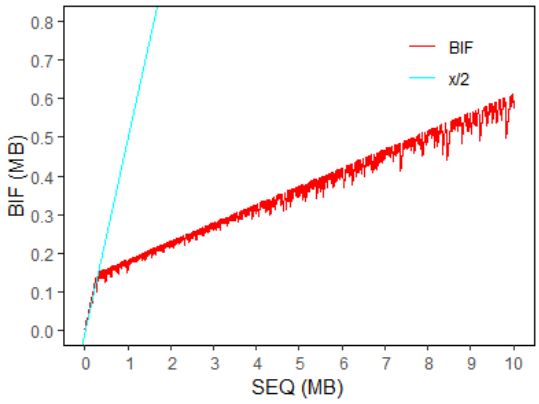

Figure 15: Slow Bif growth after SS exit: Jitter issue

A. Pattern 1: Constant BIF after SS exit

In this ideal case, the binary search performed during the SS successfully discovers the path bottleneck. Then the $BIF = f(SEQ)$ curve stays constant, as shown in figure 13. This denotes a good and stable QoS.

B. Pattern 2: BIF dive after SS exit

Figure 14 exhibits a sudden drop in BIF value after what we identify as the SS exit time; then, the BIF value further stays significantly lower than its peak at SS exit. This pattern is typically associated with QoS degradation due to loss. These losses may originate from cross-traffic competition (i.e. traffic emitted by other sources) or transmission errors.

C. Pattern 3: BIF growth after SS exit

Beyond losses, packet delay variation (also know as jitter) is another typical root cause for early SS exits [10], as they may occur while the bottleneck capacity has not been reached yet. Figure 15 illustrates this case. In contrast with pattern 1, the BIF here continues to grow after SS exit, but at a much slower pace. This allows for an easy discrimination between these two patterns. Pattern 3 also clearly differs from pattern 2, as it exhibits no reduction in BIF, that is, no packet loss.

This behaviour is typical of mobile access networks affected with large jitter values.

VII. Related Work

We identified two areas of work related to our study.

*1) **Identification of CCA states to infer TCP behavior**:* Hagos et al. [20] use machine learning approaches to recognize loss-based TCP CCAs and infer the congestion window within a passively collected traffic at mid-point. Although estimating the cwnd can be useful for network operators to troubleshoot their network, it does not cover non-loss-based CCAs such BBR. Our work differs from theirs as our method focuses on the application of CCA SS state detection in order to detect network root causes anomalies, and could be applied for all types of CCA, loss-based or not.

Jaiswal et al. [21] introduce a passive measurement methodology to infer the cwnd and round-trip-time. They build a replica of the CCA state for each TCP connection at the midpoint. This replica updates its estimate of the cwnd based on the observed acknowledgments that could change the CCA state. They use those estimates to recognize 3 of the TCP flavors: Reno, NewReno and Tahoe. Even if [21] are interested in initial cwnd, SS state and congestion avoidance states to identify the CCA types, they do not try to accurately locate state transitions.

Kato et al. [22] use unidirectional packet traces to characterise TCP CCAs. They define a new metric that is seen as being proportional to cwnd size, and apply curve fitting to recognize the CCA. In the continuity of their work Kato et al. [23] identify TCP CCAs using a sequence number vs packet arrival time representation.

Zhang et al. [24] analyse TCP passive packet captures and investigate CCA mechanisms to understand the origins of limitation in the transmission rates of flows by grouping packets into flights using a round-trip-time estimator.

Yang et al. [25] present an active TCP CCA identification tool that uses a random forest algorithm to classify the CCA flavors of a Web server.

Mishra et al. [9] developed the *Gordon* active tool, it identifies the CCA through its reaction to a tool-induced disturbance, packet loss.

In summary, [24], [25] and [9] show interest in the detection of the SS state; however, [24]'s method tracks the SS state in the first flights based on explicit segmentation, which does not work consistently in real life, e.g. when ACKs are not "bursty". On the other hand, [25] and [9] only take losses into consideration in the detection of the SS state. In contrast, in our work, while we do end up using the cause of SS exit to identify the root cause of an anomaly, we start by locating the event regardless of the cause.

*2) **Troubleshooting tools**:* Guo et al. [26] developed pingmesh, a tool for large scale data center network latency measurement and analysis to track network latency issues. Zhu et al. [27] proposed Everflow, a packet level tracing and analysis tool. While [26] and [27] are 2 troubleshooting solutions, their scope is limited to a specific set of equipment-level performance metrics; this makes sense from a "repairman"'s point of view, to whom exonerating a specific router from guilt is critical, but is not sufficient to address an end-to-end scenario, where the offending connection spans continents and (possibly non-cooperative) actors. In our work, we aim to get a broader view of the issue at hand, by providing a cause-agnostic observable, the SS exit time, as input to further investigations.

VIII. Conclusion

As many network operators use the SS state duration as a key indicator for the diagnosis of faults, it is crucial to automate its extraction to save human experts time. In this work, we have presented a method to automatically detect the exit from the Slow-Start state, enabled by an innovative timeless representation of the observed packets series. We implemented our method in our active and passive probes in 4 countries with varied access networks and traffic conditions, and tested it with both Cubic and BBR. This evaluation shows our automatic method to be accurate enough for the purpose.

As a bonus, the representation, together with the SS exit time, proves to be rather powerful for easy discrimination between typical faults. In further academic work, we plan to refine the criteria so as to identify more classes and integrate the method in an automated classifier. Our purpose is to deploy this classifier in all our probes and validate the solution at scale in a field trial. On another aspect, detecting SS state exit time could be a powerful tool to investigate the fair share of a connection between multiple TCP variants, and it can be used to identify TCP flavors.

It should be noted that this method uses information in packets headers, which is impossible with passive observation of QUIC traffic. Still, the method remains valid with active measurements. Moreover, we are currently investigating a promising generalization of the BIF=f(SEQ) representation to passive capture of encrypted traffic.

References

[1] B. Claise, "Cisco systems netflow services export version 9," *RFC 3954*, 2004.

[2] L. J. R. M. Minlan Yu, "Software defined traffic measurement with opensketch," *n NSDI*, 2013.

[3] S. P. P. Phaal and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," *RFC 3176*, 2013.

[4] K. L. Bryan Veal and D. Lowenthal, "New methods for passive estimation of tcp round-trip times."

[5] P. BenkoAndrás and V. Veres, "A passive method for estimating end-to-end tcp packet loss," *Global Telecommunications Conference*, 2021.

[6] J. Iyengar and E. M. Thomson, "Quic: A udp-based multiplexed and secure transport," *RFC9000*, May 2021.

[7] S. Floyd, "Congestion control principles," *RFC 2914*, Sep. 2000.

[8] N. Cardwell, Y. Cheng, S. H. Yeganeh, I. Swett, and V. Jacobson, "BBR Congestion Control," Internet Engineering Task Force, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Mar. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-cardwell-iccrg-bbr-congestion-control/02/

[9] A. J. S. P. R. J. A. Mishra, X. Sun and B. Leong, "The great internet tcp congestion control census," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2019.

[10] P. B. Y. H. M. Olson, "Hystart++: Modified slow start for tcp," 2020.

[11] V. P. M. Allman and E. Blanton, "Tcp congestion control," *RFC 5681*, Sep. 2009.

[12] I. K. Jan Rüth and O. Hohlfeld, "Tcp's initial window—deployment in the wild and its impact on performance," *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT*, June 2019.

[13] W. Stevens, "Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," *RFC 2001*, January 1997.

[14] N. C. Y. C. C. S. G. S. H. Y. V. JACOBSON, "Bbr congestion-based congestion control," *Communications of the ACM*, 2016.

[15] J. S. Chris Sanders. (2014) Applied network security monitoring. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/wireshark

[16] E. Casey. (2010) Handbook of digital forensics and investigation. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/tcpdump

[17] I. S. I. U. of Southern California, "Transmission control protocol," *RFC 793*, Sep 1981.

[18] Kary. Understanding the tcptrace time-sequence graph in wireshark. [Online]. Available: https://packetbomb.com/understanding-the-tcptrace-time-sequence-graph-in-wireshark/

[19] Z. Tlaiss, "Anomaly root cause diagnosis from active and passive measurement analysis," *33nd International Teletraffic Congress*, 2021.

[20] D. H. Hagos, P. E. Engelstad, A. Yazidi, and O. Kure, "General tcp state inference model from passive measurements using machine learning techniques," *IEEE Access*, vol. 6, pp. 28 372–28 387, 2018.

[21] D. C. K. J. Jaiswel S., Iannaccone G. and Towsley, "Inferring tcp connection characteristics through passive measurements," *Proc. INFOCOM 2004*, March 2004.

[22] R. Y. Toshihiko Kato, Leelianou Yongxialee and S. Ohzahata, "A study on how to characterize tcp congestion control algorithms from unidirectional packet traces," *ICIMP 2016 : The Eleventh International Conference on Internet Monitoring and Protection*, May 2016.

[23] R. Y. Toshihiko Kato, Xiaofan Yan and S. Ohzahata, "Identification of tcp congestion control algorithms from unidirectional packet traces," *the 2nd International Conference*, Nov. 2018.

[24] V. P. Yin Zhang, Lee Breslau and S. Shenker, "On the characteristics and origins of internet flow rates," *ACM SIGCOMM Computer Communication*, 2002.

[25] L. X. J. D. P. Yang, W. Luo and Y. Lu, "Tcp congestion avoidance algorithm identification," *31st International Conference on Distributed Computing Systems*, 2011.

[26] V. K. Chuanxiong Guo; Lihua Yuan; Dong Xiang; Yingnong Dang; Ray Huang; Dave Maltz; Zhaoyi Liu; Vin Wang; Bin Pang; Hua Chen; Zhi-Wei Lin, "Pingmesh: A large-scale system for data center network latency measurement and analysis," *ACM SIGCOM*, 2015.

[27] M. Z. B. Y. Z. H. Z. Yibo Zhu; Nanxi Kang; Jiaxin Cao; Albert Greenberg; Guohan Lu; Ratul Mahajan; Dave Maltz, Lihua Yuan, "Packet-level telemetry in large datacenter networks," *ACM SIGCOM*, 2015.