Online Detection of Pathological TCP Flows with Retransmissions in High-speed Networks

Eduardo Miravalls-Sierra[a], David Muelas[a], Javier Ramos[a], Jorge E. López de Vergara[a], Daniel Morató[b], Javier Aracil[a]

[a]*High Performance Computing and Networking Research Group, Dpto. Tecnología Electrónica y de las Comunicaciones, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Fco. Tomás y Valiente, 11, 28049 Madrid, Spain*

[b]*Área de Ingeniería Telemática, Dpto. Automática y Computación Universidad Pública de Navarra, Campus Arrosadia, Calle Tajonar, s/n, 31006 Pamplona, Spain*

Abstract

Online Quality of Service (QoS) assessment in high speed networks is one of the key concerns for service providers, namely to detect QoS degradation on-the-fly as soon as possible and avoid customers' complaints. In this regard, a Key Performance Indicator (KPI) is the number of TCP retransmissions per flow, which is related to packet losses or increased network and/or client/server latency. However, to accurately detect TCP retransmissions the whole sequence number list should be tracked which is a challenging task in multi-Gb/s networks.

In this paper we show that the simplest approach of counting as a retransmission a packet whose sequence number is smaller than the previous one is enough to detect pathological flows with severe retransmissions. Such a lightweight approach eliminates the need of tracking the whole TCP flow history, which severely restricts traffic analysis throughput. Our findings show that low False Positive Rates (FPR) and False Negative Rates (FNR) can be achieved in the detection of such pathological flows with severe retransmissions, which are of paramount importance for QoS monitoring. Most importantly, we show that live detection of such pathological flows at 10 Gb/s rate per processing core is feasible.

Keywords: Network Management, Performance monitoring, Quality of Service, TCP retransmissions, TCP modeling

1. Introduction

Nowadays, Quality of Service (QoS) assurance is a key differentiating aspect in the Internet Service Providers (ISPs) arena, given the strong competition among them. The performance of transport protocols plays a key role in this matter, given the impact that they exert on the application layer —specifically, in this paper we focus on TCP.

Typically, QoS assurance relies on statistics related to network flows [1]. Following RFC 7011 [2], we define a TCP flow as a set of TCP packets with a common 4-tuple, which traverse a particular capture point in the network during a bounded time interval. Thus, a TCP session is composed by two TCP flows that share the same temporal locality and the 4-tuple, swapping source and destination addresses and ports.

In this light, a Key Performance Indicator (KPI) used to know the provided quality is the amount of TCP retransmissions per flow, especially for large flows produced by file transfers or multimedia content delivery. Indeed, TCP retransmissions may cause noticeable interruptions in HTTP-based streaming, as shown in [3]. Consequently, such retransmissions jeopardize the user-perceived QoS for services that strongly depend on TCP *goodput* [4]. Furthermore, video streaming shows considerable erratic behavior with link-layer retransmissions in loaded links [5].

Hence, the detection of *pathological flows*, in terms of retransmissions, is a cornerstone that indicates possible network outages and saturation. As such, it serves network managers to counter-react against QoS degradation events. Nevertheless, such KPI must be collected in real-time, in order to trigger the corresponding alerts and swiftly perform the necessary corrective actions.

As it turns out, the real-time nature of current Network Operation Centers (NOCs) entails that a tradeoff between speed and accuracy in traffic analysis tools is in order. Generally speaking, the more accurate the analysis tools are, the more processing power they require. As the network speed is increasing steadily so is the processing power required to obtain KPIs. However, the speed of electronics is not following the pace of optics in backbone networks and, as a result, current traffic analysis tools are far from being able to handle every possible KPI

☆Manuscript received 27th June 2017. Revised 16th January 2018. Accepted 5th June 2018.

☆☆The final publication is available at Elsevier via `https://doi.org/10.1016/j.comcom.2018.06.002`, to be published in *Computer Communications*.

Email addresses: `eduardo.miravalls@uam.es` (Eduardo Miravalls-Sierra), `dav.muelas@uam.es` (David Muelas), `javier.ramos@uam.es` (Javier Ramos), `jorge.lopez_vergara@uam.es` (Jorge E. López de Vergara), `daniel.morato@unavarra.es` (Daniel Morató), `javier.aracil@uam.es` (Javier Aracil)

online. Thus, there is a tradeoff between information provided by the KPI and ease of implementation at very high speed.

However, performing live TCP retransmissions analysis on a fully TCP-utilized 10 Gb/s link is rather challenging, especially when the resources assigned to monitoring tools are constrained. The reason behind this challenge is that the TCP flow should be reconstructed on-the-fly, which implies sorting the TCP flow packets as they arrive, with millions of concurrent flows. Furthermore, some of these flows can be very large, for example in Over-the-top (OTT) video services. Thus, searching for retransmissions consumes valuable processing time. In conclusion, looking for every possible retransmission is very hard to attain at high-speed and possibly irrelevant if the number of retransmissions is small compared to the total number of packets in the flow.

Note that the packet arrival rate for a fully occupied 10 Gb/s link is about 1.6 millions of packets per second [6] if the average packet size is assumed to be 760 bytes [7]. The situation only worsens if the average packet size is smaller because the packet rate increases, which puts more stress on the analysis tool.

In this paper we present two lightweight heuristics that serve to evaluate whether a TCP packet is a retransmission or not at 10 Gb/s per processing core. We assess their accuracy and performance both with analytic models and with traces captured in real production environments. Such performance per core allows to scale to tens of Gbps of total throughput in a multi-core architecture. These heuristics show very small False Positive Rates (FPR) and False Negative Rates (FNR) in the detection of pathological flows and can be integrated in online high-speed traffic sniffers for real-time QoS degradation alerting. In this light, the contributions of our work are manifold:

- We extend the discussion about the tradeoff among accuracy, performance and resource constraints. To do so, we provide heuristics for lightweight detection of TCP flows with pathological retransmissions and evaluate them with traffic traces from real-world networks in production.

- Such heuristics are also evaluated by means of an analytic model that yields the expected error based on network conditions (*e.g.*, packet loss). Thus, our experimental methodology generalizes the results to network scenarios other than specific traces.

- Finally, we also model the computing resources, mainly in terms of memory, which are necessary to run our heuristics at arbitrarily high traffic rates.

To sum up, our results pave the way for a cost-effective implementation of network analysis solutions, which can be deployed in devices with lower processing and memory capabilities.

The rest of the paper is structured as follows: first, we provide an overview of the state of the art. Then, we present both approaches to detect retransmissions, the full TCP connection reconstruction and the proposed approximate alternatives. Afterwards, we model how our proposals perform, both from a performance and resource standpoint. Finally, we present our experimental results and discussion, along with conclusions that can be drawn from this paper.

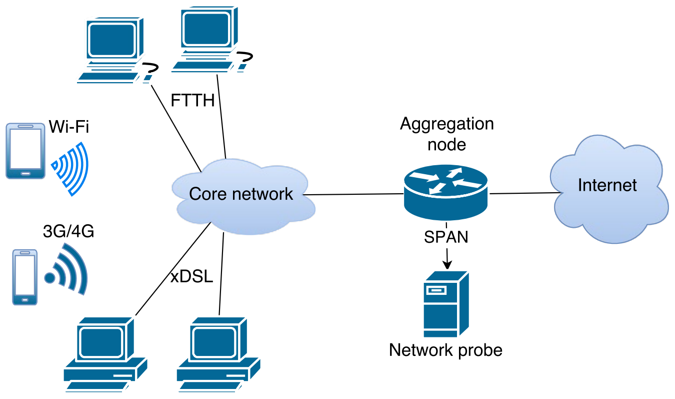


Figure 1: A typical monitoring scenario.

1.1. Problem statement and use cases

The typical traffic analysis scenario (illustrated in Figure 1) features a network probe which is attached to either a router SPAN (Switched Port ANalyzer) or mirror port, or to a network tap which acts as an aggregation node. This network probe usually consists of a traffic capture module [6], together with other software modules devoted to specific traffic analysis tasks [8, 9, 10].

Nonetheless, the former scenario is rapidly evolving with the advent of new Software Defined Networks (SDN) [11] and Network Function Virtualization (NFV) [12]. These new paradigms are encouraging the adoption of virtual ubiquitous monitoring modules that may be executed on switches and routers, maybe with limited resources. Actually, several projects are using software network probes placed in the router itself [13, 14] as a novel approach for network traffic analysis. Therefore, decreasing the processing and memory requirements to obtain KPIs becomes fundamental for the widespread adoption of such approach.

We note that retransmissions increase the duration of the flow and decrease throughput. In this light, only the flows with high number of retransmissions matter, as they are indicative of QoS degradation and, conversely, a TCP flow with a small ratio of retransmissions will not affect the perceived QoS that much.

Therefore, we propose to use the ratio of retransmissions per total number of packets in a flow as a threshold to identify possible pathological behavior. A threshold value for such a ratio is configured by the network manager, so as to obtain a right balance between sensitivity and significance. On the one hand, if the threshold value is small, a great accuracy in the measurement tool is necessary, which must be capable of detecting a small number of retransmissions per flow. Thus, keeping track of a large sequence number history in the flow becomes necessary, as counting every possible retransmission matters. Therefore, a traffic analysis tool that is highly accurate in the detection of retransmissions must have a large *memory buffer* for storing flow sequence numbers, which entails that a large data structure must be kept in memory.

On the other hand, a larger threshold value increases the significance of the findings. Actually, spurious retransmissions are common in any network and are not so much worthwhile reporting. However, a possible structural QoS degradation issue may be behind flows with large retransmission ratios. Such structural events are of interest to the network manager and provide valuable guidance to proactively fix abnormal situations.

For example, if many flows with severe retransmissions are detected between different IP addresses on both sides of a wireless link, chances are that the link is congested or suffers link layer errors.

We note that the detection of flows with severe, *i.e. pathological*, retransmissions does not require to employ a very precise retransmission detection algorithm, because counting every possible retransmission is not necessary. A less accurate algorithm suffices in this case, which has the advantage of providing a better traffic analysis throughput per processing core. Needless to say, this is beneficial for network monitoring at higher speeds or, alternatively, for network monitoring with small embedded probes with limited processing capabilities.

We show that simple retransmission detection heuristics suffice to detect flows with a retransmission ratio threshold value as small as 5% of the packets. We additionally consider that the effect of retransmissions in long-lived connections has a larger impact in QoS than in short-lived ones, for example in live video broadcasting or on-demand video services (*e.g.*, Youtube or Netflix). Furthermore, first experiments suggest that it is harder to find retransmissions in short-lived flows than in long-lived ones. Consequently, for the above mentioned 5% retransmission probability, we only consider TCP sessions with more than 100 packets aggregating both directions. The rationale behind these thresholds roots in the findings about the levels of TCP retransmissions in the Internet, and their links with TCP performance [15, 16]. That is, retransmission rates above such thresholds would surely indicate a noticeable QoS degradation.

1.2. State of the Art

Concerning analysis of TCP retransmissions, there is some previous work in the area. Aside from commercial tools, the open-source community has released a number of solutions to analyze network traffic, such as Wireshark [17] or Tstat [18, 19]. Wireshark is the *de-facto* standard in the industry to perform network analysis. It has a great variety of dissectors (more than 1350) and a very useful and intuitive graphical interface, but it was not designed for live monitoring of high speed links. Precisely, Wireshark performance is tied to the number of dissectors enabled. Furthermore, even if most of them are disabled, Wireshark performance is below multi-Gb/s. The Wireshark textual counterpart, Tshark, shows better performance due to the lack of a graphical interface and the analysis throughput is also well below multi-Gb/s rates. Both tools also suffer from large memory requirements, which limits the size of the traces that they can handle.

Tstat is a statistical traffic analyzer which outputs up to 130 metrics per TCP session depending on the configuration. Recently, the authors of Tstat released a new version [20] that can be used as statistics module which analyzes the traffic captured by a DPDK-powered network probe, achieving up to 4 interfaces of 10 Gb/s of aggregated throughput with 16 instances of Tstat. Nevertheless, this is a load balancing approach, which depends on the traffic arrival pattern. If TCP connections are concentrated on a few ports and IP addresses, such load balancing becomes unfeasible. In any case, our proposed technique is also amenable for load balancing in several instances, as we support 10 Gb/s *with a single processing thread*.

Other solutions require dedicated hardware such as FPGAs [21, 22] or GPGPUs [23] in order to cope with high packet rates. Such requirements constrain applicability in general purpose equipment and significantly increase cost.

Previous work also highlights the interest of studying TCP retransmissions as network KPI, as they are related to loss ratio. For instance, in [24] two *naïve algorithms* are proposed that map retransmissions into packet losses. To do so, they keep track of the whole TCP sequence number history, to detect if a packet contains data that was previously sent by one host but not received by the other. Nonetheless, several TCP behaviors introduce errors in the estimations provided by such algorithms. For instance, segments may carry partially retransmitted data together with new data, multiple consecutive retransmissions may be grouped in a single segment or re-packaged in different segments, etc. Moreover, only pathological flows with severe retransmissions jeopardize QoS. Thus, approximate methods for retransmission detection which sacrifice accuracy for speed and memory savings suffice.

In conclusion, further efforts have to be made for the detection of pathological TCP flows, in terms of retransmissions, with a cost-effective approach. Particularly, we propose to evaluate simple heuristics to identify retransmissions, which drastically reduce the TCP sequence number list to be tracked and increase throughput, yet being indicative enough of a potential QoS degradation event.

2. Retransmission detection approaches

In what follows, we will use the following notation. Given the n^{th} packet in a unidirectional flow of a TCP session from IP address A (SYN initiator) to IP address B, we provide the following definitions in Table 1.

Table 1: Notation

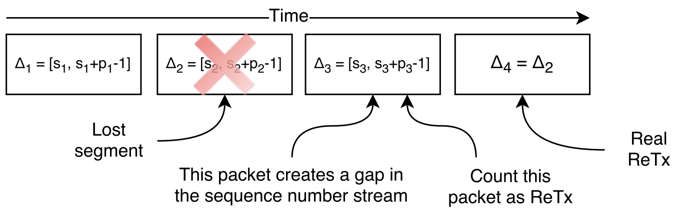
N	Number of packets in the flow.
$l_n^{A\to B}$	TCP segment payload length.[1]
$s_n^{A\to B}$	TCP segment sequence number.
$ACK_n^{A\to B}$	TCP segment ACK number.
Δ_n	*Sequence number space* covered by packet n.
ReTx	Packet with retransmitted TCP segment.

We define $s_n^{A\to B}$ such that $\forall n \in \mathbb{N}, n \leq N, \exists k < n$:

$$s_n^{A\to B} = s_k^{A\to B} + l_k^{A\to B} \tag{1}$$

for some initial $s_0^{A\to B}$ —note that a TCP implementation can choose any 32-bit value for $s_0^{A\to B}$. That is, the sequence number

[1]Since SYN and FIN flags consume a sequence number, $l_n^{A\to B}$ equals the segment payload plus one for each of those flags set.


Time
$\Delta_1 = [s_1, s_1+p_1-1]$
$\Delta_2 = [s_2, s_2+p_2-1]$
$\Delta_3 = [s_3, s_3+p_3-1]$
$\Delta_4 = \Delta_2$
Lost segment
This packet creates a gap in the sequence number stream
Count this packet as ReTx
Real ReTx
(a) *Gap Detection* heuristic.

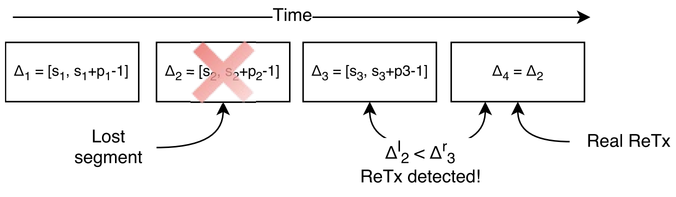

Time
$\Delta_1 = [s_1, s_1+p_1-1]$
$\Delta_2 = [s_2, s_2+p_2-1]$
$\Delta_3 = [s_3, s_3+p3-1]$
$\Delta_4 = \Delta_2$
Lost segment
$\Delta^l_2 < \Delta^r_3$
ReTx detected!
Real ReTx
(b) *Out of Order Detection* heuristic.

Figure 2: Comparison of the evaluated heuristics.

featured in the n^{th} TCP segment should be the next byte of some previous segment k. If packet n does not contain a retransmitted segment then $k = n - 1$, as TCP only transmits blocks of contiguous data.

As it turns out, we define the *sequence number space* (in short, *space*), covered by segment contained in packet n as the closed interval

$$\Delta_n = [s_n^{A\to B}, s_n^{A\to B} + l_n^{A\to B} - 1] \tag{2}$$

and we will denote the left limit of Δ_n by Δ_n^l, and the right limit by Δ_n^r.

2.1. Full detection of TCP retransmissions

We deem a TCP segment contained in packet m as a retransmission of segment in packet n if and only if $\Delta_m \cap \Delta_n \neq \emptyset$, $n = 1, \ldots, N$, $m > n$, assuming there is no duplicated traffic. We assume that the capture engine removes packet duplicates which may be due to the SPAN capture [25]. Actually, our HPCAP driver [6] effectively removes duplicates before they are forwarded to the monitoring probe software itself.

Thus, in order to detect retransmissions for a given segment in packet n we should keep track of all the previous sequence number spaces Δ_m with $m < n$. Namely, a data structure has to be created per flow and has to be sought for each incoming packets to detect possible retransmissions. Such data structure will be denoted by “sequence number list”. We note that if a packet is lost during capture or packets arrive out of order, the sequence number list will have non-contiguous *spaces*.

The sequence number list can be kept sorted using the following comparison function:

$$\Delta_i < \Delta_j \iff \Delta_i^l < \Delta_j^l \text{ or } (\Delta_i^l = \Delta_j^l \text{ and } \Delta_i^r \leq \Delta_j^r) \tag{3}$$

If $\Delta_i < \Delta_j$ and $\Delta_i \cap \Delta_j \neq \emptyset$ then we can merge them in

$$\Delta' = [\Delta_i^l, \max(\Delta_i^r, \Delta_j^r)] \tag{4}$$

to keep the sequence number list as short as possible.

In real monitoring scenarios, it is not always the case that all packets appear in the capture, nor that they appear in the same order they were actually transmitted. If a packet is lost during capture and is actually a retransmission or contains partially retransmitted data, such retransmission will not be detected. On top of that, if such lost packet is never retransmitted, a permanent gap will appear in the flow sequence number list.

Assuming there is no packet reordering, then the $ACK_n^{B\to A}$ can be used to fill the gaps in the sequence number list of host B, as it indicates that host B has received all data up to that sequence number. Moreover, if selective acknowledgement (SACK) TCP extension is enabled, it can also be used to fill gaps in the sequence number list.

In any case, in such approach we note that a data structure that contains the *sequence number spaces* seen per connection in the past is required, for all concurrent connections. Then, for each incoming packet, the corresponding connection entry in the data structure should be sought in order to update the sequence number list. Such a procedure is computationally expensive for high packet rates and does not scale to arbitrarily high speeds, thus calling for a simpler approach, which makes them also more suitable to be deployed in a NFV environment or in an embedded device.

2.2. Approximate detection of TCP retransmissions

In this subsection we present two simple algorithms to estimate the number of retransmissions in TCP flows with severe retransmissions. We will show that such simple procedures are accurate enough and serve to the purpose of on-the-fly evaluation of retransmissions at 10 Gb/s per core.

Our proposed heuristics are based on the fact that a typical TCP sender transmits packets in order, and that each segment features a sequence number $s_n^{A\to B}$ which is equal to the previous segment sequence number plus its payload length, namely,

$$s_n^{A\to B} = s_{n-1}^{A\to B} + l_{n-1}^{A\to B} \tag{5}$$

should no retransmissions occur, as derived from Equation (1). Consequently, we propose to track only the highest sequence number seen per TCP flow, and then perform a simple comparison to decide whether a segment is a retransmission or not.

In this light, two simple decision algorithms can be applied:

- The first heuristic, which will be denoted by *Gap Detection* heuristic, counts the number of packets which contain a larger sequence number than the one that was expected, thus creating a gap in the receiver's buffer ($s_{n+1}^{A\to B} > s_n^{A\to B} + l_n^{A\to B}$). The aforementioned gap will have to be filled by a latter packet, which is the actual segment retransmission.

  Figure 2a serves to illustrate this phenomenon. We note that the second packet is lost before the capture point. Then, upon arrival of the third packet, a gap arises in the

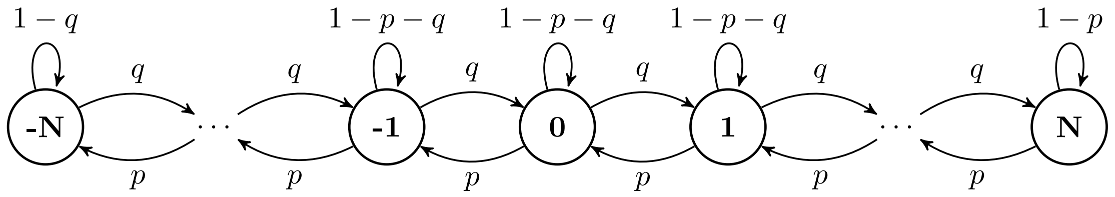


Figure 3: Markov chain of cumulative number of classification errors

Table 2: Parameters of the error model

p	$\mathbb{P}\{\text{packet is classified as false retransmission}\}$
q	$\mathbb{P}\{\text{retransmission is not detected}\}$
$1-p-q$	$\mathbb{P}\{\text{packet is correctly classified}\}$
N	The maximum number of packets a flow can have

sequence number stream and it is counted as a retransmission. In the figure, the real retransmission is the next packet (which is not counted as a retransmission), but in a general setting the real retransmission may appear several segments later.

- The second heuristic, which we will call *Out of Order Detection* heuristic, consists in counting the packets with payload which have a sequence number smaller than the one expected, *i.e.*, $s_{n+1}^{A\to B} < s_n^{A\to B} + l_n^{A\to B}$. With the given assumptions, such segments should be retransmissions.

  In Figure 2b we illustrate the former heuristic using the same example as Figure 2a. We note that $s_2^{A\to B}$ is lost in this case. If we happen to receive $s_2^{A\to B}$ again, which has a sequence number smaller than the largest one seen, namely $s_3^{A\to B}$, we count it as a retransmission. Actually, it turns out to be a real retransmission in the example.

As a result, these proposals are computationally efficient and present a very small memory footprint per flow. Furthermore, they can be generalized to track a maximum number of sequence number spaces (instead of only one) in order to overcome mild network packet reordering. That is, both generalized versions mimic the full reconstruction algorithm, but with a maximum number of tracked sequence number spaces per flow. As a result, some differences arise when updating flow state, such as:

- In case of the generalized *Out of Order Detection* heuristic, packets that overflow the sequence number list are counted as retransmissions.
- In case of the generalized *Gap Detection* heuristic, every space in the list must be checked to detect gaps which are counted as retransmissions.

2.3. Classification error

In order to evaluate the effectiveness of our proposed algorithms, we provide an analytic model for the probability of erroneously classifying a flow as pathological. We define the parameters in Table 2.

Let $\mathcal{E}_N$ be the total classification error for a given flow of N packets,

$$\mathcal{E}_N = \#\{\text{ReTx by ground truth}\} - \#\{\text{ReTx by algorithm}\} \quad (6)$$

namely, $\mathcal{E}_N$ is the sum of the classification errors incurred per packet in the flow. So for a single packet flow, p represents the probability of $\mathcal{E}_1 = -1$, q the probability of $\mathcal{E}_1 = +1$, and $1-p-q$ is the probability of $\mathcal{E}_1 = 0$. Assuming the classification of each packet is independent, we can model $\mathcal{E}_N$ as a finite-state irreducible Markov chain where each state represents the cumulative sum of classification errors. Indeed, p and q represent the transition probabilities to the next and previous states, and $1-p-q$ is the probability of remaining in the same state, as shown in Figure 3.

For such a Markov chain, with $2N+1$ states, the transition probability matrix $(2N+1)\times(2N+1)$ is as follows:

$$\mathcal{P} = \begin{pmatrix} 1-q & q & 0 & & \dots & & & \\ & & & & \ddots & & & \\ \dots & 0 & p & 1-p-q & q & 0 & \dots & \\ & & & & \ddots & & & \\ & & & & \dots & 0 & p & 1-p \end{pmatrix} \quad (7)$$

We note that the number of classification errors, namely $\mathcal{E}_N \in [-N, N]$, is a random variable whose probability mass function can be easily computed by $\lambda\mathcal{P}^N$, where

$$\lambda = (\overbrace{0,\dots,0}^{N\text{ times}}, 1, \overbrace{0,\dots,0}^{N\text{ times}}) \quad (8)$$

is our initial distribution.

In order to compute the classification error, regardless of the number of packets N in a flow, we must incorporate the distribution of the number of packets in a flow. Zipf-like phenomena has been observed in the past in Internet traffic traces [26, 27], or at least in the distribution tail. Thus, we define the probability distribution function (PDF) of the number of packets in a flow as

$$\mathbb{P}(X = x) \approx Kx^{-\alpha}, \quad K > 0,\ x \geq x_{min},\ \alpha > 0 \quad (9)$$

where K is the normalization constant and typically $2 < \alpha < 3$.

Thus we can express our expected classification error $\mathbb{E}(|\mathcal{E}_N|)$ as follows

$$\begin{aligned}\mathbb{E}(|\mathcal{E}_{N_{max}}|) &= \sum_{\varepsilon=-N_{max}}^{N_{max}} |\varepsilon| P(\mathcal{E}=\varepsilon) = \\ &= \sum_{\varepsilon=-N_{max}}^{N_{max}} |\varepsilon| \sum_{n=x_{min}}^{N_{max}} P(\mathcal{E}=\varepsilon|N=n)P(N=n) = \\ &= \sum_{n=x_{min}}^{N_{max}} \sum_{j=1}^{2n+1} |j-1-n| \, (\lambda\mathcal{P}^n)_j \, K n^{-\alpha} \qquad (10)\end{aligned}$$

where N_{max} is the maximum number of packets per flow in the trace and K is a normalization constant to make the distribution of number of packets in a flow add to unity.

Finally we remark that given error probabilities (p, q), the $\mathcal{E}_N$ distribution is symmetric to the one with error probabilities (q, p). In simulations we have observed that the expected error seems to level off at a value which depends on α, p and q for a sufficiently large value of N.

2.4. Expected memory savings

Memory consumption is a key issue for performance, because a memory lookup must be performed for each incoming packet. Therefore, performance is constrained by the read/write memory latency. As it turns out, there is a tradeoff between memory size and speed, namely cache memories are very fast, but also small.

Clearly, the required memory size depends on the number of concurrent flows present in the traffic. In this section we provide an analytical approximation that allows obtaining the memory size required for a given number of flows and vice versa. To do so, we analyze the decrease in number of *spaces* in the sequence number list brought by the heuristic algorithms. In what follows, the following notation is adopted:

- λ_t, the rate of new connections observed in the monitored vantage point.
- W_t, the average time that connections are tracked by the monitoring system. This time is related to the typical duration of a connection in the specific network, but also to monitoring parameters such as the configurable garbage collection times.
- M, denotes the upper bound of the number of *spaces* that can be tracked per flow.
- I_t^M, the expected number of *spaces* in the sequence number list per flow, with an upper bound of M *spaces*.
- L_t^M, is the total expected number of sequence number *spaces*, with an upper bound of M *spaces* per flow.

With the above metrics and, following Little's law, the number of tracked *spaces* is given by the expression in Equation (11):

$$L_t^M = \lambda_t \cdot W_t \cdot I_t^M \qquad (11)$$

Namely, the expected number of total sequence number *spaces* for a given bound per flow (M) is the rate of new connections times the average time a connection stays in the system times the expected number of *spaces* per flow.

In this light, we can compare the complexity and memory usage of two algorithms with different M values. Specifically, we are interested in the comparison of our proposal, $M = 1$, with the approach that considers the whole list of sequence numbers M, using the metric in Equation (12):

$$\Delta L_t^M = L_t^M - L_t^1 = \lambda_t \cdot W_t \cdot (I_t^M - I_t^1) \qquad (12)$$

Note that W_t equals the mean TCP session duration plus the additional time it resides in memory until it is exported [1].

We distinguish two set of sessions regarding the expiration mechanism:

- TCP flags: a session is expired after receiving FIN or RST flags. However, some packets may still be in transit in the network. In order to tackle this issue, RFC 1122 [28] defines TCP TIME-WAIT state timeout as 2 MSL, with MSL arbitrarily set to 120 seconds.
- Inactivity: a session is expired if no packets are received in a given time interval. For example, if client and server become isolated due to loss of connectivity the corresponding TCP sessions may crash. Then, the corresponding flow entry in the sequence number list must be released, to avoid filling the memory with garbage. Therefore, we can choose the maximum time an inactive session is kept alive in our monitoring system. We named such parameter "Inactivity Time-Out", or ITO. Hereafter, we consider a value of 15 minutes (900 seconds) during our tests, as we have empirically found that this figure provides a good tradeoff between memory fingerprint and flow completeness.

In this light, we define

$$W_t = D_t + c_1 \cdot (2 \cdot MSL) + c_2 \cdot ITO \qquad (13)$$

with D_t the random duration of a TCP session and c_1,c_2 are the expected proportion of flows expired by TCP flags or inactivity, respectively. We note that $c_1 + c_2 = 1$, as they represent the probabilities of complementary sets. Hence, both depend on the number of sessions that have to be expired due to inactivity. As a result, increasing ITO will decrease c_2 and vice versa, because a smaller value of ITO increases the chance that a flow is deemed inactive.

In Table 3 we show some estimations of required memory to track a given number of concurrent TCP sessions. For such estimations, we consider a minimal state representation of 64 bytes per TCP session, that includes the flow 4-tuple (12 bytes),

Table 3: Memory requirements in MB

Concurrent TCP Sessions	6.4M	3.2M	1.6M	800K	400K	200K	100K
minimal	512	256	128	64	32	16	8
full reconstruction 1 *space* per flow	716.8	358.4	179.2	89.6	44.8	22.4	11.2
full reconstruction 1.1 *spaces* per flow	747.52	373.76	186.88	93.44	46.72	23.36	11.68

Table 4: Traffic traces used in the experiments. (*) TCP sessions with a hundred packets at least.

Trace	Size (GB)	Millions of packets	Average packet size	TCP Sessions*	Flows* with ReTx	Average % Reordered packets	Client to Server		Server to Client	
							α	K	α	K
A	91	149	634B	98279	2972	0.15	2.14	13.00	1.92	6.68
B	120	211	754B	196580	705	$1.97 * 10^{-3}$	2.28	19.71	1.88	3.93
C	387	539	591B	725648	13721	$2.16 * 10^{-4}$	2.35	24.02	2.03	9.24

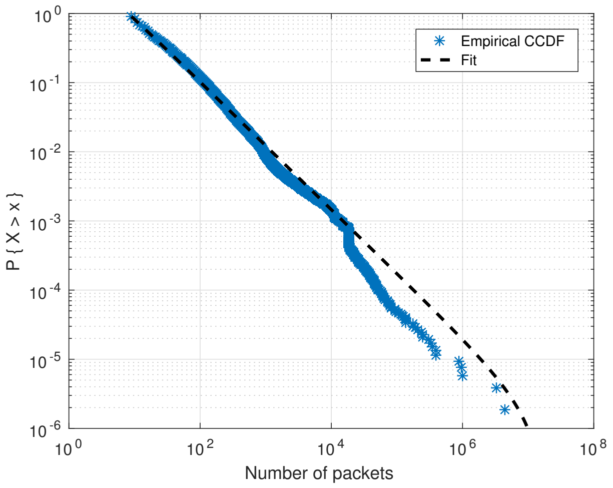


Figure 4: Example of packet distributions in our traces flows.

number of bytes, number of packets and number of ReTx counters (6×8 bytes), and the last packet timestamp in seconds (4 bytes). Additionally, we consider an expected average number of *spaces* per flow equal to 1.1 (which is a consistent with the empirical findings presented in Section 3), *i.e.*, 10% more *spaces* than tracked TCP flows. For the full reconstruction approach, the session representation size increases with the number of tracked *spaces* as further state information is required. We consider 24 bytes per *space*, corresponding to the use of a linked list. In this situation, we would approximately need 750MB to track 6.4 million concurrent TCP sessions. Furthermore, even if we track just one *space* per TCP session, about 720MB of memory would be required to track 6.4 million of concurrent TCP sessions. However, considering a minimal state representation of 80 bytes per TCP session, with exactly one *sequence number space* per flow (2×2×4 bytes per session), the memory footprint to track 6.4 million concurrent TCP sessions reduces to approximately 512MB.

As a result, our approach paves the way to implement a TCP retransmission detection module in small embedded devices, such as FPGAs [22], using much less memory (about 30% savings) with only a minimal loss in accuracy, as will be shown later.

3. Experimental evaluation

In this section we present an experimental evaluation of both the accuracy and performance of the proposed heuristics. First, we assess the accuracy of the heuristics when compared to the full reconstruction algorithm. Afterwards, we compare them to Tshark results, as it is considered the reference tool. These tests were performed at the receiver server, by reading the packet traces from disk. Lastly, we implement the heuristics in a traffic capture probe, M^{3}Omon [9], and measure whether the achievable throughput decreases due to the heuristics. In this case, packets are read from the NIC at line rate, to characterize the online system performance.

3.1. Experimental test setup

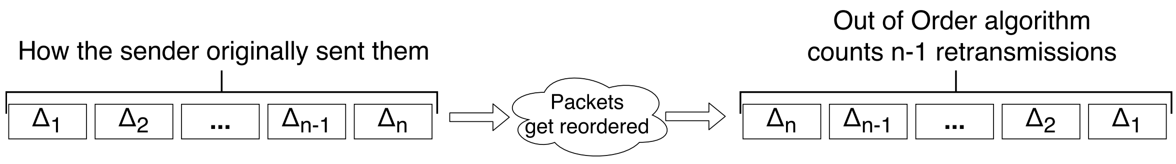
Our experimental setup consists of two Commodity Off-the-Shelf (COTS) servers directly connected with an optical fiber link. One server acts as the sender and the other one as the receiver.

Our sender server is based on an Intel Xeon E3-1230 v3 processor with Hyper-Threading disabled on a Supermicro X10SL7-F motherboard with 32 GB of DDR3 RAM. Such server uses a custom FPGA-based system [29] to send traces at 10 Gb/s over the fiber and reads the traces from a software-controlled RAID-0 with 8 SSDs disks. The disks are Samsung SSD 840 with 250 GB capacity each.

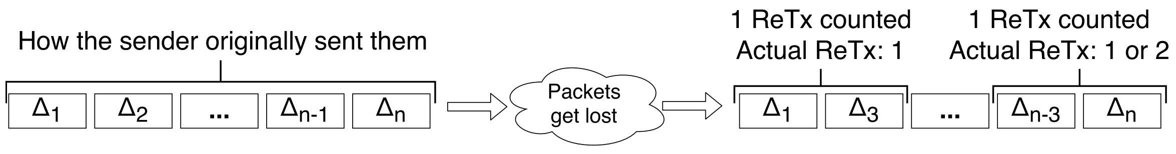
The receiver server is based on an Intel Xeon E5-2630 processor on a Supermicro X9DR3-F motherboard with 128 GB of DDR3 RAM. The receiver uses our custom driver called HPCAP [6] to capture the incoming traffic. The traffic is processed with our M^{3}Omon software [9] which concurrently stores packets in a 10 disc RAID-0 controlled with a LSI Logic MegaRAID SAS 2208 hardware controller and processes them. The RAID disks are 3TB Hitachi drives, model HUA723030ALA640 with SATA-3 interface.

Table 4 presents several parameters of the traffic traces that we used for our experimental analysis. They were captured during several days in enterprise networks, and most of the traffic is HTTP or HTTPS. The average packet size is approximately 760 bytes, similar to the average packet size of Internet traffic [7], resulting in approximately 1.6 million of packets per second with a rate of 10Gb/s. The average number of reordered packets per flow, shown in the table, are computed with the full reconstruction algorithm. These results are consistent to the ones present in previous studies, as in [30], where an average 0.79% of reordered packets is reported for IPv4 traffic.

In Figure 4 we show the Complementary Cumulative Distribution Function (CCDF) or *survival function* of the number of packets in a flow. As expected, packets seem to follow a Zipf distribution whose parameters have been estimated using the Maximum Likelihood Estimator (MLE) given in [31]. We have observed that TCP flows in the rest of our traces follow


How the sender originally sent them
Δ1 Δ2 ... Δn-1 Δn
Packets get reordered
Out of Order algorithm counts n-1 retransmissions
Δn Δn-1 ... Δ2 Δ1

(a) Disordered segments negatively affect *Out of Order Detection* heuristic.


How the sender originally sent them
Δ1 Δ2 ... Δn-1 Δn
Packets get lost
1 ReTx counted Actual ReTx: 1
1 ReTx counted Actual ReTx: 1 or 2
Δ1 Δ3 ... Δn-3 Δn

(b) Example showing *Gap Detection* heuristic limitations.

Figure 5: Situations that limit the accuracy of our heuristics.

similar distributions. In Table 4 we show the fitted parameters for Equation (9) for each trace, where x_{min} took values between 6 and 8. Note that the value of α and K heavily depend of the choice of x_{min}. These parameters were computed minimizing the Kolmogorov-Smirnov distance of the fit distribution to the empirical distribution. This figure shows that the analytical approximation used in equation 9 is sound, as also expected from the state of the art in modeling packets per TCP flow [26, 27].

3.2. Accuracy evaluation

In this section, we show the accuracy of the heuristics versus the full reconstruction approach. From section 1 we recall that a unidirectional flow of a TCP session, with at least one hundred packets, is classified as pathological (*PF*) if more than 5% of its packets are retransmissions, being the rest non-pathological (*NPF*).

Then, the following situations arise, on a per-flow basis:

1. True Positive (TP): our heuristics classify the *NPF* flow correctly as *NPF*.

2. False Positive (FP): our heuristics detect more retransmissions than the ground truth and they classify the *NPF* flow as *PF*.

3. False Negative (FN): our heuristics do not detect enough retransmissions and they classify a *PF* flow as *NPF*.

4. True Negative (TN): our heuristics classify a flow correctly as *NPF*.

In Table 5 we show the False Negative Rates (FNR) and False Positive Rates (FPR) of the heuristics for each test case. In the tables, we refer to *Client* to the host who sent the first SYN packet and *Server* to the other host. We have shortened *Out of Order Detection* heuristic by "OO" and *Gap Detection* heuristic by "GD".

We observe that *Gap Detection* heuristic yields very poor detection results compared with *Out of Order Detection* heuristic, which has reasonable FPR and almost a 0% of FNR.

In order to understand these results we performed manual inspection with Wireshark, resulting in the following cases:

Table 5: Experimental results versus full reconstruction algorithm

Trace	Client to Server				Server to Client			
	FNR (%)		FPR (%)		FNR (%)		FPR (%)	
	OO	GD	OO	GD	OO	GD	OO	GD
A	0	62.56	0.69	17.18	0	96.50	1.41	0.91
B	0	88.89	0.02	0.01	0	96.82	0.09	0.01
C	0	14.29	0.01	13.19	0	99.88	0.02	0.01

- In trace A there were bursts of packets in reverse order. If one of such bursts has n packets, the *Out of Order Detection* heuristic will compute $n-1$ retransmissions. This is illustrated in Figure 5a. The *Gap Detection* heuristic will compute just 1 retransmission because only the first packet generates a gap in the sequence number stream. Thus, the *Out of Order Detection* heuristic overestimates the number of retransmissions for that flow, which may give a False Positive.

- If a gap is present in the sequence number stream, the *Gap Detection* heuristic counts a single retransmission only, as shown in Figure 5b. As stated before, some flows present multiple retransmissions after a gap, and in such cases the estimated number of retransmissions is less than the real number, giving rise to False Negatives.

- Flows with low packet counts are subject to higher FNR and FPR. Incorrectly classifying two packets out of a hundred may change the class of the flow, whereas erroneously classifying two packets out of a thousand does not affect the flow classification.

In conclusion, most of our experimental results present $\mathcal{E} = 0$ for the majority of flows using *Out of Order Detection* heuristic, very few false negative errors and even less false positive errors. Consequently, we expect to have a very low value of probability p (packet is classified as false retransmission) and even lower value of probability q (retransmission is not detected), which will be estimated later.

3.3. Comparison with Tshark

In this section we present our validation of the full reconstruction algorithm and the *Out of Order Detection* heuristic

Table 6: Flow classification versus Tshark

Algorithm	Client to Server		Server to Client	
	FNR (%)	FPR (%)	FNR (%)	FPR (%)
Full Reconstruction	0	0	0	2.38
OO	0	0	0	2.38

versus Tshark. In our experimental setup we used Tshark version 1.10.6 distributed with Ubuntu 14.04.

Both the accounting of retransmissions per flow and the consideration of a packet as including retransmitted data depends on the flow definition criteria. On the one hand, Tshark does not include any type of timeout for flow expiration. For instance, Tshark identifies a new TCP stream for a given 4 tuple only after a new SYN, even if more than 2 MSL seconds have passed after the connection entered the TIME-WAIT state[32]. On the other hand, Tshark does not seem to implement any garbage collection timeout, so even after multiple hours of silence, as long as the SYN segment is not in the capture, two TCP sessions will be merged together. However, most flow based monitoring solutions implement some sort of flow expiration policy to avoid running out of resources for inactive flows [8]. For example, NetFlow expires flows with timeouts in the order of minutes of inactivity [1]. The consequences are that in case of packet loss during the capture process, which is quite usual in high-speed networks, Tshark may merge several TCP sessions and distort the per-flow statistics.

Additionally, we note that Tshark is ill-suited for very large traces which prevented us from using it to analyze our whole dataset Coherently, our experimental methodology consists in two steps. First, we pre-filter TCP sessions in individual traces to prevent Tshark from merging sessions. More specifically, we performed a stratified sampling of trace C, randomly sampling 2500 TCP sessions which had no retransmissions and 2500 TCP sessions which had some retransmissions. Out of such 2500 sessions with retransmissions, 14% turned out to have severe retransmissions. These 5000 TCP sessions add up to about 5.2 million packets in total. Second, we obtain fine grained results in a packet-based fashion, and after that we aggregate them to homogenize the definition of flows.

In Table 6 we show the comparison results for the 5000 TCP sessions sample. We note the error rates for both the full reconstruction algorithm and the *Out of Order Detection* heuristic are the same for the sampled TCP sessions, and the classification error for packets is small. This is expected as Tshark has a small reordering window and marks some packets as "out of order" rather than retransmissions. That explains the FPR slightly above 2% in Server to Client TCP flows, which present higher retransmission rates than Client to Server ones. In any case, the results show remarkable accuracy of the *Out of Order Detection* heuristic.

Finally, we note that this TCP sessions sample can be used to estimate the (p, q) parameters of our error model. We instrumented our implementation of the algorithms to get per packet classification, and compared it against Tshark classification. We estimate that $\hat{p} \approx 1.2 \times 10^{-3}$ and $\hat{q} \approx 1.8 \times 10^{-4}$, which, according to our model, gives less than 1.5% error when classifying flows of up to $N = 8000$ packets.

Table 7: Performance evaluation results

	Throughput (Gb/s) $\bar{x} \pm \sigma$	Throughput (pkt/s) $\bar{x} \pm \sigma$	Lost packets (%) $\bar{x} \pm \sigma$
M^{3}Omon	9.652 ± 0.014	$1.584 \cdot 10^6 \pm 2510$	1.21 ± 0.09
M^{3}Omon + OO	9.645 ± 0.013	$1.585 \cdot 10^6 \pm 2662$	1.25 ± 0.11

3.4. Performance evaluation

Following the accuracy results presented above, we select *Out of Order Detection* heuristic for the performance evaluation, as it outperforms the other proposed heuristic approach. To carry out this performance assessment, we chose to send trace C at 10 Gb/s because it is the largest in bytes, so it made each test run longer —about 336 seconds. This allowed us to test if the system had any performance degradation or instabilities over time. First we measured the amount of lost packets and the average throughput of M^{3}Omon [9] without our algorithm as a baseline. The experiment was performed 20 times and then, we calculated the average and standard deviation of each metric that appears in the table. Then we repeated the experiment adding the *Out of Order Detection* heuristic in M^{3}Omon [9] and checked for performance penalties.

In Table 7 we show our experimental results. We note that most lost packets occurred during the first few seconds of the capture, then the system stabilized and run stable the rest of each experiment. Our results show that the cost of running the algorithm is negligible, and that it can be deployed in production environments to detect flows with pathological retransmissions with a packet rate of at least 1.5 million of packets per second.

3.5. Evaluation of Memory Consumption

We have also assessed the accuracy of the memory consumption model presented in section 2.4. We found that it matches the experimental data during working hours, but it marginally underestimates the real memory usage during the nights. Additionally, and to minimize the effect of the empirical value of W_t in our estimations, we provide further bounds of the memory consumption based on the empirical distribution of such parameter. More specifically, we define lower and upper bounds by using the 10th and 90th percentiles of W_t, respectively.

In Figure 6 we show the CCDFs of both the values of *spaces* predicted by the model given in section 2.4 and the relative error for trace C. Since the sequence number list is compacted whenever possible, peaks of up to 2% of retransmitted packets do not necessarily correlate with spikes in memory usage. We note that our measured memory consumption is adequately bound by the values predicted by our model, and that, on average, a minimal memory representation saves 10% of the *spaces* the full reconstruction approach tracks.

4. Discussion

In a nutshell, our proposal provides highly accurate results in typical network conditions. However, we identified the following error-prone cases:

1. If the network severely reorders the packets before the point of capture, both proposed algorithms give incorrect results. In the full reconstruction algorithm, the Out-of-Order (OO) packets can be placed in order and no retransmissions are detected. However, both of our proposed heuristics count multiple retransmissions, depending on the ordering of the segments. We note that reordering rates are dependent on the site monitored and the path that packets take on the Internet [33]. Reordered packets within the same flow constrain TCP performance, and previous studies show that packet reordering happens with low probability [33], also shown in our experimental data in Table 4.

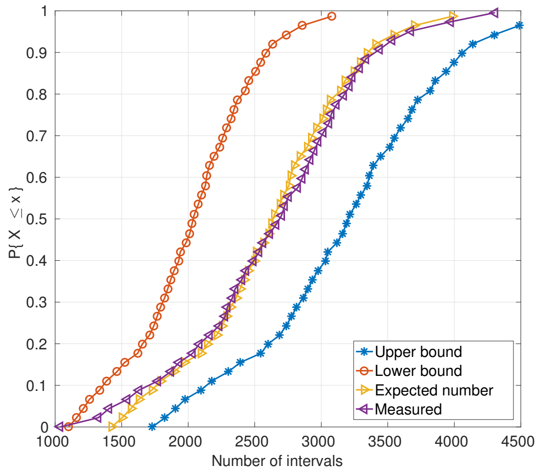
2. If TCP segments are lost before the point of capture, the corresponding retransmissions will not be detected unless the host sends more data packets. As shown in Figure 7, if Host A sends an ACK after the lost segment, the corresponding ACK sequence number shows that packets have been lost, but our heuristics will not detect it because they do not take ACK packets into account.

3. The *Gap Detection* heuristic shows effectiveness in detecting gaps but cannot estimate the number of packets in the gap, which is necessary in order to determine the number of retransmissions. It merely counts one retransmission per gap, but there can be many, which hinders its ability to estimate the number of retransmissions.

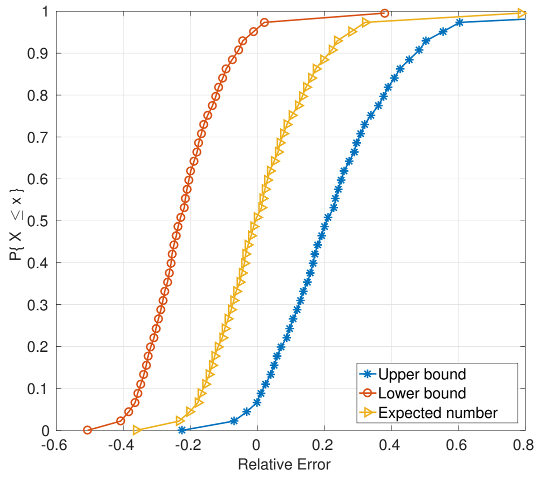
4. We have also evaluated the effect of increasing the upper bound of tracked *spaces* per flow in the generalized versions of the algorithms. Our results show negligible improvements in accuracy and higher resource utilization, which further motivate the exploitation of the simplest versions.

Additionally, the performance evaluation of the *Out of Order Detection* heuristic shows that the computational cost of such algorithm is marginal with respect to the overall cost of flow record generation. Moreover, it also reduces the required memory of more complex techniques to detect TCP retransmission.

This set of results allows us to define a guide to select the most suitable tool for a specific TCP traffic analysis scenario. To do so, we summarize the findings extracted throughout this paper in Table 8. If we consider Tshark, we have shown that it cannot cope with huge traffic volumes analysis as a result of its memory consumption. Additionally, it cannot process more than 1Gb/s per processing core. Consequently, this tool should be applied only in low rate scenarios, as it provides very detailed results. M^{3}Omon overcomes these limitations after integrating the *Out of Order Detection* heuristic, providing accurate TCP retransmission detection rates with high performance and



(a) Distributional values.



(b) Relative error.

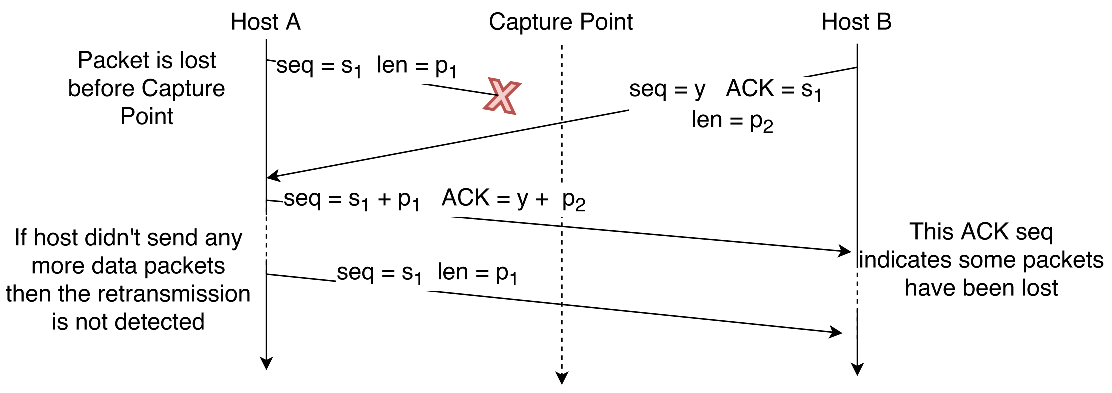
Figure 6: Behavior of predicted memory and complexity savings versus measured values.

low hardware requirements. Therefore, our solution paves the way for advance network monitoring elements that take into account the existing tradeoff between accuracy and resource consumption for demanding scenarios.

5. Conclusions

This paper has shown that simple heuristics to estimate the number of TCP retransmissions serve to the purpose of detecting flows with pathological retransmissions at 10 Gb/s speed per processing core. We thoroughly analyzed two approximate detection algorithms, that only use partial information from the sequence number list of TCP sessions.

From the accuracy standpoint, we concluded that a very low FNR and FPR can be obtained with the *Out of Order Detection* heuristic. As for performance, we successfully achieved 10 Gb/s throughput per core, which is the maximum traffic rate our testbed supports. In fact, preliminary results of further internal tests reading traffic from memory show that the limits



Host A
Capture Point
Host B
Packet is lost before Capture Point
seq = s1 len = p1
seq = y ACK = s1 len = p2
seq = s1 + p1 ACK = y + p2
If host didn't send any more data packets then the retransmission is not detected
seq = s1 len = p1
This ACK seq indicates some packets have been lost

Figure 7: The last packet is lost and its retransmission goes undetected.

Table 8: Comparison of the evaluated tools

System	Memory requirements	Complete session state	TCP ReTx detection	Bandwidth / processing core	Suitable for big traces
Tshark	O (packets)	✓	✓	< 1 Gb/s	✗
M^{3}Omon	O (TCP flows)	✗	✗	10 Gb/s	✓
M^{3}Omon + OO	O (TCP flows)	✗	✓	10 Gb/s	✓

of our proposal can outperform those presented in this current work.

Future work includes scaling up the algorithms in throughput per core. To achieve scalability, some packet sampling strategy could be devised, which is the focus of our current research.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under the projects TRÁFICA (MINECO / FEDER TEC2015-69417-C2-1-R), Preproceso Inteligente de Tráfico (MINECO / FEDER TEC2015-69417-C2-2-R) and RACING DRONES (MINECO / FEDER RTC-2016-4744-7). The authors also thank the Spanish Ministry of Education, Culture and Sport for a collaboration grant.

References

[1] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, A. Pras, Measurement Artifacts in NetFlow Data, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 1–10.

[2] B. Claise, B. Trammell, P. Aitken, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information RFC 7011 (2013).

[3] A. Biernacki, K. Tutschku, Performance of HTTP video streaming under different network conditions, Multimedia Tools and Applications 72 (2) (2014) 1143–1166.

[4] R. K. P. Mok, E. W. W. Chan, R. K. C. Chang, Measuring the quality of experience of HTTP video streaming., in: N. Agoulmine, C. Bartolini, T. Pfeifer, D. O'Sullivan (Eds.), Integrated Network Management, IEEE, 2011, pp. 485–492.

[5] A. Chan, S.-J. Lee, X. Cheng, S. Banerjee, P. Mohapatra, The impact of link-layer retransmissions on video streaming in wireless mesh networks, in: Proceedings of the 4th Annual International Conference on Wireless Internet, WICON '08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 55:1–55:9.

[6] V. Moreno, J. Ramos, P. M. Santiago del Río, J. L. García-Dorado, F. J. Gómez-Arribas, J. Aracil, Commodity Packet Capture Engines: Tutorial, Cookbook and Applicability, IEEE Communications Surveys Tutorials 17 (3) (2015) 1364–1390.

[7] R. Sinha, C. Papadopoulos, J. Heidemann, Internet packet size distributions: Some observations, USC/Information Sciences Institute, Tech. Rep. ISI-TR-2007-643.

[8] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, A. Pras, Flow monitoring explained: From packet capture to data analysis with netflow and ipfix, IEEE Communications Surveys Tutorials 16 (4) (2014) 2037–2064.

[9] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gómez-Arribas, J. Aracil, Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems, International Journal of Network Management 24 (4) (2014) 221–234.

[10] J. L. García-Dorado, P. M. Santiago del Río, J. Ramos, D. Muelas, V. Moreno, J. E. López de Vergara, J. Aracil, Low-cost and high-performance: VoIP monitoring and full-data retention at multi-Gb/s rates using commodity hardware, International Journal of Network Management 24 (3) (2014) 181–199.

[11] W. Xia, Y. Wen, C. H. Foh, D. Niyato, H. Xie, A Survey on Software-Defined Networking, IEEE Communications Surveys Tutorials 17 (1) (2015) 27–51.

[12] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, R. Boutaba, Network Function Virtualization: State-of-the-Art and Research Challenges, IEEE Communications Surveys Tutorials 18 (1) (2016) 236–262.

[13] S. Sundaresan, S. Burnett, N. Feamster, W. De Donato, BISmark: A testbed for deploying measurements and applications in broadband access networks., in: USENIX Annual Technical Conference, 2014.

[14] Z. Aouini, A. Kortebi, Y. Ghamri-Doudane, Traffic monitoring in home networks: Enhancing diagnosis and performance tracking, in: 2015 International Wireless Communications and Mobile Computing Conference (IWCMC), 2015, pp. 545–550.

[15] S. Basso, M. Meo, A. Servetti, J. C. De Martin, Estimating packet loss rate in the access through application-level measurements, in: Proceedings of the 2012 ACM SIGCOMM Workshop on Measurements Up the Stack, W-MUST '12, ACM, New York, NY, USA, 2012, pp. 7–12.

[16] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, D. Towsley, A measurement-based study of multipath tcp performance over wireless networks, in: Proceedings of the 2013 Conference on Internet Measurement Conference, IMC '13, 2013, pp. 455–468.

[17] U. Lamping, L. E. Ontanon, G. Bloice, Wireshark developer's guide.

[18] M. Mellia, R. L. Cigno, F. Neri, Measuring IP and TCP behavior on edge nodes with Tstat, Computer Networks 47 (1) (2005) 1 – 21.

[19] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, P. Di Torino, D. Rossi, Experiences of internet traffic monitoring with tstat, IEEE Network 25 (3) (2011) 8–14.

[20] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, D. Rossi, Traffic analysis with off-the-shelf hardware: Challenges and lessons learned, IEEE Communications Magazine 55 (3) (2017) 163–169.

[21] D. V. Schuehler, J. W. Lockwood, A modular system for fpga-based tcp flow processing in high-speed networks, in: International Conference on Field Programmable Logic and Applications, Springer, 2004, pp. 301–310.

[22] M. Forconesi, G. Sutter, S. Lopez-Buedo, J. E. L. de Vergara, J. Aracil, Bridging the gap between hardware and software open source network developments, IEEE Network 28 (5) (2014) 13–19.

[23] P. Roquero, J. Ramos, V. Moreno, I. González, J. Aracil, High-speed TCP flow record extraction using GPUs, The Journal of Supercomputing 71 (10) (2015) 3851–3876.

[24] P. Benko, A. Veres, A passive method for estimating end-to-end TCP packet loss, in: Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE, Vol. 3, 2002, pp. 2609–2613 vol.3.

[25] I. Ucar, D. Morato, E. Magaña, M. Izal, Duplicate detection methodology for ip network traffic analysis, in: 2013 IEEE International Workshop on Measurements Networking (M N), 2013, pp. 161–166.

[26] J. L. Garcia-Dorado, J. A. Hernandez, J. Aracil, J. E. L. de Vergara, F. J. Monserrat, E. Robles, T. P. de Miguel, On the duration and spatial characteristics of internet traffic measurement experiments, IEEE Communications Magazine 46 (11) (2008) 148–155.

[27] N. Brownlee, K. C. Claffy, Understanding Internet traffic streams: dragonflies and tortoises, IEEE Communications Magazine 40 (10) (2002) 110–117.

[28] W. R. Stevens, TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms RFC 1122 (1997).

[29] J. F. Zazo, M. Forconesi, S. López-Buedo, G. Sutter, J. Aracil, TNT10G: A high-accuracy 10 GbE traffic player and recorder for multi-terabyte traces, in: 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), 2014, pp. 1–6.

[30] F. Li, X. Wang, T. Pan, J. Yang, Packet delay, loss and reordering in ipv6 world: A case study, in: 2016 International Conference on Computing, Networking and Communications (ICNC), 2016, pp. 1–6.

[31] M. Newman, Power laws, Pareto distributions and Zipf's law, Contemporary Physics 46 (5) (2005) 323–351.

[32] J. Postel, et al., Transmission Control Protocol RFC 793 (1981).

[33] Y. Wang, G. Lu, X. Li, A Study of Internet Packet Reordering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 350–359.