On the Impact of TCP Segmentation: Experience in VoIP Monitoring

David Muelas*, Jorge E. López de Vergara*†, Javier Ramos*, José Luis García-Dorado*, Javier Aracil*†
* High Performance Computing and Networking Research Group,
Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain.
† Naudit HPCN, S.L., Spain.
Email: {dav.muelas, jorge.lopez_vergara, javier.ramos, jl.garcia, javier.aracil}@uam.es

***Abstract*—Quality of Service (QoS) and Experience (QoE) monitoring is a must during the management of services deployed over the Internet. This is particularly critical for Voice over IP (VoIP), as its degradation is immediately perceived by end users. From our experience, we highlight the impact that TCP segmentation exerts on online VoIP monitoring systems. On the one hand, it makes difficult to interpret the segmented signaling messages for application monitoring. On the other hand, complete messages do not provide information about packet level behavior, which is necessary for internetworking layer monitoring. Paradoxically, the Network Management community has not paid much attention to this fact, although it compromises several VoIP management tasks. To fill in this gap, we provide an empirical evaluation of its impact for the most popular VoIP signaling protocols using traces from enterprise networks, and present the architecture and heuristic that we are currently developing to partially solve the effect of the segmentation. Our proposal avoids the overhead of reconstructing the entire data stream and, at the same time, it enables the analysis of the packets that are actually sent. To do so, it maps TCP flags with segmented application messages, and exploits data structures that reduce latency. In this way, our solution paves the way for online monitoring tools that take into account both internetworking and application layers performance.**

***Index Terms*—TCP segmentation; Traffic reconstruction; Online metrics; Voice over IP; Application Performance Evaluation.**

I. Introduction

The increasing number of services deployed over IP networks and the variety of activities that depend on such services call for integral network monitoring and analysis to assure their correct operation. From the network management standpoint, this is usually accomplished by analyzing the traffic that those services generate. The case of Voice over IP (VoIP) deserves particular attention, as end users rapidly detect its degradation. Hence, the monitoring of such services requires the continuous consideration of both Quality of Service (QoS) and Experience (QoE) metrics to assure optimal service levels. Interestingly, it usually encompasses signaling and multimedia traffic, and both types of traffic must be considered to obtain a comprehensive set of Key Performance Indicators (KPIs) —some of them related to multimedia streams, such as estimated Mean Opinion Score (MOS), and others purely associated to signaling traffic, such as the Post Dial Delay (PDD). Furthermore, this network monitoring field does not only requires retrieving the information which is present in the application level messages, but also to account for the traffic profile at transport and network layers. Coherently, network monitoring tools sniff traffic below application layer and all applications *suffer* from the peculiarities of the transport level protocol behavior —*e.g.* TCP or UDP. Particularly, if traffic is encapsulated on top of TCP, the apparition of segmented messages is a possibility that must be considered to extract fully valuable measurements[1].

During our experience in the monitoring of VoIP deployments, we have realized the importance of this fact: the segmentation of messages in the TCP stack produces errors during the analysis of such traffic, especially during on-line monitoring. Additionally, on-line monitoring tools provide network managers with almost instantaneous indicators of the network state which makes complete TCP stream reassembly unfeasible in current high throughput and demanding scenarios —especially when the latency between the data acquisition and the presentation of results must be low, as it is usually the case for VoIP management. Furthermore, a complete characterization of the service levels for VoIP monitoring requires to correlate signaling connection and multimedia transmissions. As a result, delays in signaling TCP stream reconstruction jeopardize the results of traffic analysis.

This work provides an insight into the impact of TCP segmentation in VoIP monitoring and the solution we have applied to face it. Our study focuses on voice signaling messages which include relevant data during calls negotiation. The importance of such messages is justified because if they are lost, monitoring tools may produce erroneous statistics and lose the corresponding multimedia transmissions. Hence, our contributions are intended to pave the way for smarter *application agnostic strategies* that improve the mitigation of this matter, given its potential effects during the study of applications over the Internet. With *application agnostic strategies* we denote methodologies that cover all kind of applications, leading to a common middleware for upper-layer analysis devices.

Our solution exploits the relations between the segmented application messages and TCP flags to mimic the sender/receiver TCP stack in in-between monitoring systems.

[1]http://www.lovemytool.com/blog/2016/08/analyzing-tcp-segmentation-offload-tso-with-wireshark-by-paul-offord.html

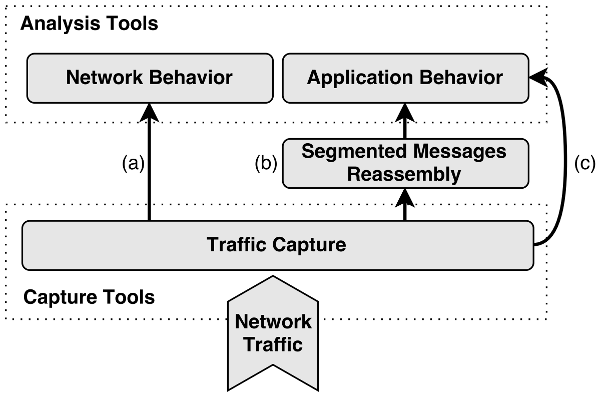

Fig. 1. Description of the segmented messages reassembler architecture.

It leverages a heuristic treatment of segmented messages to improve the trade-offs between the delay and accuracy of their analysis. Remarkably, this design allows a simultaneous consideration of internetworking and application level behaviors during traffic analysis. It provides an improved access to traffic data, as upper layer tools can analyze the behavior at network and transport level if the traffic is directly analyzed after the capture, illustrated as stream (a) in Fig. 1, and also at the application level by accessing the reassembled messages in stream (b) instead of using (c) as data source.

The rest of the paper is structured as follows. Section II includes a review of some previous works related to our study. We highlight the advantages of our approach when compared to several state-of-the-art solutions by discussing the limitations we have found when applying them to traffic analysis and VoIP monitoring. In Section III, we present an algorithm that makes use of TCP header information to mitigate segmented application messages for the main VoIP signaling protocols. After that, in Section IV we evaluate the impact of TCP segmentation for VoIP services as well as its reduction after the application of our algorithm. To do so, we consider a set of real traces from enterprise VoIP deployments using the most popular signaling protocols. Finally, we summarize the main contributions of our work and depict future work lines in Section V.

II. TCP Segmentation and Network Monitoring

Let us now review some previous results that motivate our study and pointed out the impact of TCP segmentation during traffic analysis. As distinguishing elements, this paper extends this state-of-the-art results by *(i)* estimating the impact of TCP segmentation in the VoIP domain, and *(ii)* describing an algorithm to improve on-line monitoring tools.

In [1], the authors included an evaluation of TCP segmentation in the case of HTTP traffic classification. They found that TCP reassembly could be omitted as they focused on traffic switching taking into account layer 4 (and higher) information. Nonetheless, such information losses are unacceptable during traffic analysis while on-line processing may still be a must.

The technical report in [2] includes an evaluation of security issues related to the TCP stack behavior in several systems. The authors pointed to two different situations where TCP segmentation makes more difficult to monitor and analyze network traffic. On the one hand, they claim that malicious segmentation of application messages can mislead Intrusion Detection Systems (IDSs) when detecting some anomalous event. On the other hand, TCP segmentation also changes the manner in which applications and middleboxes manage the transmitted data. These facts illustrate the motivation of strategies that mitigate the segmented application messages which are indiscriminately processed by traffic analysis tools.

Other works, such as [3], [4], [5], [6], considered different aspects of complete TCP stream reassembly, which is a demanding task that can be exploited by adversaries to saturate network elements which implement it. This motivated the solution presented in [3], where a hardware TCP reassembly module is presented. Alternatively, our solution considers segmented messages and not the whole TCP stream, which reduces the cost of the traffic preprocessing —as it filters a subset of segments to reduce reassembling cost. In [4], [5], the authors used the reconstructed TCP streams to detect anomalous events. Their solutions highlight the importance of the study of the relations between transport and application layers' behavior during the optimization of network monitoring tools. Finally, the authors in [6] exposed the importance of improving the reconstruction of incomplete network data, as otherwise important information losses may arise —a matter partially related to TCP segmentation. Nonetheless, all the approaches in such works can be useless during network monitoring tasks with strict latency and synchronization requirements.

Regarding voice analysis solutions, the system which is presented in [7] includes a module to cope with segmented SIP messages. The authors claimed that such module is required to track all the VoIP connections, as complete signaling messages over TCP are required to fully form call records. Nonetheless, their approach is specific for SIP traffic, while the problem appears in other VoIP signaling protocols —*e.g.*, Skinny Client Control Protocol (SCCP) used in many Cisco VoIP deployments.

The incorporation of software elements such as our solution or the layer for SIP messages reconstruction commented above can help attenuating the distortions that the network and transport layers introduce in the results of traffic analysis. Remarkably, this problem does not only affect to VoIP monitoring, but also to every application which uses TCP as transport protocol —see, for example, the study in [8], which illustrates the effect of middleboxes on the dynamic of TCP streams. The conclusions of that work pointed out to a wide range of alterations, which motivates the incorporation of elements that minimize the consideration of bogus traffic data in application-oriented monitoring tools. In this light, the modular design of current tools makes the testing of these functional modules easier: for example, in [9] the authors presented an architecture which separates common activities (such as flow generation) from the main analysis in network monitoring systems. As we will show, the inclusion of our lightweight TCP reassembly layer is totally straightforward with this type of architectures, and it can help to better understand the TCP segmentation impact during the monitoring of applications over the Internet.

III. Agnostic reassembling of application messages

A. The root of the problem

TCP uses a stream-oriented strategy to send data, but the description and specification of its implementation *(i)* does not provide any fixed criteria about the data segmentation and *(ii)* is not always met by vendors. Thus, the behavior of segments transporting application data depends on the concrete TCP stack implementation —*e.g.*, in the case of Linux-based systems, the RFC 793 [10] is implemented for the TCP specification, and RFC 1122 [11] for the TCP requirements.

Moreover, some Network Interface Cards (NICs) provide functionality to both segment and reassemble large data buffers. We can distinguish different TCP offloading capabilities, such as Simple TCP Segmentation Offload (TSO) and TCP Offload Engines (TOEs) [12], among others. The use of such offloading methods is related to performance improvements during data transference [13], [14] and energy efficiency and power savings [15].

Many Operating Systems take advantage of TSO capabilities to reduce data movements between the different levels by *(i)* sending only one large data buffer which is finally segmented by the NIC; and *(ii)* recovering only a large data buffer that includes data from several TCP segments. TOEs are implemented by vendors, and the inclusion of support for such elements depends on the specific Operating System which is used —*e.g.* Linux does not support such elements [12]. Furthermore, some high performance capture engines cannot use these mechanisms [16], which forces the introduction of software elements above the system network stack to handle segmentation of application data in scenarios with a demanding traffic load —which also motivates the use of lightweight heuristics.

B. Algorithm description

We have designed an algorithm based on a heuristic that takes advantage of the meaning of the TCP PUSH flag to reduce the amount of segmented messages in on-line monitoring systems: roughly speaking, our heuristic can be understood as a selective buffering policy based on the semantic of that flag. Algorithm 1 illustrates the main stages of the whole process. Some memory and data management activities must be implemented to get a fully operational reassembler —*e.g.*, as in general traffic capture processes segment losses can arise, garbage collection elements must be implemented. Thus, a configurable expiration time for segmented messages is included in our design, as such losses derive in unnecessary resource occupancy and extended latency when processing the available information.

The implementation of this algorithm should be included as a middle software layer as described in Fig. 1 (b). Interestingly, the design of the algorithm provides means to use that middle layer as a traffic source in the same manner as the standard `libpcap`: when the upper software layer request a new packet, it is checked before served. If it is a TCP segment with application data and the PUSH flag set to 0, all the segments in such connection are buffered until a segment with the PUSH flag set to 1 is received. After that, the next packet to be served to the upper layer is the buffered one, reducing the probability of serving segmented application messages to the upper analysis layer. We note that with this design, the integration of such a middle layer in a broad variety of network analysis tools that use `libpcap` is straightforward [17].

Algorithm 1 Application Agnostic Message Reassembler

```
$packet = getNextPacket()$
$IPHeader = getIPInfo(packet)$

**if** $(IPHeader.protocol == TCP)$ **then**
    $TCPHeader = getTCPInfo(packet)$
**else**
    **return** $packet$
**end if**

**if** $(getTCPPayloadLength(packet) == 0)$ **then**
    **return** $packet$
**end if**

**if** $(TCPHeader.PUSH == 0)$ **or**
$((TCPHeader.PUSH == 1)$ **and**
$(inSegmentedMessage(packet)))$ **then**
    $message = addSegmentedMessage(packet)$
    **if** $testCompleteSegment(message)$ **then**
        **return** $getPacket(message)$
    **end if**
**else**
    **return** $packet$
**end if**
```

Finally, the algorithm tests if the message has been completely received when a new segment is added to it: that is, if one of the segments includes the PUSH flag set to 1 (which is considered as the last segment with data from that specific message) and all the sequence numbers from the first to the last detected segments have been received.

C. Design and implementation

To decrease the latency during the buffering and data management, we use the structure represented in Fig. 2. This structure provides an access with cost $\mathcal{O}(1)$ during segment updates, thanks to the hash table (a) in the figure which provides an index of the active segments in terms of their 4-tuple —source IP address, source TCP port, destination IP address, and destination TCP port. Furthermore, the deletion of time-expired segments also requires access of cost $\mathcal{O}(1)$ as a result of the self-ordering properties of the list of messages

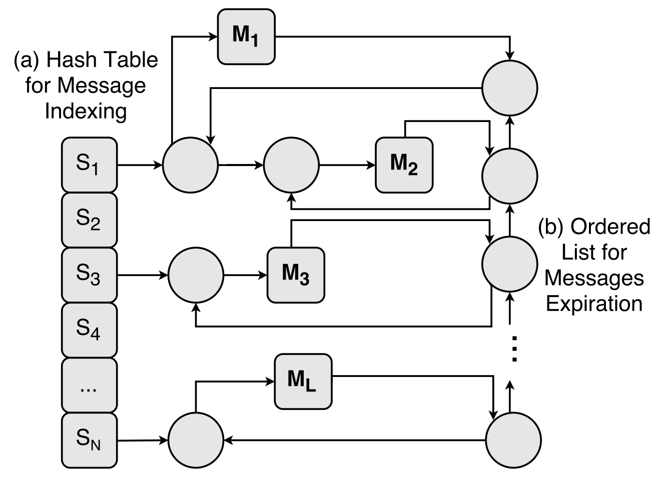
(a) Hash Table for Message Indexing
S_1
S_2
S_3
S_4
...
S_N
M_1
M_2
M_3
M_L
(b) Ordered List for Messages Expiration

Fig. 2. Description of the links among the data structures in our implementation of Algorithm 1.

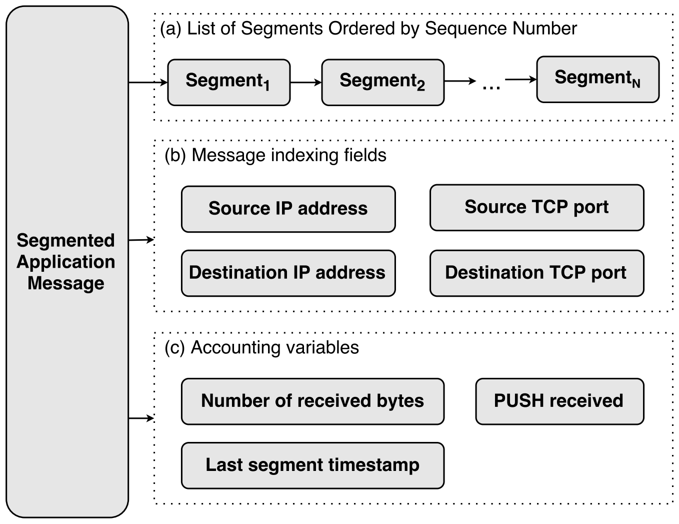

Fig. 3. Description of the content of the data structures to maintain the segment buffer state.

(b): we maintain it ordered by appending the message which includes the last received segment to the head of the list. The garbage collector expires every message starting from the tail of the list until it reaches a non-expired one.

The fields in Fig. 3 are considered to maintain the state of segmented messages during the reassembly process, and the information of all the segments is buffered in a list which is ordered by sequence number. Interestingly, this strategy maximizes the robustness of the algorithm against out-of-order segments, which is a situation that arises during network monitoring. Additionally, the fields of the 4-tuple used to index the segments must be included to solve hash collisions, and some accounting variables are required to conform a proper network packet with the buffered segments. While this mimics the behavior of the TCP stream reconstruction process, our proposal restricts its application to the segments in $\mathcal{P}$.

IV. Evaluation

A. Expected reassembly delay

As a first performance indicator for our method, we compare its expected reassembly delay with the state of the art approaches previously commented. We can classify all of them as methods which analyze complete segments to detect incomplete messages, or methods which are based on buffers with timeouts. For each category, we can see that:

- On the one hand, our method reduces the delay of methods that analyze complete segments because it only requires the information present in the TCP header. It shrinks the portion of data which is inspected per packet to detect segmented messages and, consequently, reduces the computational cost of such detection.
- On the other hand, our method only adds delay to the acquisition of such packets containing messages which have been filtered and detected as potentially segmented at TCP level. Then, it improves the expected delay of methods which are based on buffering with timeout.

We also note that the time until a reassembled message is delivered to the upper analysis layer is minimal, given that the trigger for the completion test is the reception of the last segment of the message.

B. Measuring the TCP Segmentation Impact

As a first step during the evaluation of our method's accuracy, we measure the impact of TCP segmentation in voice service detection and analysis. To do so, we have used `tshark 1.10.6` [18] as ground truth. This tool is able to reconstruct TCP streams and detect segmented application messages, while it is not usable for on-line monitoring tasks in demanding scenarios —given its latency and performance. Hence, we follow a forensic approach using captured traces and we consider the following segment sets as they appear in the formulation of our algorithm:

- N, the number of TCP segments with data from a signaling protocol. We denote this set of segments as $\mathcal{N}$.
- S, the number of TCP segments with an incomplete signaling message —detected with `tshark`. We represent this set as $\mathcal{S}$. We note that this tool marks as segmented the last segment which contains a part from the signaling message.
- P, the number of TCP segments with payload and the PUSH flag set to 0. The corresponding set is denoted with $\mathcal{P}$.

With those sets, we define the following metrics as indicators that characterize the effect of TCP segmentation and performance of the algorithm, respectively:

1) The ratio S/N, as it indicates the proportion of segments with incomplete messages of voice signaling protocols.
2) The ratio P/S, as it indicates the number of incomplete application messages detected by the previously described algorithm.

The consideration of set P is related to the semantic of the PUSH flag: when set to 1, it indicates that the receiver

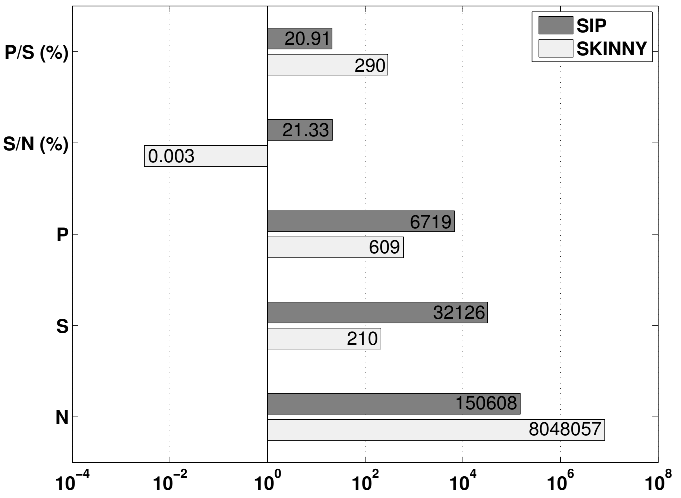

Fig. 4. Comparison between the cardinal of the sets and values of the metrics involved in the characterization of segmentation impact for SIP and Skinny.

application should pass the data as soon as received [19]. We hypothesize that segments with that flag set to 0 and carrying application data are linked to a block of one or more incomplete messages [10], [11], thus P is the set of TCP segments which are candidates to contain segmented messages.

To evaluate points (1) and (2) in a controlled environment, we have collected traffic traces from two enterprise voice deployments, with SIP and Skinny (also known as SCCP) as signaling protocol respectively. Our objective is also to study the differences between the impact of TCP segmentation on two protocols with different characteristics:

- **SIP** is a text-based protocol, with relatively long messages. SIP is an open protocol and is highly accepted as a *de facto* standard for VoIP signaling.
- **Skinny** is a binary protocol, with relatively short messages. It is a proprietary protocol, currently used by Cisco terminals.

Following the results we present in Fig. 4, which includes the value of N, S and P restricted to SIP and Skinny protocols in our traffic traces, we can see that the impact of TCP segmentation is much more important in the case of SIP than in the Skinny one —21.33% and 0.003% respectively. The results for SIP messages confirm the requirements manifested in [7], while the proportion of Skinny segmented messages is relatively low —this is an expected result, as SIP messages are larger and so the probability of segmentation is.

C. Reduction of segmented messages

Not only the number of segments in sets $\mathcal{N}$, $\mathcal{S}$ and $\mathcal{P}$ is important but also the existing relation between such segments to reassemble segmented application messages using transport-level information. To illustrate this, we evaluate the relation between TCP segments in each set to estimate the partial mitigation of segmented application messages that our algorithm produces.

TABLE I
SUMMARY OF RESULTS.

Protocol	Detected Segments	Reassembled Messages (%)
SIP	6719	20.91
Skinny	167	79.52

The relations $\mathcal{S} \subset \mathcal{N}$ and $\mathcal{P} \subset \mathcal{N}$ are clear by the definition of these sets. Nonetheless, there are no inclusion relations between sets $\mathcal{S}$ and $\mathcal{P}$ given their definitions and the results of the detection of segmented application message by `tshark`. After our empirical analysis, we have detected that, if we consider two segments $Segment_n, Segment_{n+1}$ in a unidirectional TCP stream between two given hosts, then:

$$Segment_n \in \mathcal{P} \Rightarrow Segment_{n+1} \in \mathcal{S}$$

Thus, the application of our heuristic leads to a reduction of segmented application messages that are considered by any application-oriented monitoring tool. This relation is also observable in the Skinny traces, but there is not such a direct evaluation as certain Skinny messages are not well interpreted by the `tshark` dissector —Skinny is a proprietary protocol, which makes it difficult to get a fully operational dissector. Hence, we have manually tested those messages which were not correctly detected by the dissector, so our accuracy evaluation is robust against this limitation.

Apart from this qualitative consideration, we also provide some figures about the effect of including our solution during the analysis of the traffic collected from enterprise VoIP deployments. The results show that our proposal reduces the impact of the segmented application messages in approximately a 21% and an 80% for SIP and Skinny, respectively —a summary of the obtained results is presented in Table I. We note that in this table, P is restricted to SIP and Skinny messages that are correctly detected by the corresponding `tshark` dissector, which is the cause of the differences between the values in this table and the included in Figure 4.

V. CONCLUSIONS AND FUTURE WORK

The segmentation of application messages as a result of the TCP stack behavior is a problem that must be taken into account during network traffic analysis. Actually, if monitoring applications do not include countermeasures, this process can produce erroneous results. In many situations, this fact leads to biased conclusions when studying the behavior of services deployed over TCP/IP networks.

We have evaluated the impact of such segmentation in the VoIP monitoring domain, where the consequent information losses produce important errors during on-line monitoring. We have presented the results of our experience after the application of a general methodology to mitigate its impact on traffic analysis which exploits the relation between segmented messages and TCP flags.

We have analyzed the expected latency of such approach by comparing it with the corresponding to other approaches in the literature. Furthermore, we have also provided the community with some figures on the impact of segmentation for the main VoIP signaling protocols. Particularly, the measured effect is of $<< 1\%$ and $\approx 20\%$ for SIP and Skinny messages, respectively, in real traffic traces from two different enterprise deployments. Additionally, we have assessed the attenuation of this matter after applying our algorithm, which we estimate in a $\approx 21\%$ in the case of SIP messages, and an $\approx 80\%$ for Skinny ones.

Based on this experience, our future lines of work are related to the implementation and performance evaluation of an optimized version of this algorithm —currently, we are carrying out further quantitative evaluations of the computational cost of a partial reassembly of TCP streams. With that implementation, we plan to obtain more accurate estimations of the effect of TCP segmentation not only in voice services, but also in other protocols that use this transport protocol.

ACKNOWLEDGMENT

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under the project TRÁFICA (MINECO/FEDER TEC2015-69417-C2-1-R).

REFERENCES

[1] T. Harbaum, M. Zitterbart, F. Griffoul, J. Rothig, S. Schaller, and H. Stuttgen, “Layer 4+ switching with QoS support for RTP and HTTP,” in *Proceedings of the IEEE Global Telecommunications Conference*, ser. GLOBECOM, vol. 2, 1999, pp. 1591–1596 vol.2.

[2] J. Novak and S. Sturges, “Target-based TCP stream reassembly,” *Sourcefire, Incorporated*, 2007.

[3] S. Dharmapurikar and V. Paxson, “Robust TCP stream reassembly in the presence of adversaries,” in *Proceedings of the Conference on USENIX Security Symposium - Volume 14*, ser. SSYM, 2005, pp. 65–80.

[4] G. Wagener, A. Dulaunoy, and T. Engel, “Towards an estimation of the accuracy of TCP reassembly in network forensics,” in *Proceedings of the International Conference on Future Generation Communication and Networking - Volume 02*, ser. FGCN, 2008, pp. 273–278.

[5] N. T. Tran, S. Tomiyama, S. Kittitornkun, and T. H. Vu, “TCP reassembly for signature-based network intrusion detection systems,” in *Proceedings of the International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, ser. ECTI-CON, 2012, pp. 1–4.

[6] P. Matoušek, J. Pluskal, O. Ryšavý, V. Veselý, M. Kmeť, š. Karpí, Filip, and M. Vymlátil, “Advanced techniques for reconstruction of incomplete network data,” in *Digital Forensics and Cyber Crime*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 157, 2015, pp. 69–84.

[7] J. L. García-Dorado, P. M. Santiago del Río, J. Ramos, D. Muelas, V. Moreno, J. E. López de Vergara, and 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*, vol. 24, no. 3, pp. 181–199, 2014.

[8] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, “Is it still possible to extend TCP?” in *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*, ser. IMC, 2011, pp. 181–194.

[9] V. Moreno, P. M. Santiago del Río, J. Ramos, D. Muelas, J. L. García-Dorado, F. J. Gómez-Arribas, and J. Aracil, “Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems,” *International Journal of Network Management*, vol. 24, no. 4, pp. 221–234, 2014.

[10] J. Postel, “RFC 793: Transmission Control Protocol,” 1981.

[11] R. Braden, “RFC 1122: Requirements for Internet hosts,” 1989.

[12] P. Shinde, A. Kaufmann, T. Roscoe, and S. Kaestle, “We need to talk about NICs,” in *Proceedings of the USENIX Conference on Hot Topics in Operating Systems*, ser. HotOS, 2013, pp. 1–1.

[13] C. Benvenuti, *Understanding Linux network internals*. O’Reilly Media, Inc., 2006.

[14] L. Rizzo, “Revisiting network I/O APIs: The Netmap framework,” *Communications of the ACM*, vol. 55, no. 3, pp. 45–51, Mar. 2012.

[15] R. Bolla, R. Bruschi, O. Jaramillo Ortiz, and R. Rapuzzi, “Enabling the TCP segmentation offload to save energy,” in *Tyrrhenian International Workshop on Digital Communications - Green ICT*, ser. TIWDC, Sept 2013, pp. 1–5.

[16] V. Moreno, J. Ramos, P. Santiago del Rio, J. García-Dorado, F. Gomez-Arribas, and J. Aracil, “Commodity packet capture engines: Tutorial, cookbook and applicability,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, pp. 1364–1390, thirdquarter 2015.

[17] V. Duarte and N. Farruca, “Using libpcap for monitoring distributed applications,” in *Proceedings of the International Conference on High Performance Computing and Simulation*, ser. HPCS, 2010, pp. 92–97.

[18] G. Combs, “Tshark-the wireshark network analyser.” [Online]. Available: http://www.wireshark.org

[19] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.