High-Speed Analysis of SMB2 File Sharing Traffic without TCP Stream Reconstruction

Eduardo Berrueta*, Daniel Morato*†, Eduardo Magana*†, Mikel Izal*†
*Dept. of Electrical, Electronic Engineering and Communications, Public University of Navarre, Pamplona, Spain
†Institute of Smart Cities, Pamplona, Spain
email: {eduardo.berrueta, daniel.morato, eduardo.magana, mikel.izal}@unavarra.es

***Abstract*—This paper presents a file sharing traffic analysis methodology for Server Message Block (SMB), a common protocol in the corporate environment. The design is focused on improving the traffic analysis rate that can be obtained per CPU core in the analysis machine. SMB is most commonly transported over Transmission Control Protocol (TCP) and therefore its analysis requires TCP stream reconstruction. We evaluate a traffic analysis design which does not require stream reconstruction. We compare the results obtained to a reference full reconstruction analysis, both in accuracy of the measurements and maximum rate per CPU core. We achieve an increment of 30% in the traffic processing rate, at the expense of a small loss in accuracy computing the probability distribution function for the protocol response times.**

***Index Terms*—File sharing, traffic analysis, response time**

I. Introduction

The use of network file sharing systems is a common practice in the corporate environment [1]. User and workgroup files are stored in disk arrays on Network Attached Storage (NAS) systems. This architecture simplifies collaborative work and the application of backup policies for critical documents. Microsoft Windows, in its several versions, is the *de-facto* Operating System (OS) for user desktop computers, and SMB is the default file sharing protocol for this OS.

The first version of SMB was considered a very chatty protocol. It was deprecated in June 2013 by Microsoft, due also to its numerous vulnerabilities. Windows Vista established SMB version 2 (SMB2) as the default file sharing protocol, getting better performance and security features [2]. Finally, SMB3 replaced SMB2 as the default version in Microsoft Windows 8. SMB3 uses the same SMB2 messages, adding an encryption layer. The version used is negotiated on session establishment and both endpoints (host and NAS filer) must support SMB3 in order to be used. As far as we have witnessed in several corporate environments, SMB2 is still the prevalent version and even SMB version 1 is being used (as it has been reported in many cases of malware using SMB1 vulnerabilities [3]). What we present in this paper about SMB2 applies also to SMB3, provided the encrypted messages can be decrypted using the private keys or shared secrets.

The performance of a file server is critical for the productivity in a collaborative environment because users expect a behavior similar to that obtained from a local disk. The responsiveness they perceive is the result of the file sharing protocol design, the effects of network transport and the response time from the hardware and OS at the NAS. From a user's standpoint, the whole system can be evaluated by the response time the user measures for each SMB command his computer sends to the NAS.

The most common non-intrusive methodology to measure response times in network protocols is the analysis of passively monitored network traffic between client (the user) and server (the NAS). Due to the multi-gigabit per second rates present in the corporate LAN, efficient traffic analysis is required in order to obtain measurement results in a timely manner using an affordable hardware. A basic measure of analysis performance is the maximum rate of packets or bits per second per CPU processing core. A high processing rate could be achieved by low precision or biased measurement techniques. Therefore, an evaluation of the accuracy in the measurement is required, as it presents a trade-off with the analysis rate.

In this paper we present the design of a traffic analysis procedure capable of dealing with several gigabits per second of SMB file sharing traffic per CPU core, in close to real time. It is based on an approximate methodology where not every command is analysed. We evaluate the trade-off in accuracy in the analysis, comparing the results to those obtained from a perfect history reconstruction of protocol messages. Compared to a traffic analysis based on perfect TCP stream reconstruction, we obtained more than 30% increase in traffic analysis rates. Meanwhile, for network scenario with 1% packet losses we maintained an error less than 0.53% for the estimation of the 99.9 percentile of the response time distribution and of 6.1% for a scenario with 3% packet losses.

The rest of the paper is organized as follows: Section II presents the SMB2 protocol and the previous work about its analysis. Section III describes the perfect reconstruction and the approximated methodologies. In Section IV we describe the measurement scenario and the results from the performance evaluation are presented in Section V. Finally, Section VI concludes the paper.

II. SMB protocol analysis

A. Related work

The scientific literature about SMB protocol is focused on the evaluation of its performance and the optimization for efficient file access. For example, the authors in [4] recently compared the read and write speed for SMB version 1 and internet Small Computer System Interface (iSCSI), using RAID0 and RAID1 disk configurations.

Although there have been previous works that measure the response time of servers using SMB, they were focused on protocol and server performance, and not in the design of the analysis tool. For example, in [5] the authors highlighted the importance of response time analysis but they did not deal with the problem of real time analysis. They were only concerned with detecting problems due to high disk load. However, a monitoring tool working in a production environment must be capable of producing these measurement results in a timely manner even when several gigabits per second of file sharing traffic are produced.

Bro [6] is a well known software solution capable of SMB2 protocol analysis. However, it is focused on the detection of malicious access to network shared volumes and the lateral movement of some malware using SMB vulnerabilities. It is not designed for the evaluation of response times to SMB commands.

Tools like Wireshark and TShark can be considered the reference open source tools for traffic analysis. They are capable of SMB2 analysis and they do compute response times between protocol requests and responses. However, they are tools designed for the analysis of small traffic traces. They are not capable of dealing with a continuous stream of several gigabits per second of traffic [7], both in terms of CPU and RAM usage. They are also unable to decode SMB2 messages under certain segmentation circumstances (see section III).

In the presence of packet losses, application level analysis of protocols over TCP requires the reconstruction of the continuous transport stream in order to provide correct results. This can be a problem for network monitoring tasks with strict latency and synchronous requirements [8].

We have not found any previous work on the design of traffic analysis algorithms for SMB2 based on avoiding TCP stream reconstruction. We propose a methodology that obtains highly accurate analysis results with a much reduced cost in CPU usage (resulting in faster analysis using a single CPU core). In order to evaluate the performance gain, we have also implemented a response time SMB2 analysis program based on full TCP stream reconstruction.

B. SMB2

SMB2 uses a single TCP connection between client and server for all the network volumes accessed from the client. The protocol uses a request-response architecture, with some asynchronous notification capabilities. Each request is identified by a numeric *messageID*, which is piggybacked in the response and is used to pair it to the request.

SMB2 supports pipelining, which means that several requests can be sent before a response to any of them is received. The responses are not required to follow the same ordering of the requests. This last feature makes pipelining in SMB very different to for example pipelining in version 1.1 of Hypertext Transfer Protocol (HTTP)[1], where the ordering in the responses must repeat the ordering in the requests. Therefore, although previous works exist on efficient HTTP analysis [7] they do not apply to the SMB scenario.

[1]Pipelining is not a feature of HTTP/2 as it supports stream multiplexing

The degree of pipelining in SMB2 is controlled by the server using a credits system. The client usually consumes one credit for each request command and the server provides new credits in the response [9]. The client can keep as many requests sent to the pipeline as credits are available.

Pipelining in SMB2 is prevalent and a fundamental protocol characteristic to obtain high file transfer throughput. Large read or write operations instruct large bursts of read or write commands, creating a long pipeline. A traffic analysis software must deal with large numbers of requests in the pipeline, pairing responses to requests in an efficient manner.

III. Software structure

In this paper we describe and evaluate two methods for the measurement of response times in SMB2. The main difference between these two methods is the reconstruction of the TCP stream. The method named *smbtime* reconstructs the TCP stream and is able to decode every SMB2 message present. The module named *smbtimefast* does not reconstruct the TCP stream. Packet losses, disorders and retransmissions in TCP cause some SMB2 messages to not be detected using *smbtimefast*. The result is a loss of response time samples and therefore a probable reduction in the accuracy measuring the probability distribution function of the response time.

Both methods analyse data as soon as they are available. In the case of *smbtimefast* they are available right on packet arrival, while using *smbtime* they could require being stored waiting for the arrival of segments that fill gaps in the TCP sequence. Due to the presence of pipelining in SMB protocol they also require maintaining a list of pending requests. When a new response message arrives, the corresponding request must be located in the dynamic data structure. Both methods use the same management for this data structure in order to provide a fair comparison. Searching for the corresponding request can reduce the processing rate as pipelining increases and the dynamic structure grows.

We take the results from *smbtime* as the ground truth values we want to measure. We allow an infinitely deep list of pending requests in *smbtime* in order to support any degree of pipelining. Due to the TCP stream reconstruction and the unlimited list of pending requests, the module *smbtime* computes the response time for every request issued (be it a successful response or an error response). We could not use well-known tools such as TShark or Wireshark as the ground truth providers because we detected missing messages in the results of their analysis. Using TShark 3.0.0 we discovered at least two scenarios where it drops SMB messages: when the SMB2 header is fragmented between two TCP segments and in some cases of TCP disorders. In certain situations, the loss of a message can even create a cascade effect, losing several of the following messages in the analysis provided by TShark.

A. *smbtime*

For the *smbtime* method we implemented full TCP stream reconstruction. When a TCP segment that continues the sequence is received, its data are readily available for analysis. When the new segment creates a gap in the TCP sequence, the packet is kept in memory in a linked list, waiting for the packets that fill the gap. Several packets could be received in a burst after the gap. All of them are kept in memory and they cannot be offered for analysis because data is missing in the stream. When the missing gap is filled, the whole consecutive sequence of application level bytes is delivered to the analysis method. Even though some data were available earlier, the end host could not deliver the data to the SMB client/server, therefore we cannot use each packet timestamp when there is disorder, but the time when the sequence becomes continuous.

We focus on measuring time from a user standpoint, therefore we take the time between the first packet in the request to the last packet in the response.

We have validated the implementation of stream reconstruction software and the SMB message dissection procedure creating an extension of this module capable of extracting the content of files transferred using SMB [10].

B. *smbtimefast*

In order to measure the improvement in analysis time with the proposed approximations, we share as much code between *smbtimefast* and *smbtime* as possible. Both methods must read the traffic trace from file or from a network interface and both must decode the headers in the IP and TCP layers in order to group the segments from the same TCP connections.

smbtimefast takes the size of the message from each request or response header and computes the TCP sequence byte which corresponds to the end of the message. Packets with sequence numbers in between the SMB header and the end of the message can be ignored. Packet losses in the burst of packets from the response to a message reading from disk are not a problem if the beginning and end are present. If the end of the message is lost and further packets are received, *smbtimefast* losses track of the stream. It could keep information about the stream and wait for the retransmitted packets, however, this is precisely the stream reconstruction feature we are trying to avoid.

When losses or disorders create a gap at the end of an SMB message, the method enters a state from which it tries to recover by searching for a valid SMB header at the beginning of each TCP segment. A burst of messages could be lost if TCP joins application level messages into TCP segments, however, as soon as a small pause exists, the next message will appear at the beginning of a TCP segment and the analysis will be recovered.

The loss of a response message in the analysis (not in the network) results in a request in the list of pending requests that will never be paired with its corresponding response. The TCP connection used by SMB lasts as long as the volume is mounted or until the system is restarted. The analysis module can accumulate a large number of requests without response during this long lived connection. It is not only a matter of samples of response time lost but also an in memory data structure per TCP connection which only grows. Hence, we had to implement a limiting mechanism for the size of the pending requests lists. The larger the number of requests in the dynamic structure the longer the search times when a response arrives, therefore the effect of maximum list size could impact the processing rate.

We set an expiration time T_{th} for requests in the pending requests list and a maximum number N of requests in this data structure. In the following sections we evaluate the impact of these parameters in accuracy and performance.

IV. Measurement scenarios

We consider two different measurement scenarios: a custom ad-hoc emulation scenario and a real case of a NAS appliance in a production environment. We use the emulated scenario in order to provide an in-depth analysis of the dependence of accuracy and speed results on the configured parameters and the losses in the network.

The emulation scenario is built using Virtual Machines (VMs) for client, server and for an intermediate virtual switch. We try to create a worst-case scenario based on high intensity file transfers created by the end user and network losses introduced at the packet switch. We measure the effect they have on both analysis methods.

The VMs are Windows 7 hosts. The shared directory contains 5,000 files with sizes between 1 and 100 MBytes. We expect a worst case behavior in accuracy and speed when a high degree of pipelining is present, therefore we simulate an intense use of the shared directory by reading, writing, renaming and deleting files from an automated script.

The virtual switch is a linux VM, where we configured the kernel using the *tc* command for independent and identically distributed (i.i.d.) packet losses. We repeated the procedure in order to create traffic traces in three network scenarios: no losses, 1%, and 3% i.i.d. packet losses.

The packet losses are only in the path between SMB client and server and no losses are introduced in the monitoring process. The network probe records every packet and retransmission at the virtual switch port where the client is connected.

The traffic trace from a production environment is the monitored traffic from the population of users in our university campus to a single NAS. More than 300 users and 400 TCP connections were registered during the working hours of a single day, with an average connection duration close to 6 hours. We removed short connections, used for other services over SMB. No artificial losses were introduced, as the traffic was passively monitored using a network probe connected to a port mirror in an Ethernet switch.

Table I shows a summary of the macroscopic characteristics of the traces.

V. Results

In this section we compare the results obtained from both analysis procedures. We take *smbtime* response time results

TABLE I
DESCRIPTION OF TRAFFIC TRACES

	1dayCampus	Emulated 0% losses	Emulated 1% losses	Emulated 3% losses
Duration	8h40min	3h27min	11h49min	53h24min
Size (Gbytes)	212	141	147	147
SMB connections (Client hosts)	401 (330)	1 (1)	1 (1)	1 (1)
Avg. connection duration	5h55min	3h27min	11h49min	53h24min
Avg. files opened per connection	7640	44558	46704	46732
Avg. operations per connection	41.9 (Read) 50.2 (Write)	1.11M (R) 1.07M (W)	1.15M (R) 1.08M (W)	1.12M (R) 1.07M (W)
Max. pipelining	257	160	157	120

as the ground truth and we measure the accuracy in *smbtimefast*. We do not compare the response times on a one-on-one basis, as there is a different amount of samples with each methodology. We compare the probability distribution functions of response time obtained from each one. Taking *smbtime* as the ground truth, we want the distribution obtained from *smbtimefast* to be as close as possible, specially for the range of large response times, where the anomalies in system behaviour should be located. This accuracy result is presented in subsection V-A.

The perfect measurement obtained from *smbtime* is at the expense of a higher processing time than *smbtimefast*. This means that a lower traffic rate can be processed by the same hardware using the exact procedure. Using *smbtimefast*, although there is a loss in accuracy, there is an increase in processing speed. In subsection V-B we evaluate how much faster is the approximate methodology.

Finally, in subsection V-C we validate the results in a different measurement scenario.

A. Accuracy results

We name T_{resp} the random variable describing the per SMB command response time. The parameter T_{th} limits the maximum T_{resp} measurable by *smbtimefast*. T_{th} should be configured to a value larger than the support of T_{resp}. This support depends on the maximum response time the NAS filer provides, which for extreme cases could be large (hundreds of milliseconds). Instead of being able to measure extreme cases, we will set a value of T_{th} that provides a small enough probability $P(T_{resp} > T_{th})$.

Figure 1 shows the Complementary Cumulative Distribution Function (CCDF) of T_{resp}, obtained using *smbtime*, and the approximation $\hat{T}_{resp}$, obtained from *smbtimefast* using $T_{th} = 1$ and N unbounded. The traffic trace used presents 0% extra packet losses. Both distributions are similar until values of T_{resp} close to T_{th}, where the result from *smbtimefast* must fall to zero. For $T_{th} = 0.9s$ the real value is $P(T_{resp} > 0.9s) = 0.28\%$ while for the estimated $\hat{T}_{resp}$ it is $P(\hat{T}_{resp} > 0.9s) = 0.10\%$. We must configure a larger value of T_{th} in order to obtain a better approximation for large response times.

Figure 2 shows the value of the 0.1 percentile $\hat{T}_{0.1}$ estimated from $\hat{T}_{resp}$, i.e. the value of $\hat{T}_{resp}$ such that $P(\hat{T}_{resp} > \hat{T}_{0.1}) = 0.001\%$. The larger the parameter T_{th} the better $\hat{T}_{0.1}$ estimates the real value obtained from *smbtime*. For $T_{th} > 2s$ the error margin is negligible. We select $T_{th} = 5s$ as the maximum response time; a value that should also deal with larger response times when losses are present in the network.

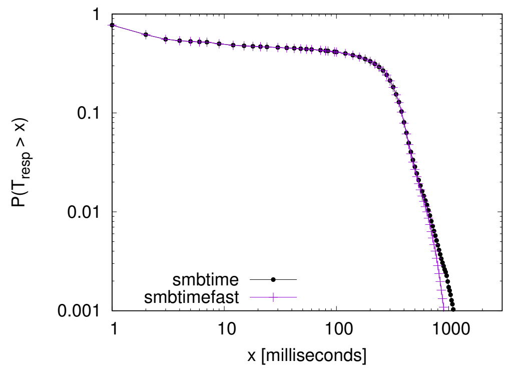

Fig. 1. CCDF of response time with T=1 and N unbounded. 0% packet losses

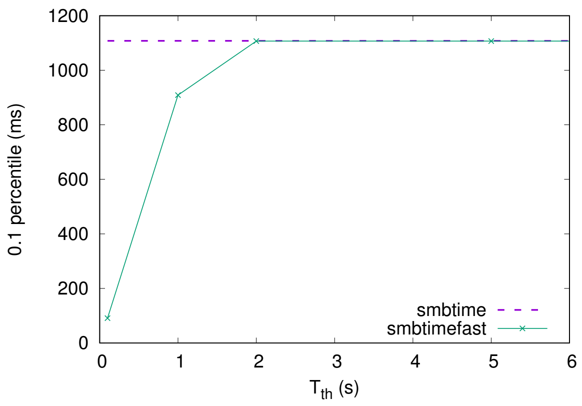

Fig. 2. Improvement in the approximation of the CCDF tail of the response time, measured at the 0.1 percentile. N unbounded. 0% packet losses

Figure 3 shows the effect of N on the estimation of the probability distribution function of T_{resp}. We use $T_{th} = 5s$ and a range of values of N up to the maximum degree of pipelining present in the traffic trace. For small values of N, large bursts of pipelined commands result in many commands lost in the processing method implemented by *smbtimefast* and a strong bias in the CCDF. We cannot accurately estimate the CCDF for small values of probability when there is pipelining and a small value of N is configured.

As N grows, the CCDF provided by *smbtimefast* gets closer to the ground truth. Values of N larger than the maximum pipelining present in the trace provide no improvement in the estimation, as no further commands can be recovered. The maximum degree of pipelining depends on the number of credits offered by the server and it is only reached when long files are accessed and/or many files are read/written simultaneously. In these experiments we tried to create an intense traffic and reached a maximum of 160 command in the pipeline.

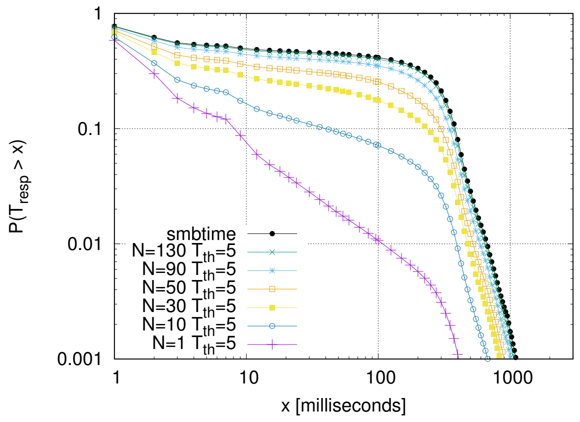

Fig. 3. CCDF of response time using different values of N in *smbtimefast*. 0% packet losses

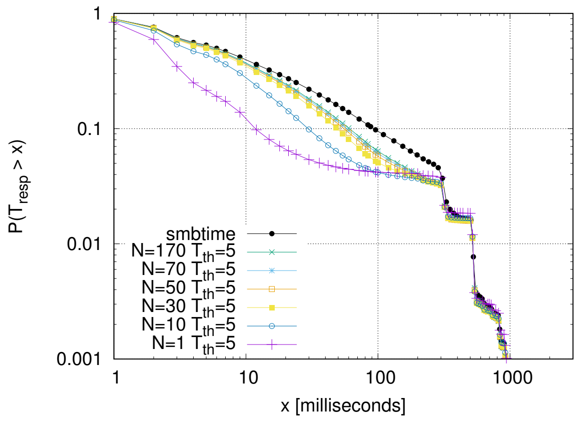

Fig. 4. CCDF of response time using different values of N in *smbtimefast*. 1% packet losses

For the scenario with 1% network losses, we used the conservative value of $T_{th} = 5s$. Figure 4 shows the CCDF for T_{resp} and $\hat{T}_{resp}$. Similar to the case without network losses, the accuracy improves as N grows and gets closer to the maximum level of pipelining. However, in this scenario, **the CCDF of $\hat{T}_{resp}$ never reaches the CCDF of T_{resp} for values of $T_{resp} < 300ms$**. The estimated CCDF is not accurate for low values of T_{resp}, while it provides a good estimation for the tail of the CCDF. We have explained that *smbtimefast* cannot measure the response time for every command, as those commands affected by network losses and retransmissions can be lost in the analysis process due to the lack of TCP stream reconstruction. As the tail of the distribution is accurately estimated, the lost measurements must be biased toward the small values of response times.

TCP detects losses by the triple acknowledgment mechanism or by the expiration of the retransmission timer. In both events it reduces the congestion window, therefore reducing the sending rate. A smaller congestion window reduces the maximum number of commands that could be pipelined, therefore $\hat{T}_{resp}$ reaches T_{resp} for smaller values of N than in the case without network losses. When losses and retransmissions occur, the *smbtimefast* method cannot measure correctly the response times. The minimum TCP retransmission timer is in the range of 200 ms to 300 ms, depending on the operating system. Samples of response time up to those values are affected by network losses and dropped by *smbtimefast*, which results in a worse estimation of the CCDF. For values of T_{resp} larger than the TCP retransmission timer, the reason for these values is not network losses but the response time from the NAS filer (disk access times). Those samples are lost in the same proportion as any other samples, without a strong bias from network losses, with the result of a good estimation of the probability values for large T_{resp}.

Figure 5 shows the effect of both N and network losses. We plot the value $\hat{T}_{10}$ of $\hat{T}_{resp}$ which provides $P(\hat{T}_{resp} > \hat{T}_{10}) = 0.1$. As N grows, the estimated probability value gets closer to the real one, as the cases of pipelining are better measured. However, it reaches a limit where the dominant effect comes from the messages lost due to network packet losses and the lack of TCP stream reconstruction.

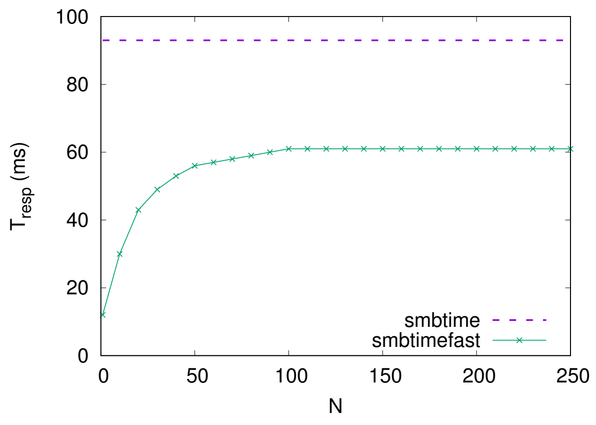

Fig. 5. Improvement in the approximation of the CCDF of response time, measured at the 10 percentile. $T_{th} = 5s$. 1% packet losses

B. Improvement in processing speed

In order to compare the maximum processing rate achievable by *smbtimefast* compared to *smbtime*, we have run 50 experiments using each one of the traffic traces described in Table I. We have configured $T_{th} = 5s$ and we have tried different values of N in order to evaluate its impact on processing speed. The larger the pipeline, the more expensive it is to keep and locate in memory the corresponding request to a new response.

We used a computer with an Intel Xeon E5-2609 @ 1.7GHz CPU. One CPU core was dedicated to reading the trace file and the second one ran the analysis code. In order to reduce the effect of reading the trace from secondary storage we used a Solid State Disk (SSD) which offered reading rates of at least 8.7 Gb/s.

Figure 6 shows the total time required to process the whole 141 GB trace from the scenario without network losses. We also added the 95% confidence interval. We immediately observe at least a 30% reduction in processing time compared to *smbtime*. Taking the whole trace size, the average processing rate using *smbtime* is 4.6 Gb/s, while using *smbtimefast* it reaches an average of 7.1 Gb/s. In terms of RAM consumption, *smbtime* analysed the whole trace using around 200KB of RAM, and *smbtimefast* decreased the RAM usage to 21.5KB. In comparison, TShark was using 10GB of RAM when only 7.3GB had been processed, increasing steadely its RAM usage.

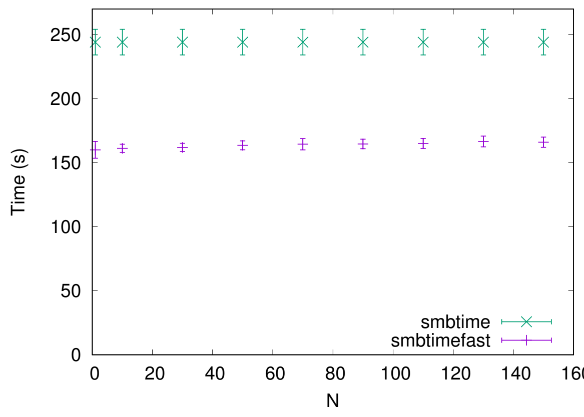

Fig. 6. Processing time using both methods. $T_{th} = 5s$ and various values of N. 0% network losses

As the value of N is increased, the average computation time using *smbtimefast* slowly increases, due to the longer lists of pending messages. However, the average processing time stays within the error interval for $N = 1$.

Similar results are obtained for the traces from the scenarios with network losses. Table II shows a summary of the processing speed obtained for each trace using reconstruction or the approximate method. The effect of N is negligible for all scenarios.

TABLE II
PROCESSING SPEED IN THE EMULATED SCENARIO

Losses	Average processing speed (Gbps)		Improvement
	smbtime	smbtimefast	
0%	4.62 ± 0.18	6.85 ± 0.17	$48.2\% \pm 3.07\%$
1%	4.78 ± 0.11	6.51 ± 0.17	$36.19\% \pm 1.77\%$
3%	4.72 ± 0.17	6.17 ± 0.15	$30.7\% \pm 1.85\%$

C. Validation in a production scenario

We validated the results using a traffic trace from an in-production NAS filer at our university campus network. We captured the traffic during 8 office hours, obtaining 212 GB of traffic (Table I). We do not include the graphical results due to space constraints, but the results are consistent with previous experiments. The larger the value of N, the better the estimation, getting close to stabilization when it reaches the maximum degree of pipelining present in the trace (257 pending requests).

The CPU that ran the analysis software was an Intel Xeon E5-2630 @ 2.3GHz. The average processing speed was 6.8 Gb/s using *smbtime* and it increased to 8.3 Gb/s when *smbtimefast* with $N = 200$ was used (22% improvement). The RAM consumption for both proposed methodologies was less than 1 GByte.

VI. CONCLUSIONS

We have evaluated an approximated methodology (*smbtimefast*) for the analysis of response times of NAS filers that use the SMB protocol. *smbtimefast* requires two parameters that control the number of samples retained in the event of pipelining. A timer value of T_{th} larger than the support of response times T_{resp}, or at least larger than the range of values of interest, is required. It also requires a maximum number of analysed in-flight messages N close to the maximum degree of pipelining present, in order to provide an accurate estimation.

We have measured the reduction in accuracy from the approximated method when packet losses are present in the network. Even in the range of 3% packet losses, the accuracy for the events of high response times is maintained and it is only reduced in the range below the TCP minimum retransmission timer value.

Compared to a full TCP stream reconstruction, the processing speed improvement obtained from the approximated method is at least 30%, which allows processing 7 Gb/s of traffic in real time using a single CPU core running the analysis task.

ACKNOWLEDGMENT

This work was supported by Spanish MINECO through project PIT (TEC2015-69417-C2-2-R).

REFERENCES

[1] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A First Look at Modern Enterprise Traffic. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[2] Windows Explorer and SMB Traffic, September 21st 2007. https://blogs.technet.microsoft.com/askperf/2007/09/21/windows-explorer-and-smb-traffic/, Last access March 10, 2019.

[3] Tom Roeh. Shut Down SMBv1, Already!. ExtraHop. https://www.extrahop.com/company/blog/2017/shut-down-smbv1/, Last access May 17, 2019.

[4] M. Simeunovic, B. Djordjevic, V. Timcenko, and N. Jankovic. iSCSI and CIFS Network Disks for RAID0/RAID 1 Configurations. In *2018 26th Telecommunications Forum (TELFOR)*, pages 1–4, Nov 2018.

[5] Karl L. Swartz. Adding Response Time Measurement of CIFS File Server Performance to NetBench. In *Proceedings of the USENIX Windows NT Workshop on The USENIX Windows NT Workshop 1997*, NT'97, pages 12–12, Berkeley, CA, USA, 1997. USENIX Association.

[6] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[7] Carlos Vega, Paula Roquero, and Javier Aracil. Multi-Gbps HTTP traffic analysis in commodity hardware based on local knowledge of TCP streams. *Computer Networks*, 113:258 – 268, 2017.

[8] D. Muelas, J. E. L. de Vergara, J. Ramos, J. L. Garca-Dorado, and J. Aracil. On the impact of TCP segmentation: Experience in VoIP monitoring. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 708–713, 2017.

[9] Microsoft Corporation, Server Message Block (SMB) Protocol versions 2 and 3. https://msdn.microsoft.com/en-us/library/cc246482.aspx, Last access May 9, 2018.

[10] E. Berrueta, D. Morato, E. Magaña, and M. Izal. Ransomware Encrypted Your Files but You Restored Them from Network Traffic. In *2018 2nd Cyber Security in Networking Conference (CSNet)*, pages 1–7, Oct 2018.