Low-cost and High-performance: VoIP monitoring and full-data retention at multi-Gb/s rates using commodity hardware

José Luis García-Dorado, Pedro M. Santiago del Río, Javier Ramos, David Muelas, Victor Moreno, Jorge E. López de Vergara, and Javier Aracil

High Performance Computing and Networking research group,Universidad Autónoma de Madrid, Spain

NOTE: This is a version of an unedited manuscript that was accepted for publication. Please, cite as:
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, Wiley, Vol. 24, No. 3, pp. 181-199, May 2014.
The final publication is available at:
`http://dx.doi.org/10.1002/nem.1858`

Abstract

Voice over IP (VoIP) is increasingly replacing the old Public Switched Telephone Network (PSTN) technology. In this new scenario, there are several challenges for VoIP providers. First, VoIP requires a detailed monitoring of both users' quality of service (QoS) and experience (QoE) to a greater extent than in traditional PSTNs. Second, such monitoring process must be able to track VoIP traffic in high-speed networks, nowadays typically of multi-Gb/s rates. Third, recent government directives require that providers retain information from their users' calls. Similarly, the convergence of data and voice services allows operators to provide new services such as full-data retention, in which users' calls can be recorded for either quality assessment (call-centers, QoE), or security purposes (lawful interception). This implies a significant investment on infrastructure, especially on large-scale networks which require multiple points of measurement and redundancy. This paper proposes a novel methodology, architecture and system to fulfill such challenges, called *VoIPCallMon*, as well as the data structures and necessary hardware-tuning knowledge to its development. As distinguishing features, *VoIPCallMon* provides very high performance being able to process VoIP traffic on-the-fly at high bitrates, novel services, and significant cost reduction by using commodity hardware with minimal interference with operational VoIP networks. The performance evaluation shows that the system copes with the VoIP load of real-world operators. We further evaluated the system performance at a full-saturated 10 Gb/s link and no packet loss was reported, therefore demonstrating the potential of commodity hardware solutions.

Keywords: 2. Service Management ; 2.1 Multimedia service management (VoIP).

1 Introduction

Voice over IP (VoIP) has proven to be a mature technology that provides multiple advantages for both telecom operators and users. On the one hand, it allows providers the convergence of their voice and data services into a single network infrastructure. On the other hand, users typically enjoy lower invoices and extended service offering. VoIP technology was developed more than a decade ago and has received much attention by the research community [1]. However, it is in the recent years when VoIP has begun to gain ground to the old Public Switched Telephone Network (PSTN). This increase has been strengthened by multiples examples of successful implementations of large-scale VoIP networks:

KT Corporation, formerly Korea Telecom, had 2.1 million VoIP subscribers [2] in year 2011. As the authors in [3] show, Fastweb, one of the main Internet providers in Italy, which exclusively offers VoIP telephony, had a number of customers larger than 600,000 in year 2010. Similarly, Telefónica offers in Spain VoIP services to its corporate clients at reduced prices, while PSTN services are still dominant for residential users. Furthermore, it is expected that the number of residential VoIP customers will sharply increase in the near future. In this exciting scenario, providers face some challenges for the successful deployment of VoIP systems in large-scale environments. First, like any other multimedia service, VoIP requires an exhaustive monitoring of the quality of service (QoS) received by the users as well as their quality of experience (QoE) in order to provide the same level of quality than PSTN [4].

Second, such monitoring system must be able to track VoIP traffic in high speed networks. Currently and in the near future, there is a need for traffic monitoring at 10 Gb/s interfaces and even faster ones, given the ever-increasing growth of the data transmission capacity [5]. Even more so when a monitoring system has to be fast enough to cope with traffic peaks (and not only with the mean or median) and operators carry other traffic besides VoIP calls (increasing the aggregated rates).

As a third challenge, we note that new data-retention directives are being developed in Europe [6] as well as in the US (CALEA [7]) and other countries. These directives require providers to store certain information of the calls carried out by their clients and a set of monitoring requirements inherited from PSTN regulation. Such information includes the identification of caller and callee as well as the call start and end times. Moreover, more demanding directives may be established in the future, and even include some details of the content due to national security reasons.

Finally, the entire VoIP call, and not only some descriptors, may be stored in large-scale databases if proper indexing policies are designed and with the consent of the customer. For example, providers may record users' complaints and by-phone contracts of their clients. Similarly, this is useful to evaluate the quality of subcontracted call centers, which are not under the operator direct control, and are extremely popular nowadays. Furthermore, businesses may be willing to record their calls for the evaluation of customer satisfaction. Note that this is a novel monitoring service that can be offered to customers. In addition, operators can assess if the perceived QoE of a given user is adequate by effectively contrasting the QoS and QoE once the user conversation has been reconstructed.

In this light, this paper proposes a novel methodology, architecture and system to fulfill such challenges, called *VoIPCallMon*, as well as the data structures and hardware-tuning knowledge necessary to its development. *VoIPCallMon* is available on demand at [8]. The system is composed of four modules which are in charge of i) capturing traffic at multi-Gb/s rates, ii) identifying and tracking VoIP traffic, iii) generating the statistics in compliance with data retention directives and those required to ensure users' QoS and, iv) reconstructing and indexing the VoIP calls to provide novel services based on call recording.

As a distinguishing feature from the best known vendors' solutions [9], *VoIPCallMon* faces the deployment of monitoring and data retention of VoIP networks from the point of view of the cost that such tasks imply. That is, in large-scale networks many points of measurements are required, which multiplies the infrastructure investment. Some studies [6] have deemed such cost in the range of hundreds of million dollars. Therefore, providers should focus on approaches that maximize their monitoring capacity but, at same time, cut the investment down. While higher processing rates can be achieved with specialized hardware, it typically lacks of the required flexibility to include new traffic statistics or specialized analysis, and the cost is high. Thus, the use of commodity hardware turns out to be an interesting option that combines good performance and limited cost [10]. For example, the authors in [11] pinpoint the relevance that commodity hardware is gaining in both governments and militaries for their computing needs. *VoIPCallMon* uses commodity hardware to perform all its tasks. Similarly, another important issue is the cost of the integration of the monitoring system in operational VoIP networks. This calls for monitoring systems whose interference with current VoIP architectures is minimal. This is the case of *VoIPCallMon*, which only requires a probe fed with the traffic to monitor without any additional network configuration thanks to its capacity to analyze traffic at high speed. However, the use of commodity hardware to multi-Gb/s networking tasks requires tuning the operating system's networking stack, essentially, for efficient memory management and packet processing, hardware interactions at low-level, and programming optimization.

This paper reviews such challenges about commodity hardware and shows how to overcome them. Specifically, we propose a modified network driver that, differently from other proposals, focuses on accurate timestamps. This is of paramount importance for VoIP monitoring, but keeping pace with high performance. Then, we explain the methodology and data structures followed to the implemen-

tation of a VoIP monitor at application layer, which allows dealing with the rates the hardware level provides.

For performance evaluation purposes, we obtained some SIP (Session Initiation Protocol) and RTP (Real-time Transport Protocol) traces from a VoIP operator that provides service in Spain. Additionally, we were also informed of the traffic intensity of its VoIP network. With this information along with other public measurements such as [3], we have assessed the performance of our system. These traces have also allowed us to encounter some unexpected problems when analyzing real VoIP traffic as will be shown in what follows. We think this can turn out beneficial for other practitioner and researchers in the area.

The results show that this approach not only complies with the requirements of real VoIP scenarios but far exceeds them. Consequently, in order to find the performance bounds of the system, we employ a controlled testbed, in which the VoIP traffic is generated using Sipp application[1]. This application allows us to create synthetic VoIP traffic tuning the time between consecutive calls and their durations according to probabilistic models [3, 12]. We aggregate the traffic from several sources running Sipp and additional cross traffic to achieve multi-Gb/s transmission rates by means of a 10 Gb/s switch. Specifically, we generate near 9 Gb/s VoIP traffic and add cross traffic until the 10 Gb/s link is saturated. Then, this traffic is redirected via port mirroring to *VoIPCallMon*. At this rate, the system was able to manage the VoIP calls with no packet loss which provides evidence of its suitability. Thus, this work paves the way for the monitoring and full-data retention of current and future VoIP traffic network at low investment costs.

The rest of the paper is organized as follows: Section 2 presents an overview of VoIP technology over SIP protocol and the first problems we found dealing with real traces. Section 3 introduces *VoIPCallMon* and details each of its modules. Afterwards, Section 4 shows *VoIPCallMon*'s performance evaluation and a practical case study. The related work is discussed in Section 5, and, finally, Section 6 gives the conclusions of this study.

2 Overview

VoIP is a group of technologies that provide mechanisms for voice transmission over IP networks [13]. VoIP brings together different signaling protocols that follow either proprietary or open designs such as Skinny and SIP protocols, respectively. Proprietary signaling has been extensively used in the early stages of VoIP, however, the widespread deployment of VoIP demands the interoperability of different VoIP equipment vendors in order to establish end-to-end calls. In this scenario, SIP [15] is emerging as the de facto standard for VoIP signaling. As an example, it is being used by all the main US cable operators that give VoIP services [14]. In fact, we have been provided with data from a popular VoIP operator that also uses SIP as signaling protocol. For all these reasons, we take SIP as the leading signaling-protocol example to explain our proposal, *VoIPCallMon*. However it is worth remarking that *VoIPCallMon* in addition to SIP also interprets Skinny. We note that the differences are only on the coding of the signaling interpreter, while they share the methodology, architecture, data structures, and hardware-tuning knowledge followed in *VoIPCallMon*'s implementation and explained throughout this paper.

Let us focus on those SIP protocol details required to fully understand *VoIPCallMon*'s functionalities. SIP uses plain-text messages in combination with Session Description Protocol (SDP) [16] messages to establish and negotiate the parameters of the associated RTP streams (*call-ID*, available codecs, dynamic ports that RTP stream is going to use, etc.) across different systems [17]. Such RTP streams contain encoded voice as part of its data, and potentially an associated RTCP flow. RTCP is indeed an important component in a VoIP architecture used to exchange out-of-band statistics. However, it turns out that in many cases it is not used to avoid loading both the network with additional traffic and the media gateways with the work of handling RTCP sessions. Therefore a VoIP tracking system based solely on SIP and RTP and not in RTCP is a more general approach.

In the initialization phase of a call, the caller sends an *INVITE* message that is answered by the callee with several responses, finishing with a *200 OK* response and an *ACK* that establishes the call. These messages can also be exchanged with proxies, which can act as either caller or callee. Note that from the point of view of a full-data retention and monitoring tool, it is irrelevant if the source or destination edges may forward the traffic.

[1] `http://sipp.sourceforge.net/`

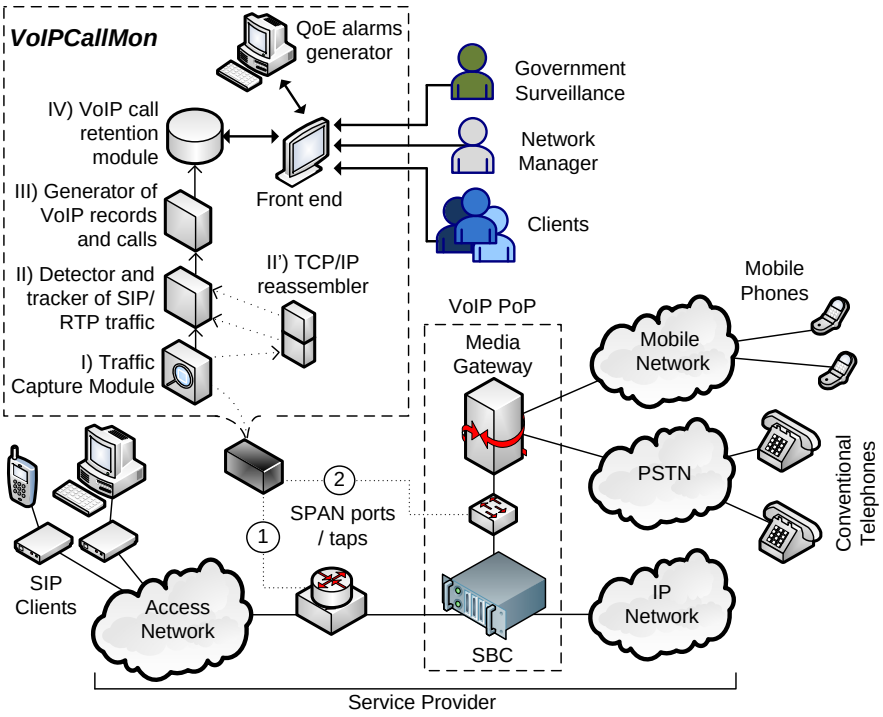

Fig. 1. SIP VoIP network (bottom) and *VoIPCallMon* architectures (top).

Once the call has been established, caller and callee use the information contained in the SDP received from the other end to send the RTP streams with the negotiated IP addresses, port numbers and codecs. It is worth noting that several *INVITE* messages can be sent during a call (the so called re-INVITEs), as SIP protocol implements expiration timeouts and they must be refreshed to keep the call alive. These messages can also be sent if the call is hold, or SDP parameters are changed. To end a call a *BYE* message is sent, being answered again with a *200 OK* response.

The basic commercial SIP VoIP network architectures are based on three elements: SIP Clients, Session Border Controllers (SBCs) and Media Gateways. SIP clients initiate calls by sending messages to SBCs. Usually, these clients are located behind a multimedia modem/router that separates VoIP traffic from normal traffic. This division is often performed by operators using different VLANs (Virtual Local Area Networks) to simplify traffic switching and routing across the Access Network. SBCs are in charge of switching the signaling and media streams present in SIP communications. Also SBCs act as SIP proxies redirecting SIP calls either to the destination client or to another SBC. However, if a SIP call must be transmitted to the PSTN network or the mobile network, signaling and media stream are redirected to the Media Gateway. Media Gateways are in charge of interconnecting VoIP networks with traditional telephone and cellular networks by making the appropriate conversions between different coding and transmission techniques. Figure 1 (bottom) shows the basic setup of a commercial SIP VoIP architecture. Usually, this architecture is replicated and spatially distributed to interconnect clients all over the coverage area.

This architecture allows the placement of VoIP monitoring probes in two different locations (labeled as 1 and 2 in the figure), either by means of a Switched Port Analyzer (SPAN port) or a tap. The first location is the router between the Access Network and SBC, this provides some advantages as system transparency but increases the capture rate needed as all the traffic is aggregated. The second possible location is the Media Gateway. The advantage of this approach is that the capture rate requirements decrease, because the Media Gateway does no receive cross traffic but end-to-end VoIP calls are not captured. It is worth remarking that as will be shown *VoIPCallMon* achieves high permanence in commodity hardware, hence tackling both scenarios.

VoIP makes use of different codecs to transmit voice samples [13]. The most popular VoIP codecs are G.711 and G.729 [1]. G.711 provides two different companding (compressing and expanding) algorithms: μ-law (PCMU) and A-law (PCMA), both of them produce data streams at 64 kb/s. G.729 produces lower rate data streams, usually 8 kb/s, but it is based on patents and requires a license to be used. In both cases, the default inter-packet gap is 20 ms, which gives packets of 200 and 60 bytes at the network layer, respectively. Note that the packet size of RTP streams is usually

constant and clearly lower than the mean packet length on the Internet, yielding a higher packet rate, which increases the difficulty of capturing this traffic. For example, a fully-saturated 10 GbE link (including Ethernet headers, CRC, Inter-Frame Gap and Preamble) with G.711 calls gives a rate of 5.3 million packets per second (Mp/s) whereas G.729 gives a rate of 12.8 Mp/s (i.e., (10^{10} / ((60 (network layer size) + 14 (Ethernet header) + 4 (CRC) + 8 (Preamble) + 12 (Inter-Frame Gap)) · 8)).

2.1 The IP fragmentation and TCP segmentation

SIP protocol can be transported on top of UDP, TCP, or SCTP involving fragmentation issues when monitoring calls. In the case of UDP, fragmentation is done at the IP layer. Moreover, when using TCP as transport layer, segmentation is additionally done by the operating system of the SIP message issuer. The TCP segmentation may cause a SIP message to be split into several segments or even parts of different messages are transmitted in a single segment (e.g., the end of a message and the beginning of the next one). Given that these messages may contain a SDP description, keeping track of the segments and obtaining each message separately are essential actions to correctly identify all the signaling conversations and their associated RTP flows. SCTP partially shares this problem with TCP. While SIP messages over SCTP will travel in different data chunks (they are not mixed), it is necessary to analyze if there are fragmented data chunks to reassemble the SIP messages before they are processed. Given this problem is equivalent to TCP segment reassembling, let us focus only on TCP.

All these cases must be considered since SIP over TCP is an extended choice in commercial networks: it provides additional signaling reliability and allows identifying the end of a connection thanks to TCP flags, which improves the QoE of the VoIP service and reduces keep-alive messages. Despite its importance, there is little work about how such fragmentation/segmentation problems impact on the development of high-speed monitoring systems [18]. On the other hand, our proposal copes with them even at multi-Gb rates.

2.2 Other internetwork problems

There are some other problems, namely, packet loss, duplicated packets and packet reordering. Regarding packet loss, if it is produced during the capture process, lost fragments will not be retransmitted and, as a consequence, the message will not be reassembled and the call will not be monitored. This highlights the high-permanence requirements of a VoIP monitoring system.

However, if that fragment is lost by a router or in the network, the sender will issue it again after a timeout implemented by SIP or TCP. This situation will be detected as out-of-order packets. Packet reordering can be resolved using a buffer to recover the monitoring process if a packet or segment is not received when expected. Duplicated packets must be detected to avoid interpreting incorrectly redundant data (e.g., a duplicated *INVITE* message as a second call). In this case, the 5-tuple {source address, source port, destination address, destination port, transport protocol} plus the TCP sequence number can solve the problem. If the SIP messages are transported over UDP, it can be necessary to use the SIP protocol fields (*call-ID* and *CSeq* SIP sequence number) to know if it is a different message or not, because the IP *identifier* field could be set to zero.

This paper explains and evaluates a methodology and data structures to circumvent all these issues at multi-Gb/s rates.

3 VoIPCallMon

In what follows, let us discuss the functionality of each of the *VoIPCallMon* modules, which are shown and labeled in top left part of Figure 1. *VoIPCallMon* software is written in C language over a Linux-based system. Particularly, the experiments were conducted over an Ubuntu Server 10.04 Long Term Support with kernel 2.6.32 running on a general purpose server.

3.1 Traffic capture module: high speed and timestamp precision

Packet capture at multi-Gb/s rates making use of commodity hardware and only-software solutions is a challenging task. Modern network interface cards (NICs), such as 82599-based Intel 10GbE, have the potential capacity of receiving traffic at such high rates thanks to its multi-queue (typically, named

as RSS, Receive-Side Scaling, queues) capabilities and the usage of multi-core processors. However, standard drivers and operating system network stacks are unable to make the most of such capacity because they are not designed for this purpose but optimized for common desktop or server tasks (e.g., Web browsing or hosting). Mainly, the limitations of the network stack and default drivers are per-packet memory operations and multiple copies through the stack. To overcome these deficiencies, novel packet I/O capture engines have been proposed [10, 19, 20]. Such capture engines have modified the standard network stack and drivers to implement three enhancements, namely: (i) memory pre-allocation, (ii) packet-level processing and (iii) direct access between the application and the driver (zero-copy). Thanks to these modifications along with a careful tuning of the software design and critical hardware parameters, existing overheads in standard operating system network stack are circumvented, sniffing packets up to 40 times faster [20] and achieving wire-speed capturing, i.e., up to 10 Gb/s and more than 14 Mp/s per interface.

VoIPCallMon requires to keep track and correlate both SIP and RTP data flows but note that these flows do not share the same 5-tuple, and that, however, such tuples are used by RSS technology to distribute packets among different receive queues at NIC and kernel level. Consequently, this prevents the use of multiple receive queues in VoIP monitoring as a signaling flow (e.g., SIP) and its corresponding payload flow (e.g., RTP) may potentially end up to different queues and, thus, cores. Moreover, the use of RSS produces the appearance of undesirable side effects such as packet reordering [21], which, again, prevents its use. Additionally, we notice that most of the previously proposed capture engines do not perform packet timestamping, even though a precise timestamp mechanism is crucial for monitoring purposes (e.g., to estimate QoS parameters such as jitter).

This encouraged us to develop a new capture engine that comprises these two challenging characteristics. That is, it must be capable of achieving 10 Gb/s line-rate packet capture using one single reception queue and it must perform timestamping. To this end, we have implemented a modified Linux driver for Intel 82599-based NICs and a user library with a C-language API (Application Programming Interface), which allows the access to data at high speed from the application level (in our case, from the detection module, labeled as II in Figure 1). The modifications can be ported to NICs based on other chipsets, but notice that such Intel chipset has become the current de facto standard for high performance computing [22], our driver's sources are available on demand at [8].

The first modifications of the driver consist in the allocation of any of the needed memory resources at load time in lieu of the allocation of memory at demand. Once this is done, the driver receives traffic from the NIC as usual by means of DMA (Direct Memory Access) transfers, that is, the NIC uses pre-allocated memory in the kernel space to copy the packets without involving the CPU. Figure 2 depicts this process, and shows the used terminology. Then, the capture module, running in a kernel-space thread, constantly polls the DMA-able memory region for new incoming packets. This is a significant difference from other approaches, and key to obtain precise timestamping at high speed. Instead of fetching packet from the DMA-able memory region when a high-level layer polls for them (i.e., an application), our modified driver asks constantly for new packets, if there are new incoming packets, it copies them to the memory region accessible for the application layer. Packets are timestamped at this moment. This is done at kernel-level by means of `getnstimeofday`[2] function, which provides sub-microsecond precision.

Note that should the driver does not poll for packets, the timestamp accuracy depends on the time the fetcher thread is scheduled by the operating system, adding imprecision to the timestamping mechanism. The driver capacity to timestamp packets is undoubtedly a need for any monitoring system, but the importance of its precision should not be ignored. The use of other approaches to fetch packets may not only add imprecisions to measurements but even generate zero-interarrival times [23]. This is because engines transfer a batch of packets per interrupt to reduce the number of interrupts from the NIC and, as a consequence, all the packets in the batch share the same timestamp. This effect applies especially when rates are low and packets from the same call likely fall in the same batch. The fact that different packets have the same timestamp dramatically affects jitter calculations, for example, which are very important for VoIP quality estimation.

The process of coping packets to make them accessible to the application layer follows the well-known mechanism of zero-copy. That is, the application, in our case the detection module, is able to map memory to positions at the kernel space (specifically, named as Kernel packet buffers in Figure 2). Accessing incoming packets from user-space in a zero-copy basis avoids extra copies as it is the case of the Standard Linux Network Stack. Such mapped memory region is a circular buffer with two

[2] `http://man.cx/getnstimeofday(9)`

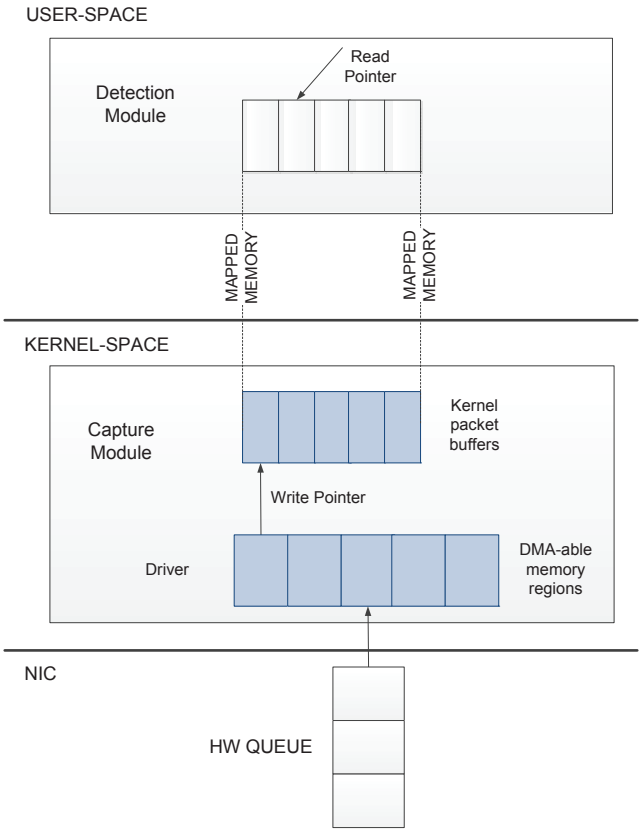

Fig. 2. Traffic capture module scheme: interactions with the NIC and detection module.

pointers, one for writing (used by the capture module from kernel-space) and another for reading (used by the detection module from user-space).

A key aspect for designing packet capture engines is the concept of affinity. On the one hand, hardware interrupts thrown by the NIC have to be handled by a specific core (interrupt affinity) whereas the capture thread must be tied to the same CPU core (thread affinity). Thus, we keep the data locality in such CPU core, which reduces cache misses, avoids context changes and, therefore, increases the performance. On the other hand, since in a NUMA (Non-Uniform Memory Access) architecture the memory is split in chunks (NUMA-nodes) and each of them is assigned to a set of cores, threads must allocate memory within their corresponding NUMA-node (memory affinity) to achieve better performance by reducing access times [19]. Thus, by the time *VoIPCallMon* is launched, or any equivalent monitoring application, both affinities must be satisfied to achieve optimal performance. From this development, we have learned that performance may drop in half if affinities are ignored. Most of the current operating systems and development libraries include functionalities to this issue. For example, in our case, Linux `taskset`[3] utility and `pthread_setaffinity_np`[4] function of the POSIX pthread library allow to fix processes and threads, and the file `/proc/irq/IRQ#/smp_affinity` interrupts to a given core, respectively. Regarding memory affinity, the `sysctl` interface provides system information about the architecture of the CPU and cores to choose its most adequate combination.

While the optimization of I/O packet capture systems has received an extensive study by the research community, the challenges that application developers face by using these new paradigms have only recently arisen expectation. There are some examples of applications such as software routers [19, 24], Network Intrusion Detection Systems [25], traffic classification [26, 27] that have exploited these systems. This paper, for the first time ever, translates the principles that those novel I/O packet capture systems apply to the development of a VoIP monitoring applications. Specifically, as shown in the following sections, we have imitated memory management mechanisms (i.e., pre-allocation and reuse), and thread and memory affinities.

[3] `linux.die.net/man/1/taskset`
[4] `linux.die.net/man/3/pthread_setaffinity_np`

3.2 Detector and tracker of SIP/RTP traffic: methodology and data structures

Once a packet is received and timestamped, it can be accessed from this second module, which is responsible for detecting and correlating the SIP and RTP traffic. This second module is executed in a different thread, running at user-space. To achieve peak performance, such thread must be tied to a different CPU core than the capture thread of the capture module but in the same NUMA-node. Thus, both threads are running in parallel increasing the power processing and, at the same time, memory affinity is kept reducing memory access latency.

SIP packets contain important information to characterize a call such as the call identifier (*call-ID*), caller and callee identifiers (*from* and *to* fields), as well as information to establish the RTP session, essentially source/destination IP addresses and port numbers (4-tuple) and codec of the RTP stream.

Typically, caller and callee exchange the RTP-stream information at the beginning of a new call by means of a couple of SIP packets that comprise in their bodies a SDP message (SIP-SDP) with this information. This step is called initialization phase as was detailed in Section 2.

However, these SIP-SDP packets can be one of various types of SIP messages (e.g., *INVITE*, *200 OK*, *100 TRYING*, *ACK*) and not necessarily the two first ones. Moreover, RTP 4-tuple can change during the course of a given call, typically named as renegotiation phase. This forces to monitor any SIP packet throughout the whole call. Essentially, such a renegotiation is initiated by a new *INVITE* message in a call already in progress. Renegotiations are quite common in SIP. Renegotiation happens periodically, as a mechanism to make sure that connections are still alive, although in such a case, the parameters typically remain the same. Other scenarios include the presence of proxies and back-to-back user agents (B2BUAs), in such case these intermediary nodes may force renegotiations because of, for example, security reasons or change in the overlay topology. Moreover, it also appears when a call is forwarded to another SIP host. This is very common in call centers and offices or due to answering machines.

This second module uses two hash-tables, as shown in Figure 3. As collisions are possible, we use node-linked lists in each entry of each hash-table to handle this issue. Such nodes and links are represented with circles and arrows in the figure. The first one, (T1), is indexed by SIP *call-ID*, which is useful when a call is updated by a SIP packet; the second one, (T2), is indexed by RTP 4-tuple, which is useful when a RTP packet is processed. Note that these two tables are necessary because RTP packets do not have *call-ID* or other SIP information, whereas SIP packets do not have the same IP addresses/ports as RTP packets. Both hash-tables access to the same data structure by means of pointers. Such structure, (S) in the figure, contains information about the call and its corresponding RTP stream (4-tuple), as well as data about the RTP traffic (that the following module needs to calculate RTP statistics), and the payload of the RTP stream (i.e., call content in raw format).

In the early stages of the system, a VoIP call was marked as expired only when a *BYE* packet is received. However, we observed that a number of calls were not properly closed (e.g., errors in the SIP negotiation phase or malicious traffic) and such calls remained in memory indefinitely. Therefore, it was necessary to implement a garbage-collector mechanism based on timeout expirations. Periodically, the calls in the tables are checked for expiration. The frequency of this expiration considers calls with silence suppression when the VoIP device does not transmit any RTP packet for a while. However, scanning the whole hash-tables requires prohibitive computational costs. Instead, we use a list of active calls, (L), which is scanned to expire inactive (or closed) calls/RTP streams. While the hash-tables are useful to insert and update a call/RTP stream, the active call list is useful to remove them. The nodes of this list have pointers to the structures they represent, and in turn, such structures also comprise pointers to the corresponding SIP and RTP entries in the hash-tables, see Figure 3. This makes it possible to remove all the references of a call in both hash-tables by means of the active call list.

Such list is sorted in decreasing order by the last packet arrival time. Thus, the garbage-collector only examines the n first active calls up to the first one that does not have to be expired given a threshold, the rest of the calls do not have to be scanned. When a call is updated by accessing to one of the hash-tables, the corresponding node in the active list is moved to the end, keeping the list sorted without any additional work.

Finally, all the memory used in the application, structures, nodes, lists and hash-tables are pre-allocated in a memory pool to reduce insertion/deletion times, avoiding dynamic allocation/release during the execution. That is, once a data structure is no longer required, the memory is returned to the pool but not de-allocated. Then, such data structure can be reused without a new allocation

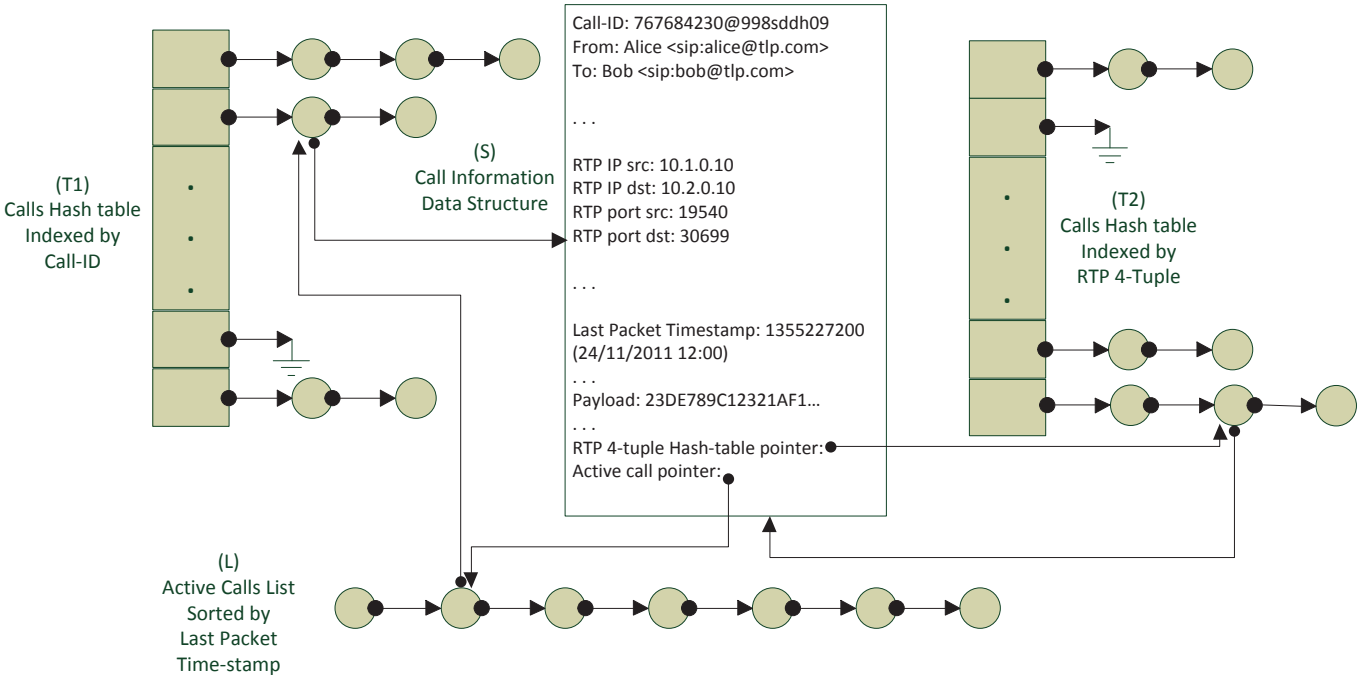

Fig. 3. Data structures to store and handle calls information in the SIP/RTP traffic detection module.

process. The use of these reviewed data structures along with the application of a pre-allocation mechanism has resulted in a major point to boost the performance of the system over multi Gb/s rates.

Bearing in mind all these implementation details, let us summarize the execution flow of this module according to the last received packet:

- SIP packets: These packets are easily identified because their source/destination port is the default port number (e.g., 5060), or other configured port (e.g., 5065, 5070). Then, to classify SIP messages among their different types, we use deep packet inspection (DPI) methodologies. The *INVITE* message is searched as an indication for a new call placement in the active calls table or a renegotiation. In the former case, a new call is inserted in the call hash-table, using the *call-ID* as key.

  Any subsequent SIP packet with the same *call-ID* is examined to complete the rest of the required fields. Essentially the IP addresses and port numbers of the corresponding RTP stream but also any additional SIP field considered useful (e.g., *to*/*from*). Once the IP addresses and port numbers fields are fulfilled, and the connection confirmed by both sides, i.e., *200 OK* and *ACK* SIP messages with correct numbers of sequence, a new entry in the RTP hash-table is created. Otherwise, the connection is ruled out and the memory released. An *INVITE* message with a *call-ID* already present in the call hash-table may indicate the first step of a renegotiation phase. Once the renegotiation is confirmed by the pertinent SIP packets, again *200 OK* and *ACK* SIP messages, the 4-tuple that characterized a RTP stream in the RTP hash-table is replaced by the new one. This allows tracking the entire call despite changes on its RTP parameters. Finally, a *BYE* message causes the call to be exported, as explained in detail in the next section.

- RTP packets: For each UDP packet arrival with even port numbers (i.e., a candidate to be a RTP packet), the module checks among the active call list entries if one call has been previously placed with the same IP addresses and port numbers. If it is the case, the payload is stored and statistics of the calls are updated. As a high-level summary, the system uses the information retrieved from SIP connections to identify the volatile RTP ports and IP addresses of calls, and then, when a RTP packet arrives at the system, its ports and IP addresses are used to deem to which call it belongs to.

- Remaining packets are discarded.

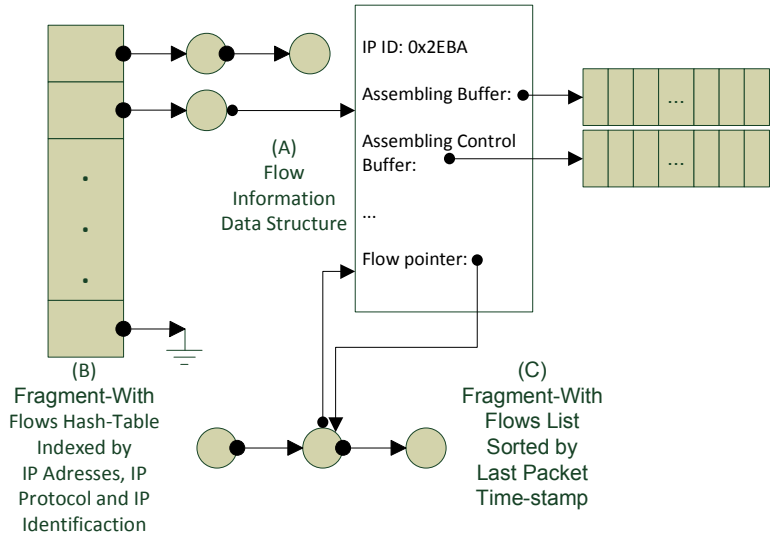

Fig. 4. IP reassembly data structures.

3.2.1 TCP/IP packet reassembler

As discussed previously, if SIP traffic is transmitted over TCP, an application layer message (in our case, a SIP packet) may be split into several segments or even several messages could be merged into one TCP segment. Thus, prior to any analysis, SIP messages must be extracted as a unique body from TCP streams and forwarded to the detection module (module II) as shown in Figure 1. We have developed an additional module, named TCP/IP packet reassembler (module II′), to deal with this potential problem. Similarly, this module also copes with the IP fragmentation. To summarize, if TCP segmentation or IP fragmentation are detected in the network, TCP/IP packet reassembler module must be activated.

The case of IP reassembly is a well-known issue that has already been solved by network stack but whose implementation is not able to deal with high rate speeds. Consequently, we have followed the development principles presented in the previous section. We have used a hash-table to track the fragments of a unique IP packet (indexed by IP addresses and identifier, and layer-4 protocol). Since packet loss may appear, we have again developed a garbage collector as a list of active connections in the same way that the ones explained in the previous section. That is, the hash-table is useful to insert new fragments and the active list is used to rule out those packets that cannot be completed. Figure 4 shows these structures. Once all fragments of a given packet have been received, it is forwarded to the following module, either the detection module or the TCP reassembly submodule if necessary.

Similarly, the TCP reassembly submodule uses a hash-table to track connections but, this time, indexes segments by 4-tuple (IP addresses and port numbers). In this case, we aggregate the TCP payloads in order according to their sequence numbers. Then, we deem that a SIP message is completely formed when we have its SIP header and the beginning of a following message is identified. In this case, we forward the formed SIP message to the detection module. Periodically, the active list is analyzed and those messages that cannot be completed are ruled out. Figure 5 depicts these structures.

3.3 Generator of VoIP records and calls: data retention and monitoring

When a call is expired, it is deleted from the detection module's active call list and hash-tables, and redirected to this third module which generates all the required information about the call. This includes the information required by the European Union directive 2006/24/EC [6]: the ID of both edges of the communication (*from* and *to* SIP fields), and the start and end times of the call. Furthermore, the module includes several flow parameters useful to assess the users' QoE for a given call such as the used codecs, count of packets and bytes, throughput, max/min/mean/standard deviation of both packet size and inter-arrival time, round-trip-time, jitter and packet loss rate. For a further thorough explanation of the calculated parameters the reader is referred to [28].

This module is also able to dump to non-volatile storage the call content in raw format. In addition to the above mentioned parameters, the call record also includes pointers to two file names comprising

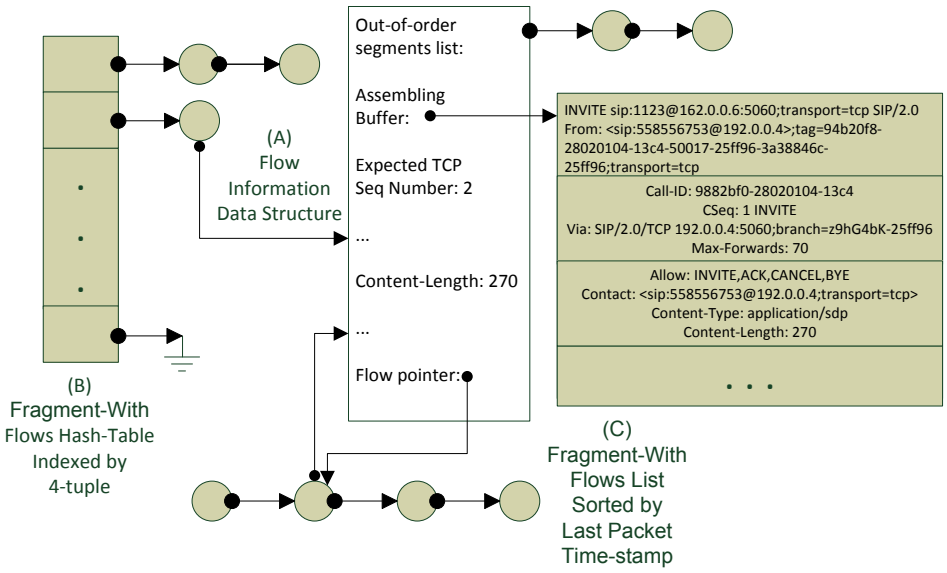

Fig. 5. TCP reassembly data structures.

the packet aggregation of the RTP communication in both directions. The total rate of the system may be limited in this step by the disk throughput [29]. In the worst-case scenario in terms of bitrate, using G.711 codec, the maximum required rate for call payload storage in a 10 Gb/s link is 6.72 Gb/s (160 RTP payload bytes / 238 total packet bytes · 10 Gb/s). Note that it is not necessary to store all the internetwork traffic as headers and cross traffic, which further reduces the required throughput of the storage media.

Several solutions of storage devices can be found in the market, namely: Serial ATA interfaces present a throughput of 1.5, 3 and 6 Gb/s in its versions I, II and III respectively; in turn, the throughput of Serial Attached SCSI is 3, 6 and, expected in a future, 12 Gb/s for its specification SAS-1, SAS-2 and SAS-3.

However, from the *VoIPCallMon* deployment, we have ascertained that such theoretical throughputs are unlikely to be achieved in real world scenarios. The performance of all these storage solutions must be improved using striping techniques, such as Redundant Array of Independent Disks (RAID) 0. Two parameters shall be configured to obtain further performance: RAID strip size and the filesystem request queue size. The former may be fixed using specific tools provided by RAID controller vendors. The latter may be tuned in the case of Linux by modifying the system configuration file */sys/block/device/nr_requests*. Specifically, the lower the value contained in this file is, the lower is the amount of petitions stored in the OS queue, thus reducing the use of OS swap memory when writing data into the RAID disk. We obtained the best results, shown in Section 4, using sizes of 1 MB for the RAID-controller blocks and by minimizing OS cache writings (by fixing *nr_requests* to the minimum value, 4, into the mentioned file).

3.4 VoIP call retention module: quality assessment and surveillance purposes

Periodically, the output of the previous module, i.e., calls records and pointers to the RTP files (if such service is required) are dumped to a MySQL database. Thus, clients, network managers or any other agent (i.e., law enforcement and intelligence agencies) may retrieve calls, measurements or alarms using a given key (e.g., the caller/callee ID, phone number/user name) via a front-end (see Figure 1).

As it turns out, the database size may be too large to achieve acceptable search times. Thus, the database can be split into several independent databases which can be accessed in parallel. We performed extensive testing of the freeware database MySQL and concluded that a database per day suffices to have a satisfactory search time (as shown in the next section), while avoiding excessive fragmentation of the data set in many databases.

If a system user is willing to listen to a given conversation, the raw file must be converted to an audible format, e.g., WAV format. To achieve this goal, a script is created which uses open source

tools such as sox[5] and Asterisk[6] to convert from the two files that comprise unidirectional raw RTP streams to a WAV file comprising both call directions. This script works together with the MySQL database to query and convert calls on-demand.

In the case of network managers, the call records and RTP aggregates are useful to both give answer to clients' complaints and to assess the offered quality. In the former case, network managers inspect calls at the request of the users, for the latter case, we have deployed an alarms generator as a new element in the deployment. This element analyzes all the traffic and marks a subset of calls as candidates to be analyzed by the network manager due to poor QoE. Such analyses can be carried out on-the-fly, after calls have finished to understand why there were problems and solve them, or even in a pro-active way to avoid them to happen.

In more detail, we have developed several ways of evaluating such QoE of the calls: the first one is a script that marks on-the-fly calls whose estimated quality is below a configurable value by means of the Mean Opinion Score (MOS) [30], thus helping network managers to focus only on those calls that deserve their attention. The estimation is based on the ITU recommendation G.107 [7] which provides a method, namely E-Model, to relate quality and transport level metrics. We have followed the further adaptation to VoIP services presented in [31] which leverages on a set of parameters related to the specific codec in use, RTT, jitter and packet loss ratios.

The second one is based on the final code received in the *SIP INVITE* transaction. By querying this value, we may count the number of successfully answered calls, as well as busy signals, other rejections and call failures. With these numbers, we may work out the Answer Seizure Ratio and Network Efficiency Ratio as defined in ITU recommendation E.411[8].

Finally, based on the stored call durations we have been able to query the average call duration, time to answer a call and number of repeated calls as others QoE metrics [32]. All of them represent useful metrics for assessing subcontracted call centers quality.

4 Performance evaluation

In this section, we assess if the proposed system, *VoIPCallMon*, is able to deal with 10 Gb/s rates.

The available testing server is based on a Sandy Bridge motherboard with two processor sockets, each with an 6-core Intel Xeon E5520 at 2.27 GHz. The system is equipped with six modules of 4 GB DDR3 RAM and an Intel X520-SR2 10 Gigabit Ethernet NIC which is based on 82599 chipset and plugged into a PCIe GEN3 x8 port. Regarding the system storage, it comprises 12 Serial ATA-III disks in a RAID-0 volume controlled by a LSI Logic MegaRAID SAS 2208 card. These disks are Seagate ES2 Constellation.

First of all, let us evaluate the performance of the traffic capture module. Note that this module captures traffic regardless whether it is VoIP traffic and its evaluation is valid for any other purpose. Therefore, we have conducted the standard experiment for comparison of capture engines' performance, i.e. stressing the engine using small packet sizes [22]. We have generated synthetic traffic composed by TCP packets with random payload and layer-2 size of 64 bytes (excluding CRC) at maximum rate for 30 minutes. This gives a rate of 14.20 Mp/s (i.e., 10^{10} / ((64 + 4 (CRC) + 8 (Preamble) + 12 (Inter-Frame Gap)) $\cdot$ 8)), about 1.6 TB and more than 25 billion packets. In this challenging scenario, no packet loss was reported.

The evaluation of the modules II, II$'$ and III requires generating VoIP and cross traffic at multi-Gb/s rates. However, the throughput of the available real VoIP traces was far from such rates, which has led us to use such real traces as a model to deploy the following controlled scenario. It comprises several computers equipped with 1 Gb/s NICs generating VoIP calls at maximum rates by means of Sipp application. Sipp allows us to generate VoIP traffic according to several traffic models and codecs. More specifically, it allows modeling the call holding time with an extensive set of distributions as well as the inter-arrivals time between two consecutive calls. We have modeled the call arrival process by a Poisson process as shown in [2], and the call hold time with an inverse Gaussian distribution as proposed at [3] with parameters $\mu = 117$ and $\lambda = 19$, both assumptions are in line with our available data. As payload, we have used real calls from our traces using codec G.711 as being the most popular in our captures. Unfortunately, Sipp was not developed to achieve a rate of 1 Gb/s and 12 machines aggregating traffic were used to generate multi-Gb/s traffic. In addition, we have used

[5] http://sox.sourceforge.net/
[6] http://www.asterisk.org/
[7] http://www.itu.int/rec/T-REC-G.107/
[8] http://www.itu.int/rec/T-REC-E.411/

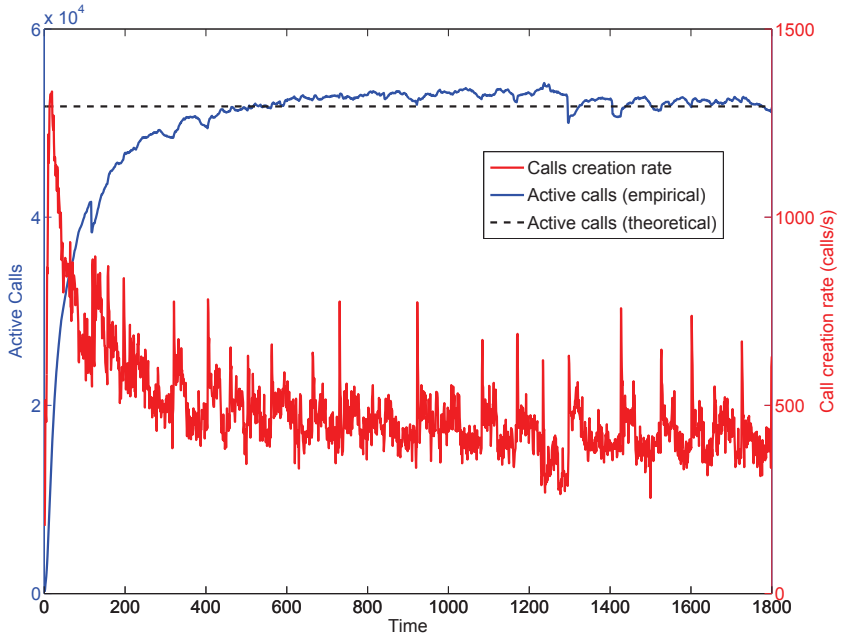

Fig. 6. Active calls and new calls managed by *VoIPCallMon* during a 30-minute experiment.

another machine equipped with a 10 Gigabit Ethernet NIC to generate cross traffic. All this traffic is aggregated by a 10 Gb/s switch, Cisco Catalyst 2960S, and then, forwarded by its SPAN port to *VoIPCallMon* system. With this experimental setup, we have generated VoIP traffic at 8.9 Gb/s, and the rest of the link capacity with cross traffic.

Figure 6 shows the number of active calls during a 30-minute experiment in this scenario as well as the call generation rate. The number of active calls gives more than 52,000 concurrent calls with a rate of 442 new calls per second in the stationary state, which yields the above-mentioned VoIP rate of 8.9 Gb/s.

Attending to storage, at least 6.72 Gb/s of disk write rate must be reached for a fully-saturated link with G.711-codec calls (RTP traffic without headers, as detailed in Section 3.3). In the above-mentioned scenario, this minimum rate should be 5.98 Gb/s. Codec G.711 represents the worst case scenario because the ratio between the RTP payload and packet size is the highest. Using our experimental setup based on a 12 disk RAID-0 the maximum achieved packet capture and write rate was 9.44 Gb/s which exceeds the required rate. This figure was obtained after configuring the RAID-controller-cache blocks to have 1 MB and disabling the kernel's cache that operating systems use when applications write in disk.

No packet was lost throughout the experiment, and only two cores were used. One core for the capture process and another for the rest of tasks of *VoIPCallMon*, which means that half of the CPU capacity is idle in our testbed. This supports the high performance capacity of the proposed system, its validity to monitor large-scale VoIP network, and even suggests that its bounds are beyond a 10 Gb/s link.

To quantify the impact of call of call insertions and subsequent retrievals in the well-known SQL database (that is, module IV), we conducted the following experiments. The first experiment measures the time devoted to insert call records when the number of records varies between 100 and 10,000,000. The second one measures the time needed to retrieve one record in a database with a number of records varying also between 100 and 10,000,000. The SQL query has been built as the worst case scenario: the statement uses a "*WHERE*" clause involving all the fields of a record. For 10,000,000 records the insert process and the search process take 900 and 6 seconds, respectively, with time decreasing linearly with the number of records with negligible variance between different trials. These numbers support the applicability of this approach.

Regarding storage, there are two different sets of data: on the one hand the call records and, on the other hand, the raw RTP streams. For the former, each call record has variable length because the length of SIP caller and callee IDs are variable fields (e.g., a phone number or an email). The call

Table 1
Empirical conversion rates from raw to WAV format for G.711 (PCMU and PCMA) and G.729 codecs. The mean call duration is 120 seconds.

Codec	Conversion rate (kB/s)	Conversion rate (calls/s)
PCMU	13,149	6.85
PCMA	19,609	10.21
G.729	1,051	4.38

records obtained from real traffic traces comprising the fields previously explained show an average call record length of 510 bytes. The average database size obtained for some 10,000,000 call records is 4.2 GB.

In the case of tracking RTP raw data, the storage becomes a significant challenge. For the case of an average call of 117 seconds using G.711, 8 kB must be stored per call and direction every second, which gives 1.87 MB of disk space per call. Assuming 10,000,000 calls per day, for full-data retention near 18 TB must be stored. This approximation shows the worst case scenario. Conversely, the most optimistic scenario gives near 2.25 TB per day using G.729.

Finally, Table 1 shows the conversion rates of a real trace provided by a VoIP operator, into WAV files. It can be observed that G.711 codec calls are converted faster than the ones from G.729 codec. In the three cases, the observed rates are not sufficient to support the throughput of the rest of the system (larger than 400 calls/s). Nevertheless, note that it is not necessary to convert all files. The conversion process can be performed on-demand when the conversation is requested.

4.1 Case study

To put these figures into perspective, we present the traffic intensity figures of a VoIP operator. This operator has fifteen VoIP points of presence (PoPs) across Spain as those shown in Figure 1. Each PoP serves between 60,000 and 100,000 calls during the busy hour whose mean call hold time is about 120 seconds, being the total number of calls per day a number ranging between 0.6 and 1 million calls per PoP. These results in a mean call arrival rate during the busy hour that ranges between 17 and 28 calls per second and between 2,040 and 3,360 simultaneous active calls. Given these specifications, in computational terms, *VoIPCallMon* is able to monitor and register calls in each PoP of the VoIP operator previously described, and, even more, it would support a growth of the number of users in, at least, one order of magnitude. Similarly, the authors in [3] explain that Fastweb VoIP network serves about 32,000 calls in a busy period of 30 minutes. This yields a rate of 17 calls per second with a call duration average of 117 seconds and about 2,000 simultaneous active calls, well below the *VoIPCallMon* capabilities.

As stated before, the deployment cost is a major issue for network operators. *VoIPCallMon* is a cost-effective solution to monitor a VoIP large-scale network. Indeed, the hardware cost of the system to run *VoIPCallMon* is estimated in $2,500 (RAID controller and server), about $1,000 (NIC) and, initially, 12 hard disks (as the ones used in the testbed) that cost, in total, lower than $2,500. Concerning storage at long term, for sure all user call records must be collected but RTP contents must be only stored when full-data retention is required. In this light, the system needs a disk capacity of 126 GB to store all call records in one month. To implement full-data retention, the first step is to assess the percentage of users whose calls must be stored. Assuming that 1% of the total calls must be recorded, 540 GB per month would be required. In the worst and unlikely scenario in which all calls must be stored, the amount of 54 TB is needed. It is worth noting that such enormous amount of storage capacity is perfectly feasible to achieve with a single 4U storage chassis. Actually, there are chassis that allow to aggregate tens of hard disks and connect to other chassis in a cascaded configuration to make up a unique solution. In terms of monetary cost, assuming a price of $100 per TB, the cost of the storage can be estimated in $50 or $5,000 per month, according to the percentage of stored calls (1% or 100%). We think that storing more than two years worth of data is not necessary, and this constitutes the limit for the storage capacity. Indeed, note that the data retention directives typically require collecting data during a period between six months and two years.

Finally, thanks to the minimal interference of *VoIPCallMon* with existing VoIP architectures, we consider the integration cost is basically reduced to the hardware cost, which supports the applicability and viability of our system.

5 Related work

We have identified two main areas that our work is related to. First, those efforts that the research community has made to detect and track VoIP traffic, and then, works describing VoIP monitoring systems once VoIP traffic is isolated.

Regarding the first issue, we remark the following works [3, 33, 34, 35]. The authors in [33] and [34] present different but similar statistical methodologies to detect Skype and Google Talk calls. Essentially, their approaches consist in characterizing some invariant patterns in the call flows of a given VoIP application, and using statistical tests or empirical thresholds to mark if a given flow is a call. Such patterns include interarrivals times, packets sizes or sizes of web responses to cite some of them. As a further step, [35] proposed to include host behavior measurements (interaction of hosts in terms of IP address or port number, for example) as discriminating parameters with notable success. So far, we note that the target of these reviewed studies are focused on detecting VoIP calls based on closed source applications while often they were precisely designed to obstruct their identification.

Therefore, the authors in [3] turned its focus from such proprietary design applications to the detection of the RTP protocol in a commercial VoIP operator. Interestingly, they proposed to snoop traffic checking if packets match with RTP headers and consecutive packets follow the typical behavior of this protocol. For example, RTP uses UDP as level-4 protocol, even port numbers and some fixed values in the headers (version field). With this information, they extracted RTP traffic from real traces, and studied useful metrics regarding calls in a commercial VoIP operator. Examples of these are call durations (model used in Section 4), RTT, jitter, among others.

All these works are useful to detect VoIP calls, however, our target is not only to detect VoIP traffic but to correlate calls with users. That is, among other services, we provide operators with mechanisms to evaluate the QoE of users, assess subcontracted call centers quality, records users' complaints and by-phone contracts in addition to meet current and future data-retention directives of governments. This is only possible by interpreting the signaling connection and relating the resulting information to the call itself. None of the previous works faced such issues, indeed the authors in [3] explain that approaches based on identifying and interpreting the signaling connections, as our work shows, are more reliable, although, probably, much more complex. This complexity is due to the requirement of carefully parsing the signaling protocol, but undoubtedly entails a higher degree of reliability because it does not depend on heuristics with different ranges of accuracy in terms of true/false positive/negative rates. Importantly, we note that these works do not study the computational burden that their approaches imply and the traffic used as a testbed is far from multi-Gb/s rates.

Once VoIP is considered as detectable, there are several works describing VoIP monitoring systems. For instance, Patent [36] describes an IP monitoring system that includes specifically a VoIP monitoring device that calculates the QoS by computing the packet loss, latency and jitter of the packets in the call. Another patent [37] envisages a system that performs VoIP traffic interception. It is based on a state machine to follow each call. Patent [38] describes a system to troubleshoot VoIP network problems, in this case, performing active tests. However, detailed technical specifications of the call monitoring engines are not provided, none is based on cost-aware hardware and their permanence is unknown.

The authors in [39] stress the importance of a monitoring system in VoIP to measure QoS, and present a system that takes the VoIP traffic directly from an IP-PBX connected to ADSL links, which are automatically adjusted according to the measured QoS. Similarly, in [40], the author shows how his nTop and nProbe applications can also monitor VoIP traffic passively tracking information from both SIP and RTP flows. However, this work states that simultaneously keeping both protocols' records in high speed links is actually a challenging task. None of these works deal with VoIP multi-Gb/s rates or show a way to intercept traffic.

It is difficult to compare our methodology, proposal and results with commercial solutions from leading vendors, as they follow closed source and proprietary designs. According to their brochures, their solutions' capacity should be similar as they announce to be able to cope with 10 Gb/s rates interfaces. Although there is neither any specific reference to full-data retention nor details of the composition of the traffic, while we have shown that the number of packets per second is even more significant than the rate itself for evaluation purposes.

Additionally, other concerns arise. First, the techniques and methodologies used, or, even the hardware principles and designs followed, are unknown to research community what hinders their analysis, development, enhancing, comparison, or even the use in other contexts of such advances by other practitioners and researchers in the area. We believe that our work may awake new ideas

and utilities for commodity hardware, and it may help to carry out other researches even outside the Internet traffic monitoring community.

Second, the cost is drastically higher.

Third, as explained before, other advantages of commodity hardware do not apply to commercial solutions as they lack of flexibility and scalability, which limits their utility at mid and long terms and increases expenditures.

6 Conclusion

This paper has described *VoIPCallMon*, a novel system to implement full-data retention of VoIP traffic on high-speed network scenarios using commodity hardware. That is, the system has been newly-designed both in cost and performance-aware fashions.

On the one hand, it uses only commodity hardware and minimizes interferences with the existing VoIP infrastructure. *VoIPCallMon* may be connected to a SPAN port of a router traversed by VoIP traffic without any other interaction or configuration, which reduces deployment costs. On the other hand, the performance evaluation has stated that *VoIPCallMon*'s limits are above 10 Gb/s, surpassing other systems presented in the literature, ergo merging low-cost and high-performance in a proposal. Such performance evaluation has combined real-world traffic traces and synthetically generated ones along with an illustrative case study.

In addition to this, our proposal offers novel services to their customers and provides network managers with automatic tools to assess the users' QoE, at the same time they fulfill current and future data retention directives.

Throughout the paper, we have explained the keys to achieve such results over commodity hardware. It has been possible thanks to modifications on the operating system's networking stack, essentially, for efficient memory management and packet processing, hardware interactions at low- level, and programming optimization. Remarkably, this spans a new network driver and novel data structures both tailored to network monitoring, especially to VoIP monitoring. Moreover, we have also highlighted any unexpected problem we have encountered by dealing with real VoIP traffic, and we have explained mechanisms to improve hardware interactions at low-level.

References

[1] Karapantazis S, Pavlidou F. VoIP: A comprehensive survey on a promising technology. *Computer Networks 2009*; **53**(12): 2050–2090.

[2] Choi Y, Silvester J, Kim H-C. Analyzing and modeling workload characteristics in a multiservice IP network. *IEEE Internet Computing* 2011; **15**(2) (2011): 35–42.

[3] Birke R, Mellia M, Petracca M, Rossi D. Experiences of VoIP traffic monitoring in a commercial ISP. *International Journal of Network Management* 2010; **20**(5): 339–359.

[4] Gibeli LH, Breda GD, Miani RS, Zarpelão BB, Mendes LS. Construction of baselines for VoIP traffic management on open MANs. *International Journal of Network Management* 2013; **23**(2): 137–153.

[5] Yu J, Zhou X. Ultra-high-capacity DWDM transmission system for 100G and beyond. *IEEE Communications Magazine* 2010; **48**(3): S56–S64.

[6] Stampfel G, Gansterer WN, Ilger M. Implications of the EU data retention directive 2006/24/EC. In *Proceedings of Sicherheit, Saarbrucken, Germany* 2008; 45–58.

[7] Bellovin SM, Blaze M, Landau S. The Real National-security Needs for VoIP. *Communications of the ACM* 2005; **48**(11): 120.

[8] High Performance Computing and Networking research group, *VoIPCallMon*. `http://www.hpcn.es` [4 March 2014].

[9] Cisco, Monitoring VoIP with Cisco network analysis module. `http://www.cisco.com/` [4 March 2014].

[10] Braun L, Didebulidze A, Kammenhuber N, Carle G. Comparing and improving current packet capturing solutions based on commodity hardware. In *Proceedings of Internet Measurement Conference, Melbourne, Australia* 2010; 206–217.

[11] Keromytis A. A comprehensive survey of voice over IP security research. *IEEE Communications Surveys & Tutorials* 2012; **14**(2): 514–537.

[12] Chen W, Hung H, Lin YB. Modeling VoIP call holding times for telecommunications. *IEEE Network* 2007; **21**(6): 22–28.

[13] Goode B. Voice over Internet protocol (VoIP). *Proceedings of the IEEE* 2002; **90**(9): 1495–1517.

[14] Zourzouvillys T, Rescorla E. An introduction to standards-based VoIP: SIP,RTP, and friends. *IEEE Internet Computing* 2010; **14**(2): 69–73.

[15] Rosenberg J, Schulzrinne H, Camarillo G, Johnston A, Peterson J, Sparks R, Handley M, Schooler E. *RFC 3261* SIP: Session initiation protocol, 2002.

[16] Handley M, Jacobson V. *RFC 2327* SDP: Session description protocol, 1998.

[17] Schulzrinne H, Casner S, Frederick R, Jacobson V. *RFC 3550* RTP: A transport protocol for real-time applications, 2003.

[18] Cascarano N, Ciminiera L, Risso F. Optimizing deep packet inspection for high-speed traffic analysis. *Journal of Network and Systems Management* 2011; **19**(1): 7–31.

[19] Han S, Jang K, Park K, Moon S. PacketShader: a GPU-accelerated software router. *ACM SIGCOMM Computer Communication Review* 2010; **40**(4): 195–206.

[20] Rizzo L. Revisiting network I/O APIs: the netmap framework. *Communications of the ACM* 2012; **55**(3): 45–51.

[21] Wu W, DeMar P, Crawford M. Why can some advanced Ethernet NICs cause packet reordering?. *IEEE Communications Letters* 2011; **15**(2): 253–255.

[22] Biersack E, Callegari C, Matijasevic M (Eds.). Data Traffic Monitoring and Analysis: from measurement, classification and anomaly detection to Quality of experience. *Lecture Notes in Computer Science series 7754* 2013; chapter 1, 3–27.

[23] Moreno V, Santiago del Río PM, Ramos J, Garnica JJ, García-Dorado JL. Batch to the Future: analyzing timestamp accuracy of high-performance packet I/O engines. *IEEE Communications Letters* 2012; **16**(11): 1784–1788.

[24] Rizzo L, Carbone M, Catalli G. Transparent acceleration of software packet forwarding using netmap. In *Proceedings of IEEE INFOCOM, Orlando, FL* 2012; 2471–2479.

[25] Vasiliadis G, Polychronakis M, Ioannidis S. MIDeA: a multi-parallel intrusion detection architecture. In *Proceedings of ACM Conference on Computer and Communications Security, Chicago, IL* 2011; 297–308.

[26] Szabó G, Gódor I, Veres A, Malomsoky S, Molnár S. Traffic classification over Gbit speed with commodity hardware. *Journal of Communications Software and Systems* 2010; **5**(3).

[27] Santiago del Río PM, Ramos J, García-Dorado JL, Aracil J, Cuadra-Sánchez A, Cutanda-Rodríguez M. On the processing time for detection of Skype traffic. In *Proceedings of Wireless Communications and Mobile Computing Conference, Istanbul, Turkey* 2011; 1784–1788.

[28] Scholz, H. *IETF Internet-Draft* RTP stream information export using IPFIX, 2012.

[29] Schneider F, Wallerich J, Feldmann A. Packet capture in 10-Gigabit Ethernet environments using contemporary commodity hardware. In *Passive and Active Measurement Conference, Louvain-la-Neuve, Belgium* 2007; 207–217.

[30] Lima AFM, Carvalho LSG, Souza JN, Mota ES. A framework for network quality monitoring in the VoIP environment. *International Journal of Network Management* 2007; **17**(4): 263–274.

[31] Cole RG, Rosenbluth JH. Voice over IP Performance Monitoring. *SIGCOMM Computer Communication Review* 2001; **31**(2): 9–24.

[32] Holub J, Beerends JG, Smid R. A dependence between average call duration and voice transmission quality: measurement and applications. In *Proceedings of Wireless Telecommunications Symposium, Pomona, CA* 2004; 75–81 .

[33] Bonfiglio D, Mellia M, Meo M, Rossi D, Tofanelli P. Revealing Skype traffic: when randomness plays with you. *ACM SIGCOMM Computer Communication Review* 2007; **37**(4): 37-48.

[34] Freire EP, Ziviani A, Salles RM. Detecting VoIP calls hidden in web traffic. *IEEE Transactions on Network and Service Management* 2008; **5**(4): 204–214 .

[35] Li B, Casner S, Ma M, Jin Z. A VoIP traffic identification scheme based on host and flow behavior analysis. *Journal of Network and Systems Management* 2011; **19**(1): 111–129.

[36] Chong R. System and method for monitoring a packet network. *US Patent 6,738,353*; 18th May 2004.

[37] Pence R, Gundabathula S. Stealth interception of calls within a VoIP network. *US Patent App. 10/378,106*; 24th February 2003.

[38] Slattery T, Pittelli F. Voice over IP analysis system and method. *US Patent 7,616,579*; 21st July 2005.

[39] Manousos M, Apostolacos S, Grammatikakis I, Mexis D, Kagklis D, Sykas E. Voice-quality monitoring and control for VoIP. *IEEE Internet Computing* 2005; **9**(4): 35–42.

[40] Deri L. Open source VoIP traffic monitoring. In *Proceedings of SANE*, Delft, The Netherlands, 2006.