Evaluation of Compression of Remote Network Monitoring Data Streams

Peter I. Politopoulos, Evangelos P. Markatos, Sotiris Ioannidis
Institute of Computer Science
Foundation for Research & Technology – Hellas
{ppolitop,markatos,sotiris}@ics.forth.gr

***Abstract*—Network monitoring and measurement is an invaluable tool for comprehending, analyzing, managing, and optimizing performance and security of networked systems. Network monitoring architectures can take the form of local or distributed deployments of sensors. Local deployments can be very precise and efficient because they benefit from fast links to the central monitoring station, but their scope can be limited to local or small-scale networks. Distributed monitoring infrastructures give a much broader view of the network state, but have the disadvantage that the amount of information they can push back to the central monitoring station is limited by the capacity of the links.**

In this paper we investigate the effects of compression on network monitoring data streams that are transmitted from distributed network sensors back to a central infrastructure. Our analysis shows that we can achieve very high compression rates, which means we remove much of the capacity overheads when transmitting sensor data back to the central monitors, while incurring only minimal delay in transmission of monitoring information. Our scheme also has the additional benefit of decreased CPU load at the monitoring sensor due to the aggregation of data which reduces the number of network messages.

I. Introduction

In our modern networks remote monitoring is of paramount importance. It can serve as a centralized security monitoring tool, as a way for network providers to monitor their service, or simply as a way to collect statistical data about network traffic. Existing monitoring infrastructures rely mostly on local network sensors but the need to track remote networks continues to grow. Where-ever there are remote sensors in place, accessing their data streams is limited by the amount of bandwidth available. That is, that in order to get an accurate picture of the remote network, the remote sensor will have to forward as much traffic as possible to our local monitoring station. This can happen either by using dedicated links for network monitoring (a solution that is not practical since it wastes expensive resources), or use the existing network infrastructure (in which case we end up using link capacity that is normally destined to be used by regular traffic).

Let us consider for example that we want to monitor traffic at a remote DNS server. To have a complete picture of the traffic our remote monitor will have to capture and propagate every packet on the link, effectively reducing the capacity by up to 50%.

A wide range of monitoring applications require packet payload inspection. Such applications include pattern matching for IDS purposes and peer-top-peer traffic classification over port 80. Our goal is to reduce the amount of bandwidth consumed by the remote network monitors, while at the same time capturing a complete picture of the network state. To accomplish this goal we used compression. The remote monitor compresses the captured traffic before pushing it to the monitoring station. The monitoring station on its end has to decompress it before using it. For our implementation we used the DiMAPI (Distributed Monitoring API) framework [1]. DiMAPI was specifically designed and built for remote packet capturing and uses the MAPI library [2].

The rest of the paper is organized as follows. In the next section we discuss related work in the areas of compression and remote network monitoring In Section III we present the design and implementation of our remote network traffic compression system. Section IV contains the evaluation of our system. Finally, in Section V we conclude and talk about future work.

II. Related Work

Most of the existing work focuses on header compression, rather than stream or entire flow compression. The Internet Engineering Task Force (IETF) workgroup for robust header compression, established numerous standards applicable to nearly every major protocol such as IP, UDP, RTP, UDP Lite and IPsec [3]. Those standards are widely used by current VoIP and other multimedia streaming applications where headers are a large part of the total data transmitted.

Header compression is also used PaMon [4], a system that has some similarities with DiMAPI. PaMon is a distributed infrastructure for passive monitoring that can be used to determine which segments of the network are the source of problems for an application data stream. This is achieved by retransmitting the entire data flow to the monitoring host, allowing inspection of the flow and eventually, identification of the problem. While PaMon uses some compression scheme, it has two important shortcomings. Firstly, it does not aggregate the captured packets so it ends up transmitting all of them. Secondly, the payload does not reduce in size, as PaMon implements only header compression. It is also worth noting that this compression scheme does not take advantage of the similarities between packets in different flows that present locality in time and space.

The IETF has proposed RFC 3173 [5] to address payload compression. This scheme is implemented and used in most SSH, VPN and similar tunneling applications [6] prior to encryption. Unfortunately, IPComp still retains the two important disadvantages mentioned before (packet aggregation and time/space localities are not taken advantage of). The approach discussed in this paper overcomes all these problems.

Trace file compression algorithms, like VPC3 [7], cannot be used in real-time monitoring applications, since they are designed to exploit similarities that reside in great distances inside trace files. However, the properties of the traces, as shown by an information theoretic approach to network trace compression [8], allow them to be greatly compressed mostly due to recurring header patterns.

Outside the area of monitoring, several systems have been proposed for end-to-end compression. Adaptive End-To-End Compression for Variable-Bandwidth Communication [9] describes such a system and so does ACE [10], in a later publication. These, admittedly complex systems, rely on information about the flows that are neither available nor useful for passive monitoring applications. Such information includes per-flow entropy and other characteristics. Packets exchanged between the monitor and the sensors typically contain many rapidly altering multiplexed flows with different attributes. Moreover, we seek to address the need for a simple, robust compressing architecture for our monitoring system, that can be applied to many similar systems, while still introducing the least possible number of modifications to existing infrastructures.

III. Architecture

DiMAPI is a framework for distributed passive network monitoring. DiMAPI builds on top of MAPI, an API for local passive monitoring applications based on `libpcap` [11]. MAPI relies on a simple yet powerful abstraction, the network flow, but in a flexible and generalized way. In MAPI, a network flow is generally defined as a sequence of packets that satisfy a given set of conditions. These conditions can be extremely broad, ranging from simple header-based filters (*e.g.* source or destination address, protocol type, port numbers *etc.*), to sophisticated protocol analysis and content inspection functions (*e.g.* deep packet inspection). For example, a very simple flow can be specified to include all packets, or all packets directed to a particular web server (by specifying port 80 and the web servers IP address). A more complex flow may be composed of all TCP packets between a pair of subnets that contain the string "`User-agent: Mozilla/5`" The architecture of DiMAPI is presented in Figure 1.

MAPI permits users to express their complex monitoring needs by allowing them to define exactly the set of the network traffic they are interested in, and subsequently only collect data that meet those criteria. Despite its flexibility MAPI has a serious shortcoming. It is designed and built to work as a local network monitor. DiMAPI addresses this shortcoming. It extends MAPI from a single, isolated monitoring tool, to a distributed network monitoring framework. DiMAPI can be used to coordinate a plethora of distributed monitoring sensors

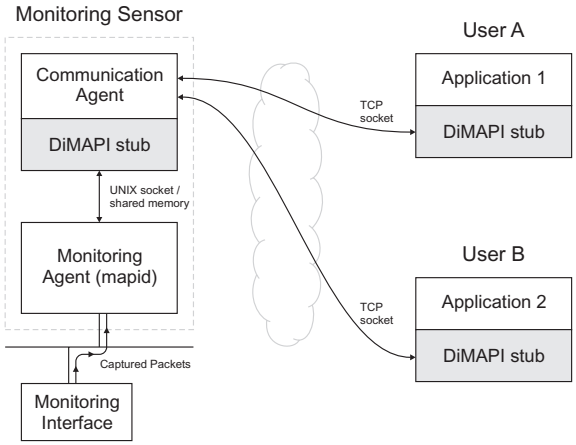

Fig. 1. DiMAPI Architecture

in a flexible and efficient way, while at the same time it maintains the specification model of MAPI.

Currently, MAPI and DiMAPI are utilised in tens of network monitors around the world as part of the LOBSTER project [12], [13].

IV. Evaluation

A. Testbed Setup

We used two hosts in our experimental testbed, each one with an Intel Core2 Duo processor running at 2.4GHz, 2GB of memory, 7200RPM SATA drives, with 1Gbps network interfaces, and running a 2.6.20 Linux kernel. The first one, called *sensor*, runs the monitoring software (MAPI) along with the remote monitoring interface (DiMAPI). On this same host, we also replay traffic on a private interface using a utility called `tcpreplay` [14]. The second one, called *client*, is located on the same subnet and switch. The client runs a loop requesting every packet seen on the private interface of the sensor. In addition to that, by utilising the MAPI library the client monitors and counts packets exchanged between itself and the sensor. The test setup is shown on Figure 2.

B. Compressing Monitored Traffic

1) Compression Algorithms Comparison: We start our evaluation by comparing a set of compression algorithms. The tests we conducted on a typical, raw network traffic trace file. The results are presented in Table I. The numbers represent an approximation of the compression ratios we can expect by applying the same algorithm on live traffic. This comparison will allow us to estimate the best choice among them, in terms of compression ratio achieved in relation to the time elapsed to compress the data. In essence which compression algorithm gives us the best *bang for our buck* with respect to compression achieved while minimizing the time taken to achieve it.

What we observe is that the best *compression ratio to time*, that is the best compression achieved per unit time, is achieved by the LZO Fast algorithm. Lempel-Ziv-Oberhumer (LZO) is a fast and efficient compression algorithm implemented by M. Oberhumer as an open-source, cross-platform library [15].

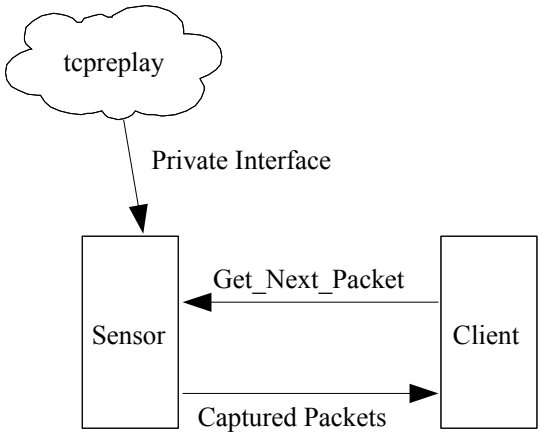

Fig. 2. Testbed setup. On the left of the schematic we have the remote sensor that monitors network traffic. On the right, client that needs to monitor this remote traffic issues commands to the sensor, which in turn returns the captured packets.

	Bytes	Buffer Size	Comp. ratio	Time
Original	1.000.001.435	N/A	0,00%	N/A
LZO (fast)	486.712.571	64KB	51,33%	13 sec
BZip 2 (fast)	597.490.154	100KB	40,25%	203 sec
Gzip (fast)	481.937.555	32KB	51,81%	32 sec
LZO (best)	479.684.340	548KB	52,03%	96 sec
Gzip (best)	479.544.036	32KB	52,05%	44 sec
BZip 2 (best)	575.745.860	900KB	42,43%	214 sec
rar	465.618.475	256KB	53,44%	137 sec
rzip	447.153.526	900MB	55,28%	146 sec

TABLE I
COMPARISON OF COMPRESSION ALGORITHMS ON THE UOC TRACE-FILE

	Bytes	Buffer Size	Comp. ratio
Original	998.239.439	N/A	0,00%
LZO (fast)	876.723.944	64KB	12,17%
BZip 2 (fast)	867.722.284	100KB	13,07%
Gzip (fast)	866.980.999	32KB	13,15%
LZO (best)	857.048.478	548KB	14,14%
Gzip (best)	856.000.024	32KB	14,25%
BZip 2 (best)	839.816.462	900KB	15,87%
rar	778.224.361	256KB	22,04%
rzip	756.044.976	900MB	24,26%

TABLE II
COMPARISON OF COMPRESSION ALGORITHMS ON THE SCHOOLMON TRACE-FILE

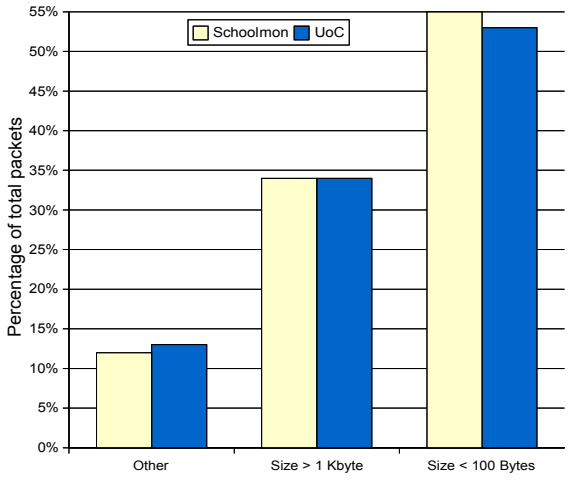

Fig. 3. Trace-file Packet Distribution

The trace-file used, which was captured from the University of Crete (UoC), was composed of normal day-to-day traffic, typical for the institution. A more extensive breakdown of the protocols used can be found in the real-time graph offered by the application monitor [16] of the LOBSTER project [17]. The second trace-file we tested was one from the Crete School Network Monitor [18]. While the original size was almost the same, the majority of the packets in this trace were parts of *already* compressed file transfers. Thus, as shown in Table II, the compression ratio observed was significantly smaller, while the times were identical within 0.01% since in both cases we compressed same sized trace-files (1GB). Both trace-files contained a large percentage of peer-to-peer traffic (>70%) and small to moderate amount of HTTP traffic (6-10%). The rest was FTP, DNS and some other protocols with very low contribution. These two traces were selected among numerous others as an average case for compression (UoC trace) and worst case (SchoolMon).

2) Per-packet Compression Measurements: The next step in our testing was to compress each packet before sending it to the client. The API remained unchanged and the benefit we observe comes from two different factors. The first factor is the per-packet compression savings, which results in actually transmitting less data. The second gain comes from reducing the number of packets transmitted from the sensor to the client. Normally, when one does remote traffic monitoring, the monitored packets are encapsulated (to include metadata, in our case the MAPI header) before transmitted back to the central monitor. If this encapsulation results in packet sizes greater than MTU this can lead to the generation of more packets. Using compression we eliminated these redundant packets.

C. Stream Compression Measurements

The results seem very promising only for specific types of traffic. For example, when monitoring an uncompressed FTP file transfer, where all packets can be significantly compressed, and all of them are exactly as big as MTU, *without* the MAPI header needed for remote monitoring, we transmitted 33% fewer packets and achieved 68% compression on the data. When this method was used on real-life traffic the results were not as good. Using `tcpreplay` with the SchoolMon trace we achieve 4% compression on the data and eliminated transmission of 2% of the packets. This was expected, as per-

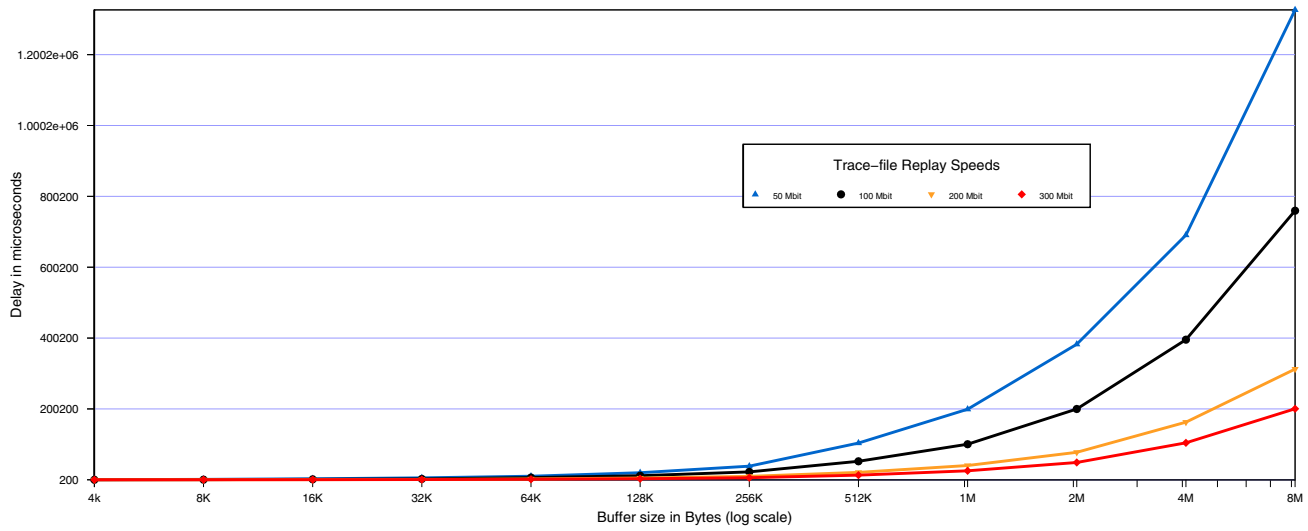

Fig. 4. Buffer delay that the client experience when requesting remote traffic. The delay is due to the time it takes to fill the remote buffer with packets, compress it, and then send it to the client. Notice that the higher the traffic rate the quicker the packet buffer is filled and therefore the delay experienced by the client is smaller.

packet compression does not take advantage of the similarity of consecutive packets in a stream, something that it was already pointed out in [19].

As already shown by the SchoolMon trace results, simply relying on data compression does not yield the optimum benefit under every possible traffic type. What we found to be common under nearly all the real-life traces we examined, is that a very high proportion of the packets inspected had a size of less than 100KB (see Figure 3). Older [20] and more recent [21] measurements show that this large number of small packets is an inherent characteristic of all internet traffic. These small packets can be control packets (SYN, ACK, RST), DNS requests, user inputs in terminal connections, HTTP GET requests, or even small UDP packets carrying voice or video fragments that need to be split up in order to maintain low delay levels. We can take advantage of this fact and transmit multiple packet data encapsulated inside a single monitoring packet. This greatly reduces the network load, while at the same time achieves better compression ratio.

1) Stream Compression: According to our measurement, a modern CPU (2.4GHz Core 2 or similar) is capable of compressing data at a rate exceeding 400MB per second. Such high rates of compression were not possible a few years ago when relevant research was conducted [22]. It is clear that there is middle ground between single packet compression and an entire traffic trace-file compression. Our approach is stream compression.

Network packets are collected in a buffer on the sensor side. When the buffer fills up, we compress it and transmit it back to the client. On the client side, the MAPI library decompresses the buffer and delivers the now uncompressed packets locally. This approach also eliminates the need of repeatedly requesting individual packets from the remote buffer. The whole process is transparent to the user application that requests them. This permits monitoring applications to be portable, since we do not modify the API.

Stream compression also exploits similarities of packets that have time locality (*e.g.* consecutive packets). Unfortunately this implementation can potentially add great packet delays in some rare cases, since we do not immediately transmit captured packets, but instead wait until the remote buffer fills up. Figure 4 presents the *buffer delay* experienced by the client, under varying buffer sizes. Measurements were carried out using various replay rates for the UoC trace-file on the sensor. It is important to note that *average packet delay* as experienced from the client does not differ significantly, if at all, from the average packet delay at the sensor whatever the buffer size may be (see Figure 5). The locally buffered packets are being pulled nearly instantly from the memory of the client, effectively negating the large buffer delays in the long run. As long as the transmission and compression process consume the monitored packets faster than they get produced, we induce no packet loss and thus the same amount of packets under the same time intervals can only give an equal packet rate.

One should note that under rare conditions the *individual packet delay* can be a lot greater. For example, if the sensor uses a filter to track only a small amount of traffic that requires, let's say, 10 seconds to fill up an 64KB buffer, the delay we will perceive for 1 in (average packet size) / (buffer size) packets will be 10 seconds + 64KB transmission time! We explain how we address this shortcoming in Section IV-C3.

Data compression is greatly increased due to the correlation of packets inside the stream. Also, the number of packets exchanged between the sensor and the client, using our compressed streaming were greatly reduced, since we can now

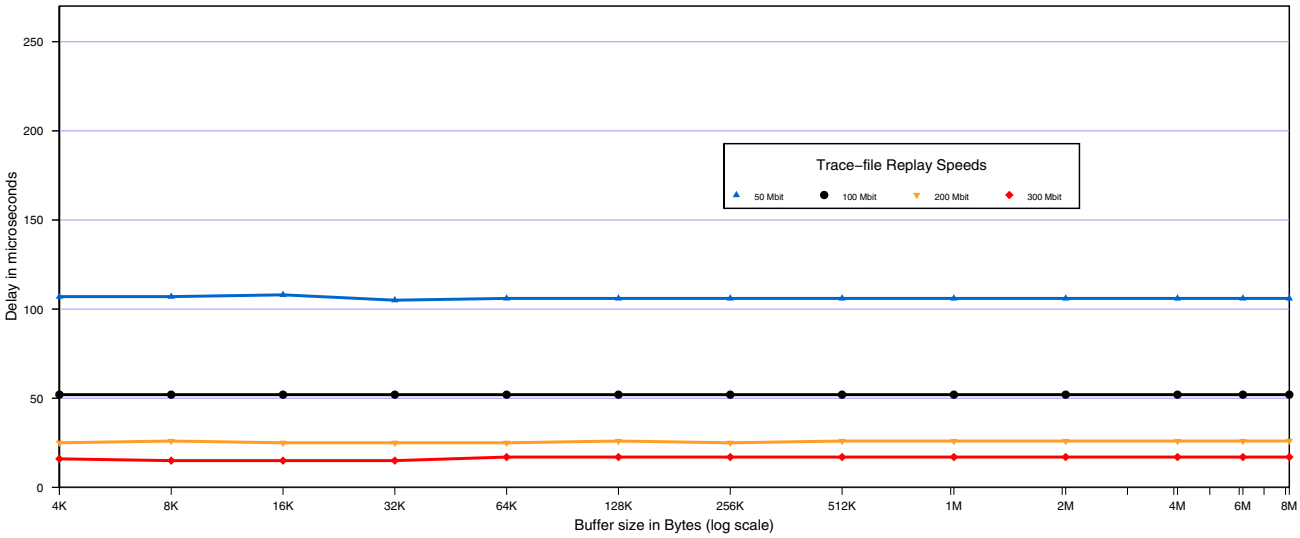

Fig. 5. Average packet delay as experienced by the client

encapsulate a significant number of individual packets inside a larger packet. Figure 6 shows the number of packets exchanged between the sensor and the client for a full monitoring session of the UoC trace-file. Without compression more than double the original 1,4 million packets are needed for monitoring. This happens because a *get-next-packet* request is sent for every packet monitored, effectively doubling the packet count. Additionally, some monitored packets that end up exceeding MTU due to the addition of the MAPI metadata, also need to be broken up into multiple packets. As the buffer sizes increases, the algorithm achieves more efficient compression and *get-next-packet* requests that need to be sent to the sensor are decreasing in number.

For buffer sizes larger than 64KB, we observed no additional benefit in either number of packets transmitted or in data compression. This limit is an artifact of the algorithm we use, that compresses data in 64KB size blocks. The optimum buffer size for our experiments was determined to be 64KB + MTU + MAPI header size, since we stop collecting packets for compression the moment we cannot fit another maximum-sized packet into our buffer. We should also not fill the buffer with more than 90% of its capacity, in case the data are incompressible. In such a case, we have the side effect that leads to small increase in data size.

As far as overall data compression goes, we achieved the theoretical maximum achievable by lzop, when tested by the command line tool that uses the LZO algorithm on the same traces and with the optimum buffer size. It is worth noting that we observed that CPU usage was lower for compression-enabled DiMAPI. This might sound counterintuitive, since one has to account for the CPU usage due to compression. The explanation is that we make up for that extra CPU time from the huge drop in packets transmitted. This leads to a reduced number of system calls which, in turn, results in lower CPU usage. We also need to stress out that the compression and decompression process consume only a minimal amount of CPU cycles by using the fast LZO algorithm.

Finally, we run an experiment where a dummy *pull* process (requesting a packets and getting a null response) was running in a tight loop. We measured the average packet delay at 98 microsecond over a 1Gb per second link. This is half the total average per-packet delay we experienced with 25Mb per second traffic. Thus, the fact that compression-enabled DiMAPI pulls packets a lot less frequently gives it a great efficiency advantage over previous systems. DiMAPI *get-next-packet* (pull) requests are sent only when the local buffer at the client side is consumed. We pushed the system and increased traffic replay rates. We observed that we were able to do remote network monitoring to rates up to 310Mb per second.

This rate represents an important improvement over the previous, non-compression-enabled implementation, that had a limit of just 27 Mbps. Any further increase of the replay rates above these limits introduces packet loss lzoat the client side.

2) Special Traffic Types: In modern monitoring applications the traffic transmitted or monitored is often of a special type (meaning either very small packets or very large packets). Most of these types of traffic are handled more efficiently by the stream compression than real-world, mixed-type, network traffic. Table III presents our experiments on special types of traffic. We present results from small, large, DNS packets, as well as packets replayed from the NLANR traces. NLANR is a trace file format developed by CAIDA for the National Laboratory for Applied Network Research (NLANR) Project [23] and contains the packet headers but not the actual payload. The compression row represents the percentage of the data transmitted from the monitor compared to the original data size, while the next row we present the same saving for

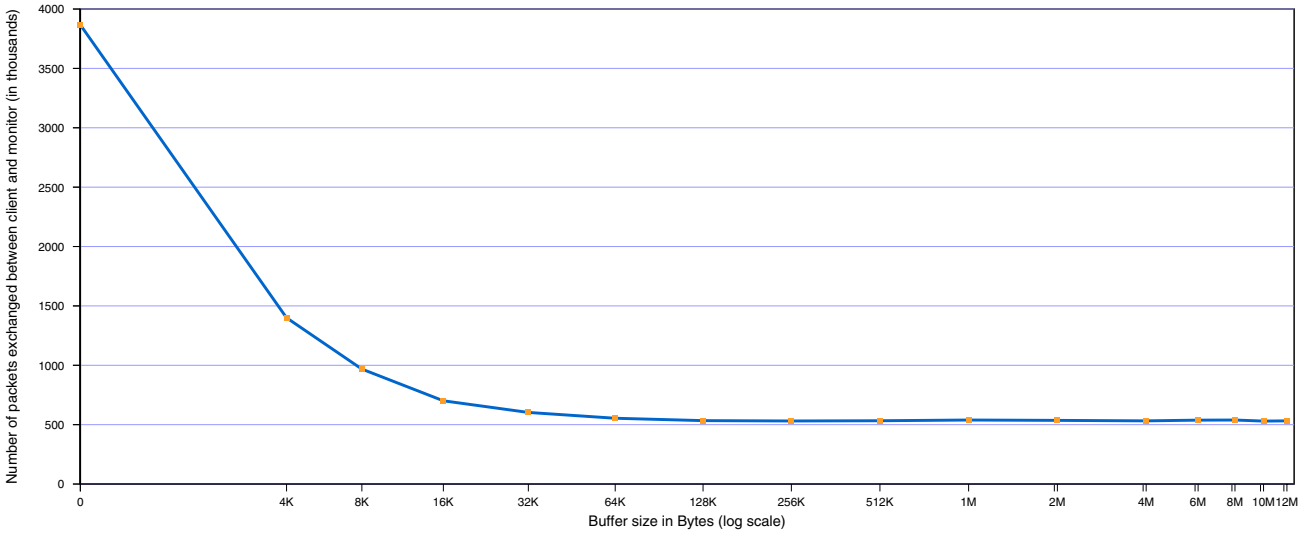

Fig. 6. Number of packets exchanged between the monitor and the client for a full trace-file monitoring session.

	Pkt_len $\leq$100	Pkt_len $\geq$1000	DNS	NLANR
Average Delay	30us	451us	37us	20us
Compression	44,12%	55,79%	31,40%	44,33%
% of original pkts transmitted	3,01%	83,02%	4,37%	3,17%

TABLE III
SPECIAL TRAFFIC TYPES AND STREAM COMPRESSION

raw packet number. In all small-packet typed traffic we save more than 95% in packet number transmissions, while the larger packets tend to be more compressible. The difference in average packet delay is expected since the experiments were conducted under a constant replay rate of 25Mb per second.

3) Adjusting the Delay: The only apparent drawback of our remote monitoring system with compression is a possible delay observed when the remote packet buffer is filled at a slow rate (due to lack of remote traffic) and a fast response is needed. For example, if we define a filter for simply monitoring a very specific size of DNS requests (that we believe resemble an attack), and such packets arrive at a rate of a few packets per minute (or even less), we are faced with the possibility of large delays in getting this information from the remote sensor. To address this shortcoming we modified our system to transmit the remote packet buffer when any of the following conditions are met:

- The packet buffer has filled.
- A certain time threshold has been exceeded from the time the client has requested the remote packet buffer and the packet buffer contains at least one packet.

The delay with this extension becomes very predictable if the packet inter-arrival rate and the average throughput is known. We can achieve any minimum delay between

$$MAPI\ request\ time + Buffer\ Transmission\ time + Packet\ Interarrival\ rate$$

and infinity, just by simply adjusting the time threshold and the buffer size. By setting the time threshold a little bit more than our average buffer delay, we can ensure a maximum limit of packet delays on low-throughput periods, without reducing the compression achieved while on normal traffic. For example, the value of 3 millisecond for the timeout did not alter the compression achieved for the sustained 25Mb per second experiments we conducted and presented in the previous sections.

V. CONCLUSIONS

Compressing the data stream from the sensor to the monitoring client reduces both the number of packets the sensor has to transmit back to the monitoring client and the size of the overall data stream transmitted. Fewer packets as well as less data, result in reduced network load, which makes it easier to use standard network links to perform remote network monitoring, instead of opting for expensive, dedicated links. Our study shows that is possible to achieve those goals using off-the-shelf, commodity hardware, with minimal overheads in CPU and memory use.

More specifically, our tests show that we are able to perform remote network monitoring at speeds us to 310Mb per second without losing a single packet from the monitored network. Additionally, our system adds only minimal delay, as we can achieve such rates with just adding a couple of hundred microseconds of latency. We believe that using compression in remote network monitoring is a practical, viable solution.

ACKNOWLEDGMENTS

This work was supported in part by the IST project LOBSTER funded by the European Union under contract 004336 and by the Marie Curie Actions - Reintegration Grants project PASS. We would also like to thank the anonymous reviewers for their comments which helped improve the quality of this paper.

REFERENCES

[1] P. Trimintzios, M. P. A. Papadogiannakis, M. Foukarakis, E. P. Markatos, and A. Øslebø, "DiMAPI: An Application Programming Interface for Distributed Network Monitoring," in *Proceedings of the 10th IFIP/IEEE Network Operations and Management Symposium (NOMS06)*, April 2006.

[2] M. Polychronakis, K. G. Anagnostakis, E. P. Markatos, and A. Øslebø, "Design of an Application Programming Interface for IP Network Monitoring," in *Proceedings of the 9th IFIP/IEEE Network Operations and Management Symposium (NOMS04)*, April 2004, pp. 483–496.

[3] "IETF Robust Header Compression (rohc) workgroup charter and list of RFC." [Online]. Available: http://www.ietf.org/html.charters/rohc-charter.html

[4] D. Agarwal, J. M. Gonzalez, G. Jin, and B. Tierney, "An infrastructure for passive network monitoring of application data streams," in *Proceedings of the 2003 Passive and Active Measurement Workshop, San Diego, CA (US) (PAM 2003)*, March 2003.

[5] A. Shacham and B. Monsour and R. Pereira and M. Thomas, "IP Payload Compression Protocol (IPComp)," Internet Engineering Task Force, RFC 3173, September 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3173.txt

[6] "IETF IP Payload compression (IPComp) implementation report." [Online]. Available: http://www.ietf.org/IESG/Implementations/IPCOMP-Implementation.txt

[7] M. Burtscher, "Vpc3: a fast and effective trace-compression algorithm," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 167–176, 2004.

[8] Y. Liu, D. Towsley, J. Weng, and D. Goeckel, "An Information Theoretic Approach to Network Trace Compression," University of Massachusetts, Tech. Rep., November 2004

.

[9] B. Knutsson and M. Björkman, "Adaptive end-to-end compression for variable-bandwidth communication," *Comput. Networks*, vol. 31, no. 7, pp. 767–779, 1999.

[10] C. Krintz and S. Sucu, "Adaptive on-the-fly compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 1, pp. 15–24, 2006.

[11] S. McCanne, C. Leres, and V. Jacobson, "libpcap," lawrence Berkeley Laboratory, Berkeley, CA. (software available from http://www.tcpdump.org/).

[12] "Lobster Project Homepage." [Online]. Available: http://www.ist-lobster.org/

[13] "Appmon sensors around the globe." [Online]. Available: http://www.ist-lobster.org/ appmon

[14] "Tcpreplay." [Online]. Available: http://tcpreplay.synfin.net/trac/

[15] "LZO Real-Time Data Compression Library," oberhumer.com GmbH, Technical Report, Oct 2005. [Online]. Available: http://www.oberhumer.com/opensource/lzo/

[16] D. Antoniades, M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ubik, and A. Øslebø, "Appmon: An Application for Accurate per Application Traffic Characterization," in *Proceedings of IST Broadband Europe 2006 Conference*, December 2006.

[17] "Appmon - University of Crete, anonymized traffic monitor and breakdown." [Online]. Available: http://lobster.ics.forth.gr/~appmon/public_sensors.html

[18] "Appmon - Crete School Network Monitor, anonymized traffic monitor and breakdown." [Online]. Available: http://lobster.ics.forth.gr/~appmon/public_sensors.html

[19] S. Dorward and S. Quinlan, "Robust Data Compression of Network Packets (Draft)," 2000, submitted to Data Compression Conference.

[20] C. C. Greg, "Recent traffic measurements."

[21] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the sprint ip backbone," 2003.

[22] B. Knutsson and M. Bjorkman, "Adaptive End-To-End Compression for Variable-Bandwidth Communication," vol. 31, no. 7, pp. 767–779, Apr 1999.

[23] "Nlanr project." [Online]. Available: http://pma.nlanr.net