Efficient Lossless Compression of CAN Traffic Logs

András Gazdag[1], Levente Buttyán[1], and Zsolt Szalay[2]

[1] Laboratory of Cryptography and System Security
Department of Networked Systems and Services
Budapest University of Technology and Economics
Email: {agazdag, buttyan}@crysys.hu

[2] Department of Automotive Technologies
Faculty of Transportation Engineering and Vehicle Engineering
Budapest University of Technology and Economics
Email: zsolt.szalay@gjt.bme.hu

***Abstract*—In this paper, we propose a compression method that allows for the efficient storage of large amounts of CAN traffic data, which is needed for the forensic investigations of accidents caused by cyber attacks on vehicles. Compression of recorded CAN traffic also reduces the time (or bandwidth) needed to off-load that data from the vehicle. In addition, our compression method allows analysts to perform log analysis on the compressed data, therefore, it contributes to reduced analysis time and effort. We achieve this by performing semantic compression on the CAN traffic logs, rather than simple syntactic compression. Our compression method is lossless, thus preserving all information for later analysis. Besides all the above advantages, the compression ratio that we achieve is better than the compression ratio of state-of-the-art syntactic compression methods, such as gzip.**

I. Introduction

Modern vehicles have multiple embedded controllers, called ECUs (Electronic Control Units), connected together by internal communication networks such as the CAN bus (Control Area Network). ECUs are programmable devices, and many of the vehicles' functions now rely on software running on them, as well as on protocols for exchanging information between ECUs via CAN buses. Often, vehicles also have interfaces, such as the OBD (On-board Diagnostic) port and various wireless interfaces that make it possible to access certain parts of the vehicle's internal network from outside. Such access may be needed for diagnostic and maintenance purposes, for connecting mobile consumer devices to the entertainment unit of the vehicle, or allowing for the transfer of various sensor data in and out of the vehicle. The concept of connected cars goes even further by introducing short range wireless connections between vehicles and to the Internet infrastructure, enabling new types of safety and infotainment applications.

All this development means that modern vehicles should be considered as cyber-physical systems, in which special purpose computers control physical processes, and those computers are no longer isolated from the cyber space out there. This introduces an entirely new domain of problems for vehicles, and road safety in general: the domain of cyber security. Indeed, ECUs in vehicles can be compromised in similar ways as computers are compromised on the Internet (e.g., exploiting a buffer overflow vulnerability in their software), and due to the increasing level of connectedness, such attacks are now possible to be carried out remotely (i.e., without requiring physical access to the vehicle). The feasibility of such remote attacks has been demonstrated by various research groups recently, showing also their potentially catastrophic consequences [1] [2] [3].

The danger of remote cyber attacks on vehicles generated a lot of interest in developing protection measures that either prevent or detect such attacks. One of the proposed approaches is to perform log analysis on recorded CAN traffic traces and identify intrusions either in real-time or in an off-line manner. While real-time intrusion detection seems to be the ultimate goal, it is not at all obvious if that would be feasible, and it is not so clear either what should be the real-time response to a detected attack. Even if these problems were solved, the off-line analysis of the logged CAN traffic would still be required for better understanding of how the attack worked and for forensic purposes in case the attack caused some physical damage.

Being able to detect and analyze cyber attacks on vehicles requires continuous collection and recording of the CAN traffic. This can potentially lead to a large amount of data that need to be stored in the vehicle. In this paper, we propose a compression method that allows for the efficient storage of that large amount of data. Compression of CAN traffic logs have other notable advantages: The logs may occasionally be off-loaded from the vehicle, and compression helps to shorten the time and bandwidth required for the off-load operation. Moreover, the large amount of data is not only a problem for storage and communication: it also makes

forensic analysis hard, resource intensive, and time consuming. In effect, this may be the most problematic issue with large amounts of log data that cannot easily be solved by advances in storage and communication technologies and by the ever decreasing cost of those. Our compression method allows analysts to perform log analysis on the compressed data, therefore, it contributes to reduced analysis time and effort. We achieve this by performing semantic compression on the CAN traffic traces, rather than simple syntactic compression. Syntactic compression methods operate on the low level byte stream representation of the data. In contrast to this, semantic compression methods interpret the data being compressed and take advantage of its semantic understanding. Just like syntactic compression methods, semantic compression can be lossless or lossy; in this paper, we propose a lossless compression method, thus preserving all information for the analysis. In addition, the compression ratio that we achieve is better than the compression ratio of state-of-the-art syntactic compression methods, such as gzip.

The rest of the paper is organized as follows: In Section II, we give an overview of existing data recording solutions in road vehicles and prior work on semantic compression. In Section III, we provide background information on the CAN technology. We describe our new semantic compression algorithm in Section IV, and we evaluate its performance in Section V where we also describe our CAN traffic collection campaign. Finally, in Section VI, we conclude the paper and sketch some possible future work.

II. Related work

A. Data recording in road vehicles

Data recording devices that can capture information continuously or triggered by an event have existed in the transportation industry for decades. The best known such devices are probably the "black boxes" used in aviation to record data that can be used by investigators to reconstruct some of the circumstances of an airplane crash. Such recording devices now also exist in road vehicles: since September 2014, a so called Event Data Recorder (EDR) is mandatory for every new passenger car and new light commercial vehicle (LCV) in the US.

The purpose of EDR devices is to collect data about the vehicle dynamics and the vehicle status that enable better accident reconstruction. It helps in validating insurance claims, encourages safer driving behavior, and extends the scientific knowledge about real accidents. The importance of an EDR-like "black box" increases with the deployment of highly automated functions in road vehicles, as there must be some objective evidence proving who was in charge of control in the vehicle in a critical situation. It is, however, not clear what would be the minimum set of data that needs to be collected in case of automated or highly automated vehicles; accident researchers and automated vehicle experts are currently working together on new regulations in this field.

While EDR devices collect data from the CAN bus, the recording of that data is not continuous in time, but triggered only by certain events that may indicate a forthcoming accident (e.g., events that trigger the airbag). In addition, the data recorded by EDR devices is limited to a short interval in time (typically a few seconds) surrounding the point in time of the accident. Unfortunately, a cyber attack that ultimately leads to an accident may happen long before the accident itself (at least, well beyond a few seconds interval around the time of the accident), and therefore, the data recorded by an EDR device will likely contain no useful information about the cyber attack causing the accident. For detecting cyber attacks and for being able to analyze after an incident how the attack was executed, one needs to collect and record a continuous flow of CAN traffic for an extended period of time.

There exist data recording devices, such as tachographs, that perform continuous data collection in vehicles. Tachographs are mainly used on heavy trucks, buses, and emergency vehicles to continuously record certain parameters of the vehicle such as its speed, its engine RPM, and odometer values. Yet, the main purpose of tachographs is to monitor the duty status of the drivers of commercial vehicles, and they are not designed to record raw CAN traffic. They usually record only a few vehicle parameters with a certain recording frequency, and they are not available on all kinds of road vehicles. Hence, similar to EDRs, tachographs in their current form cannot really be used in investigations of cyber incidents affecting vehicles.

Hence, we can conclude that, although they have seemingly similar goals, existing data recording devices in road vehicles actually address a problem different from the one that we address in this paper, and they are not appropriate for cyber incident investigations.

B. Compression

Semantic compression has generated considerable interest in the recent years. It has been successfully applied in different fields such as general database compression [4] [5] [6], video compression [7], virtual machine memory compression [8], and network traffic compression [9]. To the best of our knowledge, it has not been applied yet for forensic evidence handling in the automotive domain. Hence, what we propose in this paper is a new application area for it.

Many of the previously proposed semantic compression algorithms perform lossy compression. However, lossy compression is not appropriate for forensic purposes, as for forensic investigation, one needs to collect and retain accurate information that can potentially be used as evidence in front of court. Hence, in the sequel, we focus on semantic compression algorithms that provide a lossless service. In particular, we compare our work to [9], which was proposed to compress IP traffic captures, and [6], which is a recent semantic compression algorithm for large data tables that outperforms some earlier proposals, such as [4] and [5].

IPzip [9] is an algorithm for compressing IP network packet headers and payloads. One of the motivations for developing IPzip was to support forensic investigations and to help ISPs to comply with data retention laws. Hence, IPzip

performs lossless compression on full network traffic captures. In addition, IPzip is a semantic compression algorithm that exploits the correlations exhibited by packets that belong to the same upper layer protocol session or have the same destination port (inter-packet correlation) and correlations of header fields within individual packets (intra-packet correlation). The basic idea of IPzip is to reorder the packets in the network log such that related packets are grouped together, and to separate the structured header part of packets from their unstructured payload. In this respect, our method is similar, as we also rearrange CAN packets based on their CAN IDs and separate the unstructured CAN payload from the CAN header and other meta-information, such as timestamps. We cannot exploit however correlations of header fields, because the CAN header contains a single ID field, and we cannot either exploit redundancy in upper layer flows, because we cannot interpret the proprietary, manufacturer specific upper layer protocols. Yet, we achieve a better compression ratio than IPzip: in [9], the authors report that IPzip achieves a compression ratio between 30% and 40%, while our compression ratio is around 10-12% (in our binary format). The difference may stem from the highly periodic nature of CAN traffic, which is not a characteristic feature in IP traffic.

Squish [6] is a semantic compression algorithm that leverages the relational structure of large data tables in databases. It uses a combination of Bayesian Networks and Arithmetic Coding to capture multiple kinds of dependencies among attributes and to achieve near-entropy compression rate. More specifically, it learns a Bayesian network structure from the dataset, which captures the dependencies between attributes in the structure graph, and models the conditional probability distribution of each attribute conditioned on all the parent attributes. Then, it applies arithmetic coding to compress the dataset using the Bayesian network as the probabilistic model. Finally, it concatenates the model description file (describing the Bayesian network model) and compressed dataset file. Squish achieves a reduction in storage on real datasets of over 50% compared to its nearest competitors, including ItCompress [5] and SPARTAN [4]. It is difficult to compare it to our algorithm, because it was not used to compress network traffic logs. On tables containing discrete numbers, which are similar to a series of timestamps in our setting, it achives a compression ratio of around 32%. Our 10-12% compression ratio is better, because we can exploit the periodic nature of the timestamps.

Finally, we must mention the compression algorithms BFC and SRA proposed in [10], which were specifically developed for lossless compression of CAN packets. However, their motivation is different: they address the problem of overload on the CAN bus. BFC and SRA can compress CAN packets in real-time, which results in reduction of the bus load. They achieve a compression ratio of around 80%, and they require special hardware or software in ECUs to perform compression and de-compression. Our algorithm is not intended to be used in real-time, it does not require any modification to existing ECUs in the vehicle, and it achieves a much better compression ratio of around 10-12%. However, this comparison is not entirely fair due to the different goals of BFC/SRA and our algorithm.

III. Background - the CAN protocol

It is the responsibility of ECUs to measure the surrounding environment with various sensors and perform physical operations based on this information. The ECUs communicate with one another by sending and receiving CAN packets. The latest vehicles usually have multiple CAN buses with different speeds.

The CAN protocol was designed to be simple, causing only a small overhead in the communication. None of the messages contain any authentication information. This nature of the protocol makes it very easy to spoof CAN messages, which, in turn, can lead to numerous other attacks based on the injection of arbitrary messages into the CAN traffic. Understanding CAN messages, however, is a more complex problem, because it requires a priory knowledge of the network and the communicating parties.

The CAN protocol describes the message format. Each message has a ID field that can be either 11 or 29 bits long. After the identifier a message contains the length information followed by the message data varying from 0 to 8 bytes. The message data may differ among car manufacturers. Additional knowledge is required for its analysis. The ID is also considered as a priority field: the lower the value the higher the priority. All messages are sent as broadcast therefore the IDs are also used by the ECUs to determine whether a message is relevant for them or not.

In vehicles, the ECUs communicate with each other mostly periodically. While the communication is event based, regular repetition times enable a quite accurate prediction of the upcoming pattern of messages. Another main property of the traffic is that the content of messages are often repeating. Numerous instances of the exact same message or message type can be observed throughout traffic captures.

We confirmed these properties via capturing traffic in real vehicles. Our test vehicles, however, were not premium category vehicles. In premium category vehicles, there are more ECUs, which results in a higher variety of the CAN message types and message contents. Also, the same high variety is expected in autonomous vehicles. Yet, we believe that even in those cases, CAN traffic is rather regular and exhibits features that can be exploited for its efficient compression.

IV. Traffic log compression algorithm

A. Compression strategy

The usage of semantic compression and syntactic compression helps to achieve different goals. A clever combination of the two approaches could benefit from the advantages of both: semantic compression reduces the file size while maintaining accessibility to the data. whereas syntactic compression achieves the smallest possible file size.

To exploit the benefit of both approaches we propose to apply both methods at different operational phases. During data collection an on-the-fly semantic compression could reduce file sizes while keeping data available for immediate processing. The compressed files could be an input for IDS or other anomaly detection appliance. The compressed data format allows a fast analysis of data flows because they are stored in blocks after one another whereas investigation of causality relations is more computing-intensive.

An optional long term storage or cloud transfer of network logs requires a smallest minimum file sizes while the importance of immediate data accessibility is reduced. This implies the use of syntactic compression at this phase. It has been proven before [11] that performing semantic compression before syntactic compression still pays off.

B. Semantic compression

We propose a compression algorithm that takes advantage of the largely periodic nature of the CAN traffic. The high level approach of our algorithm is to separate the traffic into message flows, containing only messages that have the same ID, and then, compressing each message flow separately leveraging the previously identified properties. Algorithm 1. shows the pseudo code of the compression.

Algorithm 1 Semantic compression

```
messages ← read CAN traffic log
flows ← separate Messages into message groups
for all flow in flows do
    calculate_average_inter_arrival_time(flow)
    group_messages_with_identical_data(flow)
    for all message in flow.messages do
    compress_timestamp(message)
    end for
end for
for all flow in flows do
    write_compressed_flow_to_output(flow)
end for
```

After reading the log file, the first step is to filter the messages based on the ID field of the protocol. This step generates separate lists of messages (a flow) where the only remaining information for each message to be stored is its timestamp in the log and the data content of the message. In many cases, the content shows only very low variations, allowing the compression to be even more efficient by grouping together identical messages.

Storing a complete and separate timestamp for each message in a flow would be a waste of storage. Our more efficient approach takes advantage of the periodicity of messages. Theoretically, a new message with the same ID should come at an exactly predictable time point based on the inter-arrival time of this message type. However, this behavior can be changed by a higher priority message on the CAN bus. If two messages are sent at the same time, then only the one with the higher priority will be sent, shifting the inter-arrival time of the messages with the lower priority. From this point on, the arrival times of this complete message flow will be shifted.

An efficient way to store the timestamp of a message is to store the number of periods (specific for that flow) passed since the last message of the same type and an additional offset value that is induced by either priority causes or measurement distortions. This approach also allows for an efficient description of message flows, where the same message data appears repeatedly from time to time.

For each message flow, there are some additional metadata to be stored: the message ID, the first appearance of the flow in the log and the characteristic period length of the flow. These flow specific metadata should be followed by the message data and then the compressed timestamp for each message. An example of this compressed format can be seen in Example 2., where the # sign separates the period number and the offset value in each compressed timestamp.

```
1481492674.734327  0x260  8  000000000000006a
1481492674.736055  0x2c4  8  05c8000f0000923c
1481492674.738092  0x2c1  8  080335016ad9004f
1481492674.754306  0x260  8  000000000000006a
1481492674.759605  0x2c4  8  05c8000f0000923c
1481492674.769823  0x2c1  8  0803390170d90059
1481492674.774302  0x260  8  000000000000006a
1481492674.783129  0x2c4  8  05c2000f00009236
1481492674.794246  0x260  8  000000000000006a
1481492674.801541  0x2c1  8  08033b0174d9005f
1481492674.806689  0x2c4  8  05c2000f00009236
1481492674.814227  0x260  8  000000000000006a
1481492674.83034   0x2c4  8  05c5000f00009239
1481492674.833283  0x2c1  8  08033b0174d9005f
1481492674.834316  0x260  8  000000000000006a
1481492674.853767  0x2c4  8  05c8000f0000923c
1481492674.854213  0x260  8  000000000000006a
1481492674.865006  0x2c1  8  0803380172d9005a
1481492674.874181  0x260  8  000000000000006a
1481492674.877285  0x2c4  8  05c8000f0000923c
```

Example 1: Simplified CAN traffic log

1) A compression example: The operation of our semantic compression can be effectively demonstrated on Example 1. that shows a simplified CAN traffic log. It has been truncated and reduced to only contain messages from three different ID types. Other than that it is a real life traffic log.

In the first step the algorithm reads the messages separating them into groups with the same ID. In this case it would result in 3 groups: 0x260, 0x2c4 and 0x2c1. The following step is the same for each group, that is to compressing messages inside a group.

An efficient way to find repeated messages is to build a hash map of the messages using the message data as a key. At this level, the only remaining information to be stored is the arrival time of the message.

For a more efficient compression the timestamps are stored in a coded way taking advantage of the CAN traffic properties. The inter-arrival times of the messages can be calculated, based on the stamps, requiring only to store the small

difference between the predicted and the actual arrival times.

It is possible, that a message data appears in the traffic from time to time. This also has an impact on the compression, i.e. we need to store the elapsed number of periods in each and every case too. This number usually has the value of 1 but for a recurring data this may vary.

The final result of the compression of this log can be seen in Example 2. For storing the compressed timestamp the number of cycles and the arrival shifts are separated with a # sign.

C. Output formats

We defined two output formats for our algorithm. One is a text base (ASCII) representation of the traffic log, while the other is a binary format. Both formats contain the same lossless information.

It is worth having both of this options because various further usage may prefer one over the other. The binary format stores the compressed data in a more efficient way that can be seen in Section V in Table I.

```
0x260
start_time:1481492674.734327
period:19984
00 00 00 00 00 00 00 6a:  0#0,1#-5,1#12,1#-40,
                          1#-3,1#105,1#-87,
                          1#-16

0x2c4
start_time:1481492674736055
period:23540
05 c8 00 0f 00 00 92 3c:   0#0,1#10
05 c2 00 0f 00 00 92 36:   1#-16, 1#20
05 c5 00 0f 00 00 92 39:   1#111
05 c8 00 0f 00 00 92 3c:   1#-113, 1#-22

0x2c1
start_time:1481492674738092
period:31728
08 03 35 01 6a d9 00 4f:  0#0
08 03 39 01 70 d9 00 59:  1#3
08 03 3b 01 74 d9 00 5f:  1#440, 1#14
08 03 38 01 72 d9 00 5a:  1#-5
```

Example 2: Compressed CAN traffic log

V. Evaluation

We evaluated our algorithm in terms of run-time performance and efficiency. As the most important performance metric, we calculated the compression ratio and as for efficiency, we also measured the speed of our implementation. We performed our measurements multiple times with different datasets originating from different vehicles. We used vehicles of three different brands all belonging to the low mid-level category built between 2005 and 2010.

We captured traffic with a Raspberry Pi based CAN interpreter[1]. It allowed us to access the raw information on the CAN bus and we saved every CAN message with a timestamp. We performed traffic captures through the OBD interface where the design of the vehicle allowed for an uninterrupted access to the powertrain CAN bus traffic through this connection. We were able to gather traffic logs of multiple hours in all three types of vehicles that we used in the evaluation.

[1]http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2DSB.pdf

A. Run-time complexity

The time complexity of our semantic compression algorithm is $O(n)$ where n is the number of messages in the traffic log. To compress a complete traffic log, the algorithm iterates over the messages 6 times. Every iteration is linear therefore its complexity is $O(n)$. For the python implementation of the compression we also only used data structures with either $O(n)$ or $O(1)$ speeds.

The algorithm was capable of efficiently compressing data gathered during the test scenarios in every case at least a magnitude faster that the incoming speed as shown in Table I. This speed overall makes our algorithm a good candidate for on-board data compression for local usage of the information or as a preparation for a remote transmission.

B. Compression ratio

The measured compression ratios show significant progress in the data sizes (Table I. and Table II.). We were able to achieve compression ratios of less than 20% using an ASCII representation of the output of our algorithm. The binary representation shows and even more efficient compression with the results being around 10% of the original file size.

If we applied the additional syntactic compression to our semantic compression it resulted in the smallest file sizes we were able to achieve. In the ASCII representation scenario the combined result shows an approximate 6% compression ratio while the binary case show an approximate 5% compression ratio.

This result can be considered as another proof that it is worth applying semantic compression before syntactic compression because with this combination additional efficiency can be gained.

C. Correctness

We also implemented a de-compression algorithm. It allowed us to restore the data in an identical form then the original files before the compression. We performed a bit-by-bit and a SHA-256 based comparison of the original and the de-compressed files to check the correctness of our algorithm. In every case we were able to restore the original data without any loss.

VI. Conclusion

In this paper, we presented an efficient way to perform lossless compression of CAN traffic logs. Based on our observations of the periodic properties of CAN traffic, we designed a semantic compression algorithm for CAN traffic. With the use of our algorithm, storage efficiency and communication costs can be significantly improved, while

TABLE I: Semantic compression ratio comparisons

Test case	Original trace file size (bytes)	Test duration	Compression duration		Text format		Binary format	
			Text	Binary	file size (bytes)	compression ratio	file size (bytes)	compression ratio
1	10 095 971	5 min 52 sec	1.319 sec	1.428 sec	1 710 920	16,94%	1 090 757	10,80%
2	7 040 165	4 min 5 sec	0.965 sec	1.021 sec	1 334 902	18,96%	835 539	11,86%
3	19 143 383	11 min 6 sec	2.217 sec	2.478 sec	3 747 229	19,57%	2 307 146	12,05%
4	21 936 245	12 min 45 sec	2.554 sec	2.712 sec	4 233 994	19,30%	2 601 354	11,85%

TABLE II: Semantic and Syntactic compression ratio comparisons

Test case	Original trace		Semantic and Syntactic compression combined			
	file size (bytes)	zip compressed file size (bytes)	Text format		Binary format	
			file size (bytes)	compression ratio	file size (bytes)	compression ratio
1	10 095 971	1 291 315	546 725	5,41%	499 998	4,95%
2	7 040 165	937 319	429 234	6,09%	390 467	5,54%
3	19 143 383	2 569 118	1 194 758	6,24%	1 092 183	5,70%
4	21 936 245	2 895 039	1 332 585	6,07%	1 223 677	5,57%

keeping the possibility to perform analysis on the compressed data.

As part of our future work, we plan to further optimize the file formats. Our current byte level approach could still be refined if we could represent the number of cycles and offset information in a more condensed way.

Another possible follow up is to design a lossy compression algorithm for CAN traffic logs. While lossy compression would not allow for using the compressed log as evidence in a forensic analysis, it could still be used in incident investigations if the compressed file preserved all information related to attacks and discarded only information irrelevant from a security point of view. This approach could result in a dramatic improvement in compression ratio.

REFERENCES

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Snachm, and S. Savage, *Experimental security analysis of a modern automobile*, 2010, pp. 447–462.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028073

[3] A. Greenberg. (2015) Hackers remotely kill a jeep on the highway - with me in it, wired magazin. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

[4] S. Babu, M. Garofalakis, and R. Rastogi, "SPARTAN: a model-based semantic compression system for massive data tables," in *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '01. New York, NY, USA: ACM, 2001, pp. 283–294. [Online]. Available: http://doi.acm.org/10.1145/375663.375693

[5] H. V. Jagadish, R. T. Ng, B. C. Ooi, and A. K. H. Tung, "ItCompress: an iterative semantic compression algorithm," in *Proceedings. 20th International Conference on Data Engineering*, March 2004, pp. 646–657.

[6] Y. Gao and A. Parameswaran, "Squish: near-optimal compression for archival of relational datasets," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2016, pp. 1575–1584.

[7] T. Mei, L.-X. Tang, J. Tang, and X.-S. Hua, "Near-lossless semantic video summarization and its applications to video analysis," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 3, pp. 16:1–16:23, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2487268.2487269

[8] A. Rai, R. Ramjee, A. Anand, V. N. Padmanabhan, and G. Varghese, "Mig: Efficient migration of desktop vms using semantic compression," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 25–36. [Online]. Available: https://www.usenix.org/conference/atc13/technical-sessions/presentation/rai

[9] S. Chen, S. Ranjan, and A. Nucci, "IPzip: a stream-aware ip compression algorithm," in *Proceedings of the 2008 Data Compression Conference*, March 2008, pp. 182–191.

[10] Y.-j. Wu and J.-G. Chung, "Efficient controller area network data compression for automobile applications," *Frontiers of Information Technology & Electronic Engineering*, vol. 16, no. 1, pp. 70–78, Jan 2015.

[11] H. V. Jagadish, J. Madar, and R. T. Ng, "Semantic compression and pattern extraction with fascicles," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 186–198. [Online]. Available: http://dl.acm.org/citation.cfm?id=645925.671667