EventFlow: Network Flow Aggregation Based on User Actions

Petr Velan
CESNET z. s. p. o.
Prague, Czech Republic
petr.velan@cesnet.cz

***Abstract*—Network flow monitoring is being supplemented with an application flow visibility to provide more detailed information about network traffic. However, the current concept of flows does not provide a mechanism to keep track of semantic relations between individual flows that are created as a part of a single user action. We propose an extension to the flow measurement, called EventFlow, which allows to preserve relations between HTTP and DNS application flows that are a part of single user action, most typically browsing a web page. We describe an architecture of the EventFlow extension and its limitations. A prototype implementation of the EventFlow is introduced and evaluated on a packet trace from an ISP network. We show that a significant number of flow records can be recognised as a part of a single user action.**

I. INTRODUCTION

The growth of cloud-based services increases the importance of network monitoring. Information about network traffic behaviour can not only provide valuable information for performance optimisation of applications and infrastructure but also help to detect and mitigate attacks on applications and their users. To better facilitate these demands, network flow monitoring [1] solutions are starting to provide application visibility [2], [3].

Application flow monitoring parses data from application headers and adds application specific elements to flow records. This way the information from application level can be easily transferred to flow collectors, stored, and utilised together with the information about the network communication. Current approach is to treat separate application protocols individually, e.g., develop an application processing module for each monitored protocol [4]. However, connections between different protocols are lost in this scenario. For example, when a user wants to access a web page, several different flows records are created. The DNS server must be contacted to resolve the hostname of the web page to an IP address. After the basic document is loaded, the user's browser automatically loads linked content, such as images, cascading style sheets, and java script libraries. The generated requests are recorded as flows, however, little relation between the flows is preserved.

Information about relations between individual flows can be useful in several scenarios. First, when an advertisement on a web page contains malware, the page can be traced using the relation and notified of the malicious content. Second, aggregates of the related flows can be created to simplify behavioural analysis of network traffic. Moreover, the analysis can use the additional information to improve its accuracy. Last, traffic classification engines can also benefit from having access to information about flow relations [5].

978-1-5090-0223-8/16/$31.00 © 2016 IEEE

In this paper we present a flow monitoring extension, called EventFlow, which allows to keep track of relations between HTTP and DNS application flows. Information about flow relation is inserted to flow records to keep track of individual user actions, i.e., events. We develop a prototype of the EventFlow extension and evaluate its properties on network traffic trace from an ISP network. Results show that at least 10 % of HTTP and DNS flow records form more complex events. We believe, that this is only a lower bound and that further improvements can be made to relate even more flows into events.

The rest of the paper is structured as follows. Related work is surveyed in Section II. We propose the architecture of EventFlow measurement in Section III. Section IV describes the implementation of the EventFlow prototype. Experimental evaluation of the EventFlow prototype is performed in Section V. The paper is concluded in Section VI.

II. RELATED WORK

Madhyastha and Krishnamurthy [6] propose a generic language for application-specific flow sampling. Their language allows applications to select flows with special properties so that the negative impact of sampling on these applications is minimised. This can be useful for intrusion detection systems or traffic classification applications. Although the goal of this work is different from ours, it also aims to improve the collected data, so that traffic analysis applications can achieve higher accuracy.

The authors of [7] also focus on improving quality of sampled flow data. They show that the traffic classification accuracy can be increased using related sampling, which assigns higher probability to connections that are part of the same application. The authors propose to use a source IP address as a measure of relation between connection sessions.

Hu et al. [8] propose an entropy based aggregation system to mitigate an impact of DoS attacks and worm spreads on a network monitoring system. The main contribution of their approach is a flow key attribute selection algorithm that chooses key attributes by which the flows are aggregated. Two dimensional hash table is used to implement their approach.

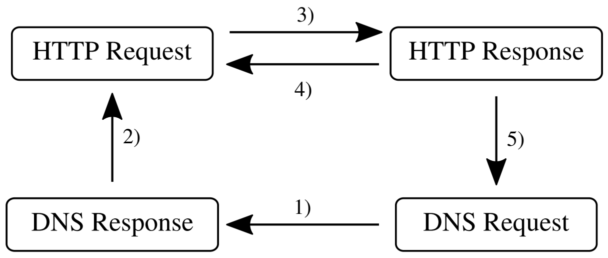

Fig. 1. Relations between HTTP and DNS requests and responses.

The aggregated flows are called metaflows. The main difference from EventFlow is that we label existing flows belonging to same user action, while the metaflow is a substitute flow for many flows created during a malicious network activity.

Dolberg et al. [9] introduce a multidimensional flow aggregation aimed to reduce the volume of collected data. The authors use tree structures for storing the data by chosen dimension such as IP addresses or ports. EventFlow proposed in our work might be used in this scenario to aggregate flows by the same events.

III. EventFlow Architecture

This section describes the architecture of the EventFlow monitoring. The goal is to label all flows that are the results of a single user action with the same event identifier (EID). For example, accessing http://www.w3.org/ creates 1 DNS request, 37 HTTP requests, and 8 HTTPS requests. We aim to assign a single unique EID to the flows generated for all these DNS and HTTP requests.

Four basic types of flows are recognised by the EventFlow: HTTP requests, HTTP responses, DNS requests and DNS responses. There are relations between these types of flows in network traffic, as shown in Figure 1. When HTTP request to a new site is performed, the IP address of the site must be resolved first. Therefore, a DNS request is created. After the request is observed, a reply usually follows, which results in the relation 1). After the DNS reply arrives, the client knows the IP address of the server and makes the HTTP request, which creates the relation 2). An HTTP response follows the request, as indicated by the relation 3). The HTTP response can contain an HTML page which links to several additional resources such as external style sheets or images. The loading of these resources triggers more HTTP requests, resulting in relation 4). When these requests point to previously unresolved domains, new DNS requests are created, which introduces the relation 5).

We base the EventFlow architecture on the relations between the requests and responses. When an HTTP or a DNS flow is encountered, we must make sure that it is assigned the same EID as the related flows. Therefore, we create four sets of records: expected HTTP requests, expected HTTP responses, expected DNS requests and expected DNS responses. When processing an HTTP or a DNS flow, we add new record to the set or sets it relates to. Then, when a next flow is processed, it is matched against appropriate expected set to see whether

TABLE I
Matched flow properties.

	Source IP	**Destination IP**	**Destination Port**	**URL**	**Domain**	**DNS Transaction ID**
Expected HTTP Request	✓			✓	✓	
Expected HTTP Reply	✓	✓	✓			
Expected DNS Request	✓				✓	
Expected DNS Reply	✓	✓	✓			✓

it is a part of existing event. If it is, an EID of the event is assigned to the flow record. For example, when a DNS response is encountered, a new record is put into the expected HTTP requests set (because of relation 2), see Figure 1). Then, when an HTTP request is processed, we check the expected HTTP requests set to see whether we are expecting this request based on a previous DNS response. If the request is matched, it is assigned the same EID as the DNS response.

Each of the sets of expected records uses different flow properties to match a flow record. When matching an HTTP request flow against the expected HTTP requests set, the source IP address of the flow must match as well as the requested domain or the URL, if available. Checking the source IP address ensures that flows from different hosts are not combined into a single event. A domain name is checked for the records that were inserted in the set when DNS reply was encountered. In case an HTTP response caused the record to be inserted, the full URL is available, not only the domain name. Replies are checked based on IP addresses and destination port. Source port is not checked since the services are expected to run on standard, well-known ports. The DNS reply is also checked for transaction ID, which is a unique identifier tying the request and response. However, the DNSSEC extension is ignored and does not affect the EventFlow. Therefore, any malicious responses would still be part of an event. List of the used properties is provided in Table I.

An expiration of the records from the expected sets must be ensured. When a record from any of the expected sets is matched, it is removed. However, many inserted records will never be matched. For example, when a DNS request is made to accommodate a different service than HTTP, the expected HTTP request might never appear. We need to free such records from the sets eventually. A timeout is used to keep the expected sets from being congested by redundant records. A timestamp is assigned to each record upon insertion to a set. Then, each time the set is searched, records older than the timeout are removed. The timeout should be as short as possible to avoid blending of several events. However, it should be at least as long as it takes to process the longest user action, which might be up to a couple of seconds in case

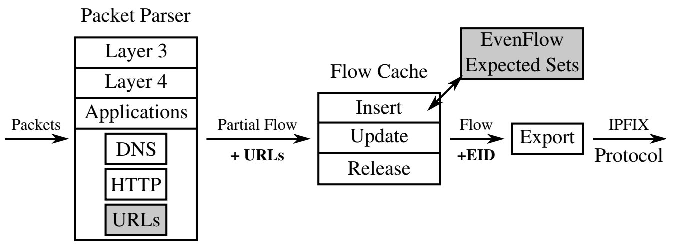

Fig. 2. EventFlow prototype schema.

of complicated queries to slow sites.

There are several caveats to our approach and some limitations of the architecture that should be addressed in the future. Our approach does not handle HTTP redirection codes, therefore the first request and the HTTP 3xx redirection response are assigned different EID than the subsequent request to the resource. This problem can be rectified simply by adding a handler for the HTTP 3xx redirection responses that will put a new record with the redirect URL to the expected HTTP requests set.

Another limitation of our approach is that the URLs are only extracted from HTML documents. However, modern web sites often use a JavaScript code to request additional resources through the Ajax technique. Such requests cannot be easily matched to an event, since it would require to reconstruct the complete web page and process the included JavaScript code, which is infeasible for the flow monitoring system.

There are also several caveats that cannot be avoided. Some of the requested documents might be cached by clients which would cause EventFlow to lose track of related URLs. However, cached DNS queries are of no consequence to the EventFlow since no traffic is generated for them and no information about a flow relation is lost. Actions of different users can be mingled when Network Address Translation is used. And finally, the growing deployment of HTTPS reduces the usefulness of the EventFlow for the HTTP protocol. Nevertheless, it can always be used in environments utilising an HTTPS proxy such as data centres or enterprise networks.

IV. EventFlow Prototype

We build the EventFlow prototype as a plugin for the FlowMon [10] flow monitoring software. The FlowMon exporter is a flexible flow exporter that provides support for various extensions. These extensions are used to implement a support for additional packet inputs, application protocol processing and different export protocols. We utilise the capabilities of the exporter to create an EventFlow extension plugin. The prototype can either be deployed to process data on live network or to analyse captured samples.

The FlowMon exporter consists of three main components as shown in Figure 2. The first component is a packet parser. It receives packets from the network and extracts information from network to application layer of each packet. The extracted information is used to create a partial flow record, which contains all necessary information about the parsed packet such as IP addresses, ports, timestamps, byte counter, etc. It also contains application layer information when an application parsing is performed. The partial flow record is passed to the second component of the exporter which is a flow cache. The partial record is either inserted as a new flow record or it is used to update an existing flow record, which is an aggregation of previous partial flow records. When a flow record expires, it is released from the flow cache to an export component. The purpose of the export component is to convert raw flow records to a flow export protocol format such as NetFlow [11] or IPFIX [12] and pass the flow records over the network for further processing.

Plugins, that extend the FlowMon exporter to process specific application layer protocols, such as DNS or HTTP, have access to several parts of the flow creation and export process. Each plugin can request to see the raw packet payload, process it, and add its own information to flow records, such as HTTP Host, Content-Type, or Response Code. Furthermore, the plugins are allowed to provide their own functionality for insert, update, and release methods of the flow cache. And last, each plugin defines how the information inserted into flow records is processed by the export component.

The EventFlow prototype is implemented as an application protocol extension. However, it also utilises the data provided by other application plugins. The EventFlow combines information from the DNS and HTTP protocols to detect relations between flows, therefore it requires the DNS and HTTP application plugins to be deployed as well. The prototype extends the packet parser to extract URLs from HTML pages. These URLs are sent together with partial flow record to the flow cache, however, they are used only internally and they are newer exported in the flow records. When a new flow record is created in the flow cache, the expected sets (see Section III) are searched for a match to the new record using DNS, HTTP, and URL information provided in the partial record. If a match is found, the new flow records is assigned an Event ID (8 byte unsigned integer) of the matched record from the expected sets. Otherwise, if no match is found, a new EID is incrementally assigned to the new flow record. After the flow is expired from the cache, the EID is a part of the record and is sent by the export component along with the rest of the flow record.

Using an 8 byte integer for EID and assigning it incrementally to individual events ensures that there are no collisions due to EID overflow in practice. However, an assignment that is individual for each flow probe and persistent over the reboots of the system would be required for a real-world deployment. The EID is assigned only to flows of the HTTP and DNS traffic, since it would provide no benefit to other flows as event relation tracking is not implemented for other protocols yet. Moreover, the size of the flows grows only by 8 bytes at maximum, which has negligible impact on the flow collector disk space requirements.

V. Experimental Evaluation

We evaluate the prototype in two scenarios. First, we assess the functionality of the prototype on a simple example web site. Once we have verified the functionality, we run EventFlow on a packet trace from live network to determine how many flows can be joined to events in real traffic. The IPFIXcol [13] flow collector is used to collect and process the generated flows. The main advantage of the collector is that it can be easily configured to work with the Event ID element.

We do not evaluate the performance of the prototype in this phase. We are aware of several performance inefficiencies that need to be solved before any valuable results can be measured. For example, one of the most expensive parts of the prototype is the management of sets of expected records. We expect that changing the underlying data structures will significantly improve the performance.

A. Functional Evaluation

For the first scenario, we create a simple website with two pages, each linking the other page, displaying an image, and referencing a different JavaScript library. The evaluation proceeds as follows. We request the first page in a browser and few seconds after it loads we follow the link to the other page. The packet trace of these actions is recorded and processed by the EventFlow prototype, and the resulting flows are collected by the IPFIXcol.

We expect to see a flow record for each of the requests and responses. However, due to the HTTP pipelining the whole communication with the web server hosting the test pages is done using a single connection. Therefore, there is a pair of flows for the accessing the two web pages with the linked images (which were on the same server), two pairs of flows for each off-site JavaScript library, and three pairs of DNS flows for IP address resolution. There are 12 flows created in this test scenario in total. The 12 flows are divided in two events by the EventFlow prototype. The first event contains flows for the two DNS requests, HTTP communication with the web server and download of the first JavaScript library. The second event does not contain an HTTP flow due to the HTTP pipelining but contains the DNS request and the subsequent download of the second JavaScript library.

The functional evaluation shows that the prototype correctly recognises related flows and labels them as a part of the same event. The flow exporter can be extended to handle HTTP pipelining by creating new flow record for each pipelined request. Such extension would make the measurement more accurate and we plan to deploy it in the future.

B. Real Traffic Evaluation

The purpose of the real traffic test is to determine how many flows can be joined into events. We collect a short (approximately one minute) trace of 10 million packets from an ISP network on ports 53 and 80 which are likely to be DNS and HTTP packets. Table II shows statistics that describe the packet trace as well as the results of the evaluation. We can see that from the total number of more than 600 thousand flows more than 400 thousand are part of events. Furthermore, over 55 thousand flows are part of events which contain more than one flow. Therefore, we can conclude that more than 10 % of observed HTTP and DNS flows are recognised as a part of more complex events by the EventFlow prototype.

TABLE II
Real traffic evaluation statistics.

Total Flows	613953
HTTP Requests	33294
HTTP Responses	49753
DNS Requests	197926
DNS Responses	224588
Events with $>$ 1 Flow	28064
Flows in Events with $>$ 1 Flow	55881
All Events	388749
Flows in All Events	418671

The number of flows in complex events is not as high as might be expected given the large number of HTTP and DNS requests and responses. We believe that this is caused by a quite short time window of our trace, which is likely to have captured large number of separate responses and requests. Moreover, we believe that better results can be achieved by fine-tuning the timeout of the records in the expected sets of the EventFlow prototype.

VI. Conclusions

We have presented an EventFlow monitoring architecture that allows to keep track of relations between HTTP and DNS application flows, which can be used to simplify behavioural analysis of network traffic, improve network threat detection and network traffic classification. The changes to existing flow monitoring architecture are negligible, which facilitates wide deployment. The proposed architecture can be further extended to handle more complex HTTP communication, such as redirection return codes.

A prototype of EventFlow plugin for the FlowMon flow exporter has been evaluated on a trace of 10 million packets. We showed that more than 10 % of observed HTTP and DNS flows are recognised as a part of more complex events by our prototype. We believe that this result will improve on longer packet trace as well as with more accurate settings of the prototype. Prospective improvements to the prototype as well as its more detailed evaluation, including a performance evaluation, are left for a future work.

We believe that the network analysis will benefit from the supplemental information about flow relations. Our work has shown that it is possible to acquire such information without a significant impact on an existing monitoring architecture and that it is possible extend the flow monitoring to trace relations of other application protocols.

Acknowledgements

This material is based upon work supported by the "CESNET E-infrastructure" project LM2015042 funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *Communications Surveys Tutorials, IEEE*, 2014.

[2] L. Kekely, J. Kucera, V. Pus, J. Korenek, and A. Vasilakos, "Software Defined Monitoring of Application Protocols," *Computers, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[3] P. Velan, T. Jirsík, and P. Čeleda, "Design and Evaluation of HTTP Protocol Parsers for IPFIX Measurement," in *Advances in Communication Networking*, T. Bauschert, Ed., vol. 8115. Heidelberg: Springer Berlin Heidelberg, 2013, pp. 136–147.

[4] ntop, "nProbe," online, 2015. [Online]. Available: http://www.ntop.org/products/netflow/nprobe/

[5] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, and B. Xie, "Internet traffic clustering with side information," *Journal of Computer and System Sciences*, vol. 80, no. 5, pp. 1021 – 1036, 2014, special Issue on Dependable and Secure Computing The 9th {IEEE} International Conference on Dependable, Autonomic and Secure Computing.

[6] H. V. Madhyastha and B. Krishnamurthy, "A Generic Language for Application-specific Flow Sampling," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 5–16, Mar. 2008.

[7] M. Lee, M. Hajjat, R. R. Kompella, and S. G. Rao, "A Flow Measurement Architecture to Preserve Application Structure," *Comput. Netw.*, vol. 77, no. C, pp. 181–195, Feb. 2015.

[8] Y. Hu, D.-M. Chiu, and J. C. S. Lui, "Entropy Based Adaptive Flow Aggregation," *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 698–711, Jun. 2009.

[9] L. Dolberg, J. François, and T. Engel, "Efficient Multidimensional Aggregation for Large Scale Monitoring," in *Proceedings of the 26th International Conference on Large Installation System Administration: Strategies, Tools, and Techniques*, ser. lisa'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 163–180.

[10] INVEA-TECH a.s., "FlowMon Probe," online, 2015. [Online]. Available: https://www.invea.com/en/products-and-services/flowmon/flowmon-probes

[11] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954, Internet Engineering Task Force, October 2004.

[12] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011, Internet Engineering Task Force, Sep. 2013.

[13] P. Velan and R. Krejčí, "Flow Information Storage Assessment Using IPFIXcol," in *Dependable Networks and Services*, ser. Lecture Notes in Computer Science, R. Sadre, J. Novotný, P. Čeleda, M. Waldburger, and B. Stiller, Eds., vol. 7279. Heidelberg: Springer Berlin Heidelberg, 2012, pp. 155–158.