Volunteer-Based System for classification of traffic in computer networks

Tomasz Bujlow, Kartheepan Balachandran, Tahir Riaz, Jens Myrup Pedersen, *Aalborg University*

***Abstract* — To overcome the drawbacks of existing methods for traffic classification (by ports, Deep Packet Inspection, statistical classification), a new system was developed, in which the data are collected from client machines. This paper presents design of the system, implementation, initial runs, and obtained results. Furthermore, it proves that the system is feasible in terms of uptime and resource usage, assesses its performance, and proposes future enhancements.**

***Keywords* — computer networks, data collecting, performance monitoring, volunteer-based system.**

I. Introduction

Monitoring of the data flowing in the inter-network is usually done to investigate the usage of network resources, and to comply with the law, as in many countries the Internet Service Providers (ISPs) are obligated to register users' activity. Monitoring can be also made for scientific purposes, like creating realistic models of traffic and applications for simulations, and to obtain accurate training data for statistical traffic classifiers.

This paper focuses on the last approach. There are many existing methods to assign the packets in the network to a particular application, but none of them were capable of providing high-quality per-application statistics when working in high-speed networks. Classification by ports or Deep Packet Inspection (DPI) can provide sufficient results only for a limited number of applications, which use fixed port numbers or contain characteristic patterns in the payload. Therefore, we designed, built, and tested a system, which collects the data directly from the machines belonging to volunteers who contribute with the traffic data. For the particular parts of the system, we described the available and chosen solutions. Our objective was to show that the system is feasible in terms of resource usage, uptime, and providing valid results. The remainder of this paper describes the previous work related to this research and then focuses on the design and the new implementation of the volunteer-based system. Finally, it shows the results of 3-months system tests and proposes further enhancements.

II. Related work

Most methods for traffic classification use the concept of a flow defined as a group of packets, which have the same end IP addresses, ports, and use the same transport layer protocol. Flows are bidirectional, so packets going from the local machine to the remote server and from the remote server to the local machine belong to the same flow. In [1], the authors proposed to collect the data by Wireshark while running one application per host at a time so that all the captured packets will correspond to that application. But this method requires each application, whose traffic characteristics have to be captured, to be installed on the host, which is run once for each application. This solution is slow and not scalable. Secondly, all operating systems usually have background processes such as DNS requests and responses, system or program upgrades. They can damage statistics of the application traffic.

A DPI solution using *L7-filter* and a statistical classification solution are proposed in [2]. Using DPI is much more convenient than the previous method, as it can examine the data in any point in the network. Unfortunately, existing DPI tools are not able to accurately classify traffic belonging to some applications like Skype (in this case *L7-filter* relies on statistical information instead of the real traffic patterns, giving some false positives and false negatives [3]). Obtaining the training data for statistical classification based on statistical classifiers will not give us high accuracy of the new classifier. The idea of using DPI for classification of the training data for Machine Learning Algorithms was used in [4]. Moreover, the DPI classification is quite slow and requires a lot of processing power [1], [5]. It relies on inspecting the user data and, therefore, privacy and confidentiality issues can appear [1]. Application signatures for every application must be created outside the system and kept up to date [1], which can be problematic. Encryption techniques in many cases make DPI impossible.

Using application ports [6], [7] is a very simple idea, widely used by network administrators to limit the traffic generated by worms and other unwanted applications. This method is very fast and it can be applied to almost all routers and layer-3 switches existing on the market. Besides its universality, it is very efficient to classify some protocols operating on fixed port numbers. Using it, however, gives very bad results in detection of protocols using dynamic port numbers, like P2P or Skype [1], [5], [8]. The second drawback is not less severe: many applications try to use well-known port numbers to be treated in the network with a priority.

III. Volunteer-Based System

A system is developed based on volunteers, which collects internet traffic data in flows together with the information, which application they belong to. The prototype version was called *Volunteer-based Distributed Traffic Data Collection System* and its architecture was described and analyzed in [9] and [10]. The design and the implementation of the prototype had numerous weaknesses and stability issues. Therefore a new reimplemention

The authors work in the Section for Networking and Security, Department of Electronic Systems, Aalborg University, DK-9220, Aalborg East, Denmark. Emails: {tbu, kba, tahir, jens}@es.aau.dk

of system has been made, later called Volunteer-Based System (VBS). Both the prototype as the VBS were developed in Java, using the Eclipse environment, resulting in a cross-platform solution. Currently only Microsoft Windows (XP and newer) and Linux (all versions) are supported because of used third-party libraries and helper applications. The system consists of clients installed on volunteers' computers, and of a server responsible for storing the collected data.

he task of the client is to register the information about each data packet passing the Network Interface Card (NIC). Captured packets are categorized into flows, with the exception of traffic to and from the local network (file transfers between local peers are filtered out). The following attributes of the flow are captured: start and end times of the flow, number of packets contained by the flow, local and remote IP addresses, local and remote ports, transport layer protocol, name of the application, and name of the client associated with the flow. The client also collects information about all the packets associated with each flow: direction, size, TCP flags, and relative timestamp to the previous packet in the flow. Information is then transmitted to the server, which stores all the data for further analysis. The client consists of 4 modules running as separate threads: packet capturer, socket monitor, flow generator, and data transmitter.

Both the VBS client and the VBS server are designed to run in the background and to start automatically together with the operating system (as a Windows service or a Linux daemon). The prototype uses the free community version of Tanuki Java Service Wrapper [11], which provides support only for 32-bit JVMs, and which requires special packaging of the Java application and placement of the libraries. To avoid these limitations, it has been replaced with YAJSW [12], an open-source project that provides support for both 32-bit and 64-bit versions of Windows and Linux.

A. Packet capturer

External Java libraries for collecting packets from the network rely on the already installed Winpcap (on Windows) or libpcap (on Linux), which makes the operating system dependency issue transparent to the application. The Jpcap [13] library used in the prototype is not suitable for processing packets from high-speed links, because transfers with rates higher than 3 MB/s cause Java Virtual Machine (JVM) to crash. Moreover, the *loopPacket* and the *processPacket* functions are broken causing random JVM crashes, so the only possibility is to process the packets one by one using *getPacket* (this bug is fixed in a new project called Jpcapng [14] evolved from Jpcap). Jpcap has not been developed since 2007 and Jpcapng since 2010, so there is no chance to get the bugs corrected. Therefore, we chose jNetPcap [15] as it contains even more useful features than Jpcap offered, such as detecting and stripping different kinds of headers (data-link, IP, TCP, UDP, HTTP) in the processed packets. It allows the client to capture packets on all the interfaces, not only on the Ethernet ones like the prototype, where the client needed to know the number of stripped bytes. jNetPcap is also able to filter out the local subnet traffic on the Pcap level by compiling dynamically Pcap filters, which saves system memory and CPU power.

B. Socket monitor

The socket monitor calls the external socket monitoring tool every second to ensure that even very short flows are registered. In the prototype, the built-in Windows or Linux Netstat was used, but it takes up to 20 seconds for Windows Netstat to display the output on some machines. We tried to solve this issue by using CurrPorts [16] instead of Netstat on Windows. Unfortunately, the only way to export the socket information was to write it to a file on the hard disk. It resulted in poor performance due to excessive disk reads and writes when executing CurrPorts each second. Finally, we chose Tcpviewcon, a console version of TCPView [17]. Tcpviewcon displays the information about the sockets in the console in a Netstat-like view, which allows us to process this information in the same manner as using Netstat. Using the external tools brings some licensing issues. These third-party applications must not be redistributed along with the VBS, but they need to be downloaded by the installer on the users' computers after accepting their license agreements.

VBS monitors both TCP and UDP sockets, contrary to the prototype, which was able to handle only TCP sockets. TCP sockets include the information about both end-points (local and remote) because a connection is established, while UDP sockets only provide the information about the local host. Since only one application can listen on a given UDP port at a time, the information about the local IP address and the local port are fully sufficient to obtain the application name for the given flow. Nevertheless, it is not possible to use the information about the UDP socket to terminate the flow, because many flows to different remote points can coexist using one UDP socket. Therefore, UDP flows are always closed based on timeout. TCP sockets are created on a one-per-connection basis, so it is possible to precisely assign a socket to a flow and close the flow when the matching socket is closed.

C. Flow generator

Collected packets are grouped into flows. If the application name can be received from the matching socket, it is assigned to the flow. When the flow is closed (the matching socket is closed or the flow is timed out in case the flow is not mapped to any socket), it is stored in the memory buffer. The prototype treated the flow and the packet data as raw byte arrays and stored them as binary files. However, it was impossible to detect the corruption of files or to look into the file to see what went wrong without binary file parsers. Therefore, we decided to use SQLite [18], which uses the proper data types (like integer, double, string) for all the captured information.

D. Data Transmitter

Before transmitting the data, the client authenticates itself to the server using a hardcoded plain-text password and obtains an identifier. The communication between the clients and the server uses raw sockets. The node authentication and the data transmission require separate connections between the clients and the server. When a

sufficient number of flows are stored in the local database (the database exceeds 700 kB), the SQLite database file is transmitted and stored on the VBS server. The transmitted database file also includes the client identifier and the information about the operating system installed on the client machine.

E. Implementation of the Server

The prototype server was based on threads. It received binary files from the clients and stored them in a separate directory for each client. The VBS server is also based on threads, however, it stores the collected data differently. The first thread authenticates the clients and assigns identifiers to them. The second thread receives files from clients and stores them in a common folder, which is periodically checked by the third thread. The files are checked for corruption and the proper SQLite database format, then they are extracted into the database. A synchronization method is used to avoid a situation where the third thread tries to process a file, which was not transferred completely – the extension of the file is changed after the file transfer is successful. The server uses the community edition of MySQL as the database, as it is quite fast and reliable for storing significant amounts of data.

IV. Initial Runs

The system was implemented and tested over a period of 3 months, to test its feasibility and usefulness in collecting valid data. The server was installed at Aalborg University on a PC equipped with an Intel Core i3 550 / 3.2 GHz processor, 4 GB of DDR3 SDRAM memory, and 70 GB hard disk and using Ubuntu 11.04 as OS. The clients were installed on 4 computers placed in private houses in Denmark and in Poland as well as on 23 machines installed in computer classrooms in *Gimnazjum nr 3 z Oddziałami Integracyjnymi i Dwujęzycznymi imienia Karola Wojtyły w Mysłowicach*, a high school in Poland. The computers used for the test were equipped with various hardware and operating systems. The objective was to prove that the system has high uptime, collects data from remotely located clients, and does not consume too much resources. The CPU usage by the VBS fluctuates with the average of around 1.7 % depending on the current rate of the traffic in the network. The CPU usage on the computers participating in our tests is shown in Fig. 1. To avoid the complexity of illustrating the CPU usage over the long time of the experiment, we illustrated the occurrence rate of the CPU consumption by each client. As it is shown, the CPU consumption in most cases amounts to 5 % or less, while the consumption of 10 % or more is extremely rare. During the experiment, no JVM errors about exceeding the default allocated memory size occurred, so we assume that VBS is free of memory leaks. The average memory usage on all the tested machines was below 5 % of the installed system memory. The minimum required amount of system memory is 64 MB because of the requirements of the Java service wrapper YAJSW. Disk space usage varied depending on the scheduling.

The test results were obtained during around 3 months, and during this time, the clients analyzed 121.21 GB of

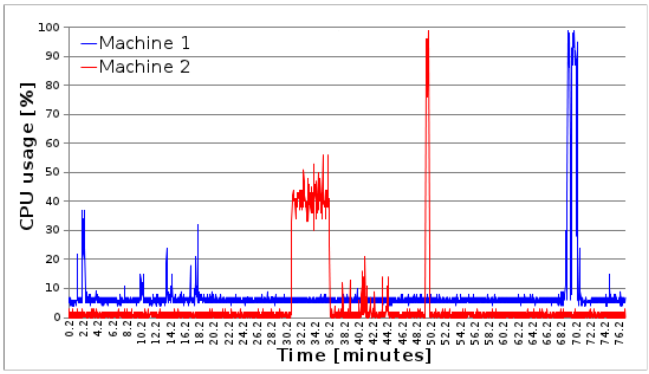

Fig. 1. CPU usage by VBS client on two computers

Internet traffic data (accumulated data from all clients). On the server side, 7.4 GB of statistical data were collected, consisting of 637.8 MB of flows data (3,506,201 records), and 6.8 GB of packets data (175,970,365 records). Communication between the VBS client and the server passes the network adapter as an ordinary remote connection, so it appears also in the database and is subject to be included in the classification. During the test period, 5 % of the collected data correspond the communication data between the client and server of the VBS. Around 14 % of flows were collected without associated application name. Flows without application name contain 5 packets in average comparing to flows with application name containing 49 packets in average. It means that the matching sockets for short flows were not noticed by the socket monitor due to very short opening times.

An example of the stored flows data is shown in Table 1. IP addresses were hidden for privacy reasons, start and end times of the flow (stored as Unix timestamps) were cut due to their length. This table also depicts a very interesting behavior of Skype – while the main voice stream is transmitted directly between the peers using UDP, there are plenty of TCP and UDP conversations with many different points (originally it was found to be around 50). The reason for this could be that the Skype user directory is decentralized and distributed among the clients.

Together with each flow, information about every packet belonging to this flow is registered. One TCP conversation is presented in Table 2, some non-relevant packets are omitted to save the space. This example shows that all the parameters are correctly collected. Thanks to such detailed flow description, it can be used as a base for creating numerous different statistics, which can then be used as an input for Machine Learning Algorithms. Moreover, presence of packet size and relative timestamp enables us to re-create characteristics of this traffic and, therefore, also behavior of the application associated with the flow. Relative timestamps show how much time passed from the previous packet in the flow.

V. Conclusion

This paper presents a novel volunteer-based system for collecting network traffic data, which was implemented and tested on 16 volunteers during around 3 months. Obtained results proved that the system is feasible and capable of

TABLE 1: EXAMPLE OF THE STORED FLOWS DATA

flow id	client id	start time	end time	no. of packets	local IP	remote IP	local port	remote port	protocol name	socket name
1	1	13...	13...	40	192.x.x.x	213.x.x.x	1133	110	TCP	thebat.exe
6	1	13...	13...	10	192.x.x.x	74.x.x.x	1151	80	TCP	opera.exe
7	1	13...	13...	10	192.x.x.x	91.x.x.x	1138	80	TCP	opera.exe
46012	1	13...	13...	20	192.x.x.x	85.x.x.x	23399	45527	TCP	Skype.exe
46013	1	13...	13...	20	192.x.x.x	78.x.x.x	23399	3598	TCP	Skype.exe
46014	1	13...	13...	11	192.x.x.x	41.x.x.x	23399	10050	TCP	Skype.exe
46015	1	13...	13...	15	192.x.x.x	41.x.x.x	23399	10051	TCP	Skype.exe
46016	1	13...	13...	207457	192.x.x.x	62.x.x.x	23399	14471	UDP	Skype.exe
46021	1	13...	13...	3	192.x.x.x	183.x.x.x	23399	33033	UDP	Skype.exe

TABLE 2: CHOSEN PACKETS FROM ONE STORED TCP CONVERSATION

flow id	direct.	packet size	SYN	ACK	PSH	FIN	RST	CWR	ECN	URG	relative timestamp
18	OUT	48	1	0	0	0	0	0	0	0	0
18	IN	48	1	1	0	0	0	0	0	0	134160
18	OUT	40	0	1	0	0	0	0	0	0	200
18	OUT	105	0	1	1	0	0	0	0	0	20262
18	IN	77	0	1	1	0	0	0	0	0	79654
18	OUT	43	0	1	1	0	0	0	0	0	1651
18	IN	58	0	1	1	0	0	0	0	0	66950
18	OUT	40	0	1	0	0	0	0	0	0	175472
18	OUT	40	0	1	0	1	0	0	0	0	1031011
18	IN	40	0	1	0	0	0	0	0	0	69279
18	IN	40	0	1	0	1	0	0	0	0	250
18	OUT	40	0	1	0	0	0	0	0	0	25

providing detailed information about the network traffic. Therefore, it can be useful for creating traffic profiles of different applications. VBS is a field for constant improvements, like implementing sufficient security in the system, as for now only plain-text pass-phrases are used for authentication and raw sockets are used for communication. An intelligent transfer protocol should be developed and implemented, which allows the negotiation of some link parameters, and scheduling transfers to effectively use the link capacity.

Another quite important drawback is the inability to distinguish between different types of traffic generated by the same application. For instance, web browser deals with various content: web requests, flash movies, radio transmissions, and FTP file transfers. All of them are treated now in the same manner. The solution can be to look into dynamic linking issues and try to get information about the library, which directly deals with the flow.

VBS requires a valid IPv4 address to listen on the network interface, but also IPv6 is planned to be supported. Another issue arises when an encapsulation, data tunneling, or network file systems (like SAMBA, NFS) are used. Then, only the most outer IP and TCP/UDP headers are inspected. The next issue is the lack of application names for short flows. Volunteers' privacy also must be protected in a better way, for example, by avoiding storing IP addresses in a clear text. An upgrade system should be developed to support automatics upgrades of VBS when a new version is available.

Another concern is about finding a large enough group of participating volunteers to be able to receive data for all the relevant applications because of privacy issues. This issue is not resolved so far, but we suspect to convince the users to install the software if it can provide some useful information to the user, like statistics about the amounts of traffic belonging to the particular groups of applications.

REFERENCES

[1] Jun Li, Shunyi Zhang, Yanqing Lu, and Junrong Yan. Real-time P2P traffic identification. In *Proceedings of the IEEE Global Telecommunications Conference (IEEE GLOBECOM 2008)*, pages 1–5. IEEE, New Orleans, Louisiana, USA, December 2008. DOI: 10.1109/GLOCOM.2008.ECP.475.

[2] Riyad Alshammari and A. Nur Zincir-Heywood. Unveiling Skype encrypted tunnels using GP. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, Barcelona, Spain, July 2010. DOI: 10.1109/CEC.2010.5586288.

[3] L7-filter Supported Protocols, 2012. [Online]. Available: http://l7-filter.sourceforge.net/protocols.

[4] Wei Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07)*, pages 310–317. IEEE, Istanbul, Turkey, October 2007. DOI: 10.1109/MASCOTS.2007.2.

[5] Ying Zhang, Hongbo Wang, and Shiduan Cheng. A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 1192–1195. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689126.

[6] Riyad Alshammari and A. Nur Zincir-Heywood. Machine Learning based encrypted traffic classification: identifying SSH and Skype. In *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA 2009)*, pages 1–8. IEEE, Ottawa, Ontario, Canada, July 2009. DOI: 10.1109/CISDA.2009.5356534.

[7] Sven Ubik and Petr Žejdl. Evaluating application-layer classification using a Machine Learning technique over different high speed networks. In *Proceedings of the Fifth International Conference on Systems and Networks Communications (ICSNC)*, pages 387–391. IEEE, Nice, France, August 2010. DOI: 10.1109/ICSNC.2010.66.

[8] Jing Cai, Zhibin Zhang, and Xinbo Song. An analysis of UDP traffic classification. In *Proceedings of the 12th IEEE International Conference on Communication Technology (ICCT)*, pages 116–119. IEEE, Nanjing, China, November 2010. DOI: 10.1109/ICCT.2010.5689203.

[9] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, and Jens Myrup Pedersen. Volunteer-Based Distributed Traffic Data Collection System. In *Proceedings of the 12th International Conference on Advanced Communication Technology*

(ICACT 2010), volume 2, pages 1147–1152. IEEE, Phoenix Park, PyeongChang, Korea, February 2010.

[10] Kartheepan Balachandran and Jacob Honoré Broberg. Volunteer-Based Distributed Traffic Data Collection System. Master's thesis, Aalborg University, Department of Electronic Systems, Denmark, June 2010.

[11] Java Service Wrapper – Tanuki Software, 2011. [Online]. Available: http://wrapper.tanukisoftware.com/doc/english/download.jsp.

[12] YAJSW – Yet Another Java Service Wrapper, 2011. [Online]. Available: http://yajsw.sourceforge.net/.

[13] Jpcap – a Java library for capturing and sending network packets, 2007. [Online]. Available: http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/index.html.

[14] Jpcapng – fork of Jpcap, aka Jpcap 0.8, 2010. [Online]. Available: http://sourceforge.net/projects/jpcapng/.

[15] jNetPcap OpenSource | Protocol Analysis SDK, 2011. [Online]. Available: http://jnetpcap.com/.

[16] CurrPorts, Monitoring opened TCP/IP network ports / connections, 2011. [Online]. Available: http://www.nirsoft.net/utils/cports.html.

[17] TCPView for Windows, 2011. [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb897437.

[18] SQLite, Self-contained, serverless, zero-configuration, transactional SQL database engine, 2011. [Online]. Available: http://www.sqlite.org/.