Volunteer-Based System for classification of traffic in computer networks

Bujlow, Tomasz; Balachandran, Kartheepan; Riaz, M. Tahir; Pedersen, Jens Myrup

Published in:
Proceedings of 19th Telecommunications Forum TELFOR 2011

Publication date:
2011

Document Version
Accepted author manuscript, peer reviewed version

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 network is usually done to investigate 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 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 limited number of applications, which use fixed port numbers or contain characteristic patterns in the payload. Therefore a system is designed, built and tested, in which the data are collected directly from the client machines, where volunteers contribute with traffic data. In the following available and chosen solutions are described for particular parts of the system. With relatively long testing, the system has shown to be 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 design and the new implementation of the volunteer-based system. Finally, it shows the results of 3-months system test and proposes further enhancements.

II. Related work

Most of the methods for traffic classification use the concept of flow, which is defined as a group of packets having the same end IP addresses, ports, and using 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 all the captured packets should correspond to that application. Nevertheless, this method requires an application installed on the host, which is run once for each application where traffic characteristics has to be captured. This solution is slow and not scalable. Secondly, all the operating systems have usually background processes, which damage the clear application traffic image (DNS requests and responses, system or program upgrades).

DPI solution by a layer-7 filter and statistical classification solution are proposed in [2]. Using DPI is much more convenient than the previous method, since it can collect the data in any point in the network. Unfortunately existing DPI tools are not able to classify accurately traffic belonging to some applications, like Skype (in this case the layer-7 tools rely on statistical information instead of the real traffic patterns, giving some false positives and false negatives). Obtaining training data for statistical classifier basing on statistical classifiers will not give us a high accuracy of the new classifier. Idea of using DPI for classification of the training data for Machine Learning Algorithms was used in [3]. Moreover the DPI classification is quite slow and requires a lot of processing power [1], [4]. Furthermore 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], what can be problematic. Worse, encryption techniques make DPI in many cases impossible.

Using application ports [5], [6] is a very simple idea, widely used by network administrators to limit traffic generated by worms and other unwanted applications. This method is very fast, and can be applied to almost all the routers and layer-3 switches existing on the market. Apart from 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 and Skype [1], [4], [7]. The second drawback is not less severe: many applications try to use well-known port numbers to be treated in the network with priority.

III. Volunteer-Based System

A system is developed based on volunteers, which collects internet traffic data in flows together with the information, what application they belong to. A prototype version of a volunteer based system and its architecture were described and analyzed in [8] and [9]. Design and implementation of the prototype had numerous weaknesses and stability issues. Therefore a new reimplemention of system has been made, later called Volunteer-Based System (VBS). Both the prototype as the VBS were

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

developed in Java, using 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.

The task of the client is to register information about each internet traffic data packet passing the Network Interface Card (NIC), and categorize them into flows, with the exception of traffic to and from the local network (file transfers between local peers are filtered out). The following flow attributes are captured: start and end time of the flow, number of packets, local and remote IP addresses, local and remote ports, used transport layer protocol, name of the application and client, which the flow belongs to. The client collects also 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 a separate thread: packet capturer, socket monitor, flow generator and data transmitter.

Both VBS client and server are designed to run in the background and start automatically with the operating system (as Windows service or Linux daemon). The prototype uses a free community version of Tanuki Java Service Wrapper [10], which provides support only for 32-bit JVMs and requires special packaging of the Java application and placement of the libraries. To avoid these limitations, this has been replaced with YAJSW [11], an open-source project providing 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), making the operating system issue transparent to the application. The Jpcap [12] library used in the prototype is not suitable for processing packets from high-speed links, because transfers above 3 MBs 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 packets one by one using *getPacket* (this bug is fixed in a new project called Jpcapng [13] evolved from Jpcap). Jpcap is not developed since 2007 and Jpcapng since 2010, so there is no chance to get the bugs corrected. Therefore jNetPcap [14] is chosen, as it contains even more useful features than Jpcap offered, like detecting and stripping different kinds of headers (data-link, IP, TCP, UDP) in processed packets. It allows the client to capture packets on all the interfaces, not only on the Ethernet ones like in the prototype, where the client needed to know number of stripped bytes. jNetPcap is also able to filter out the local subnet traffic on the Pcap level by compiling dynamically Pcap filters, what saves system memory and CPU power.

B. Socket monitor

The socket monitor executes external socket monitoring tool every second, to ensure that even very short flows are noticed. In the prototype the built-in Windows or Linux Netstat was used, but it takes up to 20 seconds for Windows Netstat on some machines to display the output. This issue was tried to be solved with CurrPorts [15] on Windows instead of Netstat. Unfortunately the only way to export the socket information was to write it to a file on the harddisk. It resulted in poor performance due to excessive disk reads and writes when executing CurrPorts each second. Finally tcpviewcon was chosen, a console version of TCPView [16]. Tcpviewcon displays socket information in the console in a Netstat-like view, allowing us to process this information in the same manner. Using the external tools brings some licensing issues. These third-party applications must not be redistributed along the VBS, but they need to be downloaded by the installer on the users' computer after accepting their license agreement.

VBS monitors both TCP and UDP sockets, contrary to the prototype, which was able to handle only TCP sockets. TCP sockets include information about both end-points (local and remote) because the connection is established, and UDP sockets provide information only about the local node. Since only one application can listen on a given UDP port at a time, information about local IP address and local port are fully sufficient to obtain the application name for the given flow. Nevertheless, it is not possible to use the UDP socket information 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 one-per-connection, 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 (matched socket is closed or time-out appears in case if the flow is not mapped to any socket), it is stored in the memory buffer. The prototype treated the flow and packet data as raw byte arrays and stored them as binary files. Unfortunately, it was impossible to detect file corruption or to look into the file to see what went wrong without binary file parsers. Therefore it is decided to use SQLite [17], which uses the proper data types (like integer, double, string) for all the information captured.

D. Data transmitter

Before transmitting the data the client authenticates to the server using hardcoded plain-text pass-phrase and obtains an identifier. Communication between clients and the server uses raw sockets. Node authentication and data transmission require separate connections between the clients and the server. When 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 includes also

client identifier and information about operating system installed on the client machine.

E. Implementation of the server

The prototype server is based on threads and received the binary files and stored them in a separate directories for each client. The VBS server is also based on threads, however 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. To avoid a situation where the third thread tries to process a file, which was not transferred completely, synchronization is used where the file extension is changed after the file transfer is successfull. The server uses a Non-commercial MySQL as the database, as it is quite fast and reliable for storing a significant amount of data.

IV. Initial runs

The system was implemented and tested over a 3 month period, 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 16 computers, equipped with different hardware and operating systems, placed in private houses as well as in an educational institution in Poland. 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 of the VBS, fluctuates with the average of around 6 % depending on the current network traffic. It was observed that during high-speed transfers (50 Mbit/s or faster) VBS starts to use 100 % of the CPU time. Results of CPU usage tests on two computers with different hardware and OS configurations are shown on Fig. 1. Sampling was performed on each 3 seconds during 76 minutes. CPU usage on the first machine (Intel Core 2 Duo 7250 @ 2.0 GHz, Kubuntu 11.04) is higher than on the second (Intel Core i3 550 @ 3.2 GHz, Windows 7) because the processing power of the first machine is significantly lower. During the 3 month experiment no JVM errors occurred about exceeding the default allocated memory size, so it is assumed that the VBS is free from memory leaks. Average memory usage on all the tested machines was below 5 % of installed system memory. The minimum required amount of system memory is 64 MB because of 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 in this time the clients analysed 121.21 GB of internet traffic data (accumulated data from all clients). On the server side 7.4 GB of statistical data was 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

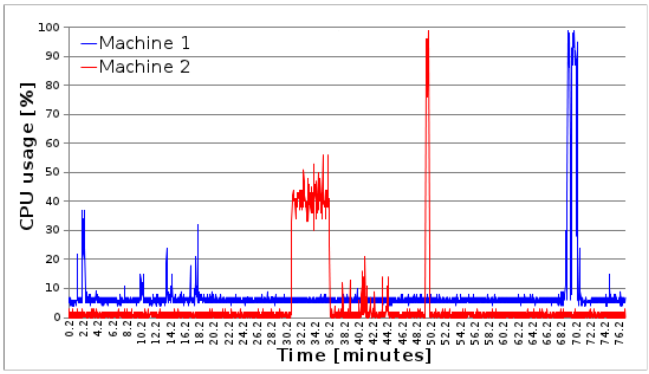

Fig. 1. CPU usage by VBS client on two computers

connection, so it appears also in the database and is a 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 contains 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 time.

Example of the stored flows data is shown in Table 1. IP addresses were hidden for the privacy reason, start and end time 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 timestamp shows, how much time passed from the previous packet in the flow.

V. Conclusion

The 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 providing detailed information about the network traffic. Therefore, it can be useful for creating different applications traffic profiles. VBS is a field for constant improvements, like implementing sufficient security in the system, as for now only plain-text pass-phrase is used for authentication, and raw sockets are used for

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

communication. An intelligent transfer protocol should be developed and implemented, which allows negotiation of some link parameters, and scheduling transfers to effectively use the link capacity.

Another quite important drawback is 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, 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 consider lack of application name for the short flows, Volunteers' privacy also must be protected in a better way, for example by avoiding to store 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 suspect to convince the users to install the software if it can provide some useful information to the user, like statistics about amount of traffic belonging to particular group of applications.

REFERENCES

[1] Jun Li, Shunyi Zhang, Yanqing Lu, Junrong Yan, *Real-time P2P traffic identification*, IEEE GLOBECOM 2008 PROCEEDINGS.

[2] Riyad Alshammari, A. Nur Zincir-Heywood, *Unveiling Skype encrypted tunnels using GP*, IEEE 2010.

[3] Wei Li, Andrew W. Moore, *A Machine Learning approach for efficient traffic classification*, Proceedings of the Fifteenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'07), IEEE 2008, pp. 310–317.

[4] Ying Zhang, Hongbo Wang, Shiduan Cheng, *A Method for Real-Time Peer-to-Peer Traffic Classification Based on C4.5*, IEEE 2010, pp. 1192–1195.

[5] Riyad Alshammari, A. Nur Zincir-Heywood, *Machine Learning based encrypted traffic classification: identifying SSH and Skype*, Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA 2009).

[6] Sven Ubik, Petr Žejdl, *Evaluating application-layer classification using a Machine Learning technique over different high speed networks*, 2010 Fifth International Conference on Systems and Networks Communications, IEEE 2010, pp. 387–391.

[7] Jing Cai, Zhibin Zhang, Xinbo Song, *An analysis of UDP traffic classification*, IEEE 2010, pp. 116–119.

[8] Kartheepan Balachandran, Jacob Honoré Broberg, Kasper Revsbech, Jens Myrup Pedersen, *Volunteer-based distributed traffic data collection system*, Feb. 7-10, 2010 ICACT 2010, pp. 1147–1152.

[9] Kartheepan Balachandran, Jacob Honoré Broberg, *Volunteer-based distributed traffic data collection system*, Master Thesis at Aalborg University, Department of Electronic Systems, June 2010.

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

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

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

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

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

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

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

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