Volunteer-Based System for Research on the Internet Traffic

Tomasz Bujlow, Kartheepan Balachandran, Sara Ligaard Nørgaard Hald,
Tahir Riaz, and Jens Myrup Pedersen

Abstract* — To overcome the drawbacks of existing methods for traffic classification (by ports, Deep Packet Inspection, statistical classification) a new system has been developed, in which data are collected and classified directly by clients installed on machines belonging to volunteers. Our approach combines the information obtained from the system sockets, the HTTP content types, and the data transmitted through network interfaces. It allows grouping packets into flows and associating them with particular applications or types of service. This paper presents the design of our system, implementation, the testing phase and the obtained results. The performed threat assessment highlights potential security issues and proposes solutions in order to mitigate the risks. Furthermore, it proves that the system is feasible in terms of uptime and resource usage, assesses its performance and proposes future enhancements. We released the system under *The GNU General Public License v3.0* and published as a SourceForge project called *Volunteer-Based System for Research on the Internet.

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

I. Introduction

MONITORING of 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 also be performed 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 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 data directly from the machines belonging to volunteers who contribute with traffic data. For particular parts of the system, we described 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 a 3-month system test and proposes further enhancements.

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

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 [2] 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 the statistics of application traffic.

A DPI solution using a layer-7 filter and a statistical classification solution are proposed in [3]. Using DPI is much more convenient than the previous method, as it can collect 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 the layer-7 tools rely on statistical information instead of the real traffic patterns, giving some false positives and false negatives [4]). Obtaining 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 the classification of training data for Machine Learning Algorithms was used in [5]. Moreover, the DPI classification is quite slow and requires a lot of processing power [2], [6]. It relies on inspecting the user data and therefore privacy and confidentiality issues can appear [2]. Application signatures for every application must be created outside the system and kept up to date [2], which can be problematic. Encryption techniques in many cases make DPI impossible.

Using application ports [7], [8] 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 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 the detection of protocols using dynamic port numbers, like P2P or Skype [2], [6], [9]. 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

We developed a system which collects flows of Internet data traffic together with the information about the application associated with each flow. The prototype version was called *Volunteer-based Distributed Traffic Data Collection System* and its architecture was described and analyzed in [10] and [11]. The design and implementation of the prototype had numerous weaknesses and stability issues. Therefore, a new implementation of the system was made, called Volunteer-Based System (VBS). The new, reimplemented version of the VBS was released under The *GNU General Public License v3.0* and published as a SourceForge project. The website [12] contains a broad description of the project illustrated with screenshots, roadmap, binary packages, source code in the Git repository, comprehensive documentation of the source code, and a system for bug tracking and feature requests. Both the prototype and VBS were developed in Java, using Eclipse environment, and that resulted in a cross- platform solution. Currently only Microsoft Windows (XP and newer) and Linux are supported because of third-party libraries and helper applications used in development. 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 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: the start and the end time of the flow, the number of packets contained by the flow, the local and the remote IP addresses, the local and the remote ports, the transport layer protocol, the name of the application and the name of the client associated with the flow. The client also collects information about all the packets associated with each flow: the direction, the size, the TCP flags, and the relative timestamp to the previous packet in the flow. One transport-layer flow can contain multiple application-layer streams of HTTP data, and each of them can carry different kinds of content, such as audio, video, or a file transfer. For that reason, packets belonging to flows which contain HTTP content require additional information to be collected. Therefore, in this case, we additionally store the information about the content type for each packet of the flow. In fact, the information about the content type is present only in the first packet of the response made to an HTTP request. It means that for each HTTP request we have one packet containing the information about the content type, which allows us to logically split all the application-layer HTTP streams. The collected data are periodically transmitted to the server, which stores all the data for further analysis. The client consists of 4 modules running as separate threads: the packet capturer, the socket monitor, the flow generator, and the 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 [13], 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 [14], an open-source project that provides support for both 32-bit and 64-bit versions of Windows and Linux.

We implemented a fully automatic update system for VBS clients. To simplify the update process, we introduced three different version numbers in our software; the first one is associated with the client, the second one with the server, and the third one with the structure of the SQLite database used for exchanging the information between the client and the server. The update is stored on the server. While registering on the server, the client asks if any update is available, and downloads it if possible. The update is automatically installed by a script executed by YAJSW during the next restart of the VBS client.

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 [15] library used in the prototype is not suitable for processing packets from high-speed links, because transfers with a rate 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 packets one by one using *getPacket* (this bug is fixed in a new project called Jpcapng [16] 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 [17] 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 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 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. Contrary to processing each packet separately by the prototype of VBS, we decided to use the native function of Winpcap or libpcap called *loopPacket*. It allowed lowering the usage of the resources consumed by the VBS client. It is worth mentioning that the Winpcap library tends to crash when the computer is placed into standby mode, sleep mode, or hibernation. Therefore the packet capturer needs to be continuously monitored and restarted in case of a crash. A need to restart the capturer also appears when new interfaces are detected in the system, for example, when a

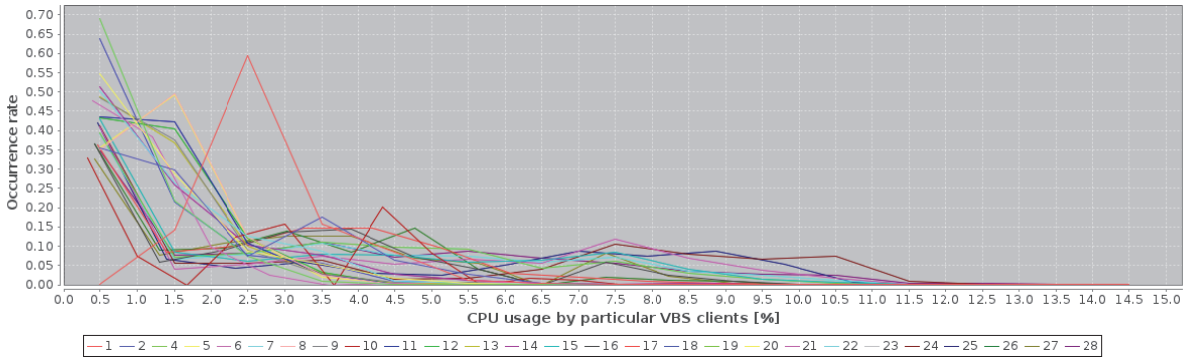

Fig. 1. The CPU usage by VBS client on test computers.

network cable is connected, or the switch of a wireless card is enabled. If an IP address on an interface is changed, the packet capturer needs to be restarted as well to prevent confusion with recognizing local and remote IP addresses.

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 [18] 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 [19]. Tcpviewcon displays the information about the sockets in the console in a Netstat-like view, what 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' 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 a connection is established, while UDP sockets only provide the information about the local node. 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 (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 [20], 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 itself to the server using a hardcoded plain-text password and obtains an identifier. Communication between the clients and the server uses raw sockets. The node authentication and 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. *The implementation of 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 a proper SQLite database format, then they are extracted into the database. Synchronization 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. Testing Phase

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

TABLE 1: STATISTICS OF THE OPERATIONS OF THE VBS CLIENTS.

Client id	*Average CPU usage [%]*	*Captured traffic [GB]*	*Traffic belonging to VBS [%]*	*Number of captured flows*	*UDP flows [%]*	*Short flows below 20 packets [%]*	*Flows without the application name [%]*
1	2.4	43.64	4.46	843552	32.00	85.57	61.28
2	1.1	0.04	2.04	149	0.00	65.77	23.49
4	0.5	248.95	4.10	3587227	51.42	86.12	3.05
5	1.0	29.09	4.75	1184464	46.82	92.56	1.58
6	0.8	0.08	4.90	1009	0.00	69.67	9.91
7	0.9	0.17	6.36	6943	8.44	76.51	13.38
8	0.9	0.34	10.39	16671	1.83	80.64	40.35
9	0.7	0.14	5.30	3084	0.29	71.07	18.00
10	2.2	0.02	5.31	774	0.52	73.00	19.12
11	0.7	0.19	6.82	4552	1.38	65.66	8.37
12	0.8	0.43	9.07	12000	0.77	80.19	27.79
13	0.8	0.11	6.81	4324	0.48	75.13	7.61
14	0.9	0.24	5.31	6186	1.00	69.95	9.80
15	2.6	0.04	11.51	5724	0.05	84.03	37.61
16	2.6	0.03	9.08	2227	0.27	75.12	23.53
17	2.3	0.08	9.39	5736	0.14	72.70	27.63
18	0.6	0.50	6.24	16158	0.03	67.81	13.67
19	2.6	0.44	9.04	28575	0.08	74.80	29.18
20	0.8	0.16	6.51	4905	0.20	75.37	14.82
21	3.2	0.39	8.01	22572	0.00	69.76	27.16
22	2.8	0.39	7.41	24389	0.00	72.58	31.77
23	0.4	0.19	4.95	5664	0.04	67.07	8.33
24	3.7	0.10	5.05	2692	0.00	73.59	33.51
25	3.5	0.06	3.20	1352	0.00	71.52	38.76
26	2.3	0.02	3.23	1056	0.00	69.22	12.69
27	2.7	0.04	5.74	1414	0.00	66.62	19.87
28	2.7	0.02	6.94	419	0.00	74.41	63.96

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 a 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 traffic in the network. The CPU usage on 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 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 occurred about exceeding the default allocated memory size, so we assume that the VBS is free from 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 requirements of the Java service wrapper YAJSW. Disk space usage varied depending on scheduling.

The test results were obtained during around 3 months, and in this time the clients analyzed 325.88 GB of Internet traffic data (accumulated data from all clients). On the server side 22.8 GB of statistical data was collected, consisting of 0.9 GB of flows data (5,799,207 records), and 21.9 GB of packets data (446,987,507 records). Communication between the VBS client and the server passes a network adapter as an ordinary remote connection, so it also appears in the database and is a subject to be included in the classification. During the test period, 4.21 % of collected data correspond to the communication between the VBS client and the VBS server. Short flows, which contain less than 20 packets, represent 87 % of the total number of flows. Around 12 % of flows were collected without the associated application name. TCP flows without the application name contain 13 packets on the average compared to flows with the application name containing 74 packets on the average. It means that the matching sockets for short flows were not registered by the socket monitor due to a very short opening time. This rule does not apply to UDP flows because all the flows created by one application use the same UDP socket in the system. It means that either all or none flows associated with the socket have assigned an application name, regardless of the length of the particular flow. Detailed statistics on per- client basis are shown in Table 1.

An example of stored flows data is shown in Table 2. The IP addresses were hidden for privacy reasons, the start and the 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

TABLE 2: EXAMPLE OF STORED FLOWS DATA, TIMESTAMPS ARE STRIPPED DUE TO THEIR LENGTH.

Flow id	*Client id*	*Start time*	*End time*	*No. of packets*	*Local IP*	*Remote IP*	*Local port*	*Remote port*	*Protocol name*	*Socket application 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 3: CHOSEN PACKETS FROM ONE STORED TCP COMMUNICATIONS.

Flow id	*Direction*	*Packet size [B]*	*SYN*	*ACK*	*PSH*	*FIN*	*RST*	*CWR*	*ECN*	*URG*	*Relative timestamp [μs]*	*Content type*
18	OUT	48	1	0	0	0	0	0	0	0	0	1
18	IN	48	1	1	0	0	0	0	0	0	134160	1
18	OUT	40	0	1	0	0	0	0	0	0	200	1
18	OUT	105	0	1	1	0	0	0	0	0	20262	1
18	IN	77	0	1	1	0	0	0	0	0	79654	1
18	OUT	43	0	1	1	0	0	0	0	0	1651	1
18	IN	58	0	1	1	0	0	0	0	0	66950	1
18	OUT	40	0	1	0	0	0	0	0	0	175472	1
18	OUT	40	0	1	0	1	0	0	0	0	1031011	1
18	IN	40	0	1	0	0	0	0	0	0	69279	1
18	IN	40	0	1	0	1	0	0	0	0	250	1
18	OUT	40	0	1	0	0	0	0	0	0	25	1

directory is decentralized and distributed among the clients.

Together with each flow, the information about every packet belonging to this flow is registered. One TCP conversation is presented in Table 3, some non-relevant packets are omitted to save space. This example shows that all the parameters are correctly collected. Thanks to such a detailed flow description, it can be used as a base for creating numerous different statistics. These precise statistics can be used as an input to Machine Learning Algorithms (MLAs). We tried this approach in [21] and we classified traffic belonging to 7 different applications with accuracy of over 99 %. In [22] we used the VBS to distinguish different kinds of browser traffic, such as file download, web browsing, audio and video. The statistics obtained from particular groups of the traffic were provided as an input to MLAs. Moreover, the presence of packet size and the relative timestamp enables us to re-create the characteristics of this traffic, and therefore also the behavior of the application associated with the flow. The relative timestamp shows how much time passed from the previous packet in the flow.

V. THREAT ASSESSMENT

The Volunteer-Based System, as other distributed systems, is prone to various types of security attacks. For that reason, we made a deep analysis of potential risks, and we suggested solutions for mitigating or avoiding them. We started the assessment from finding all interfaces which can be used to interact with the system. These interfaces result from both the architecture of VBS and the use cases. Then, we composed the list of possible threats for each interface. For each threat we assessed its probability and severity on a scale from 1 to 5, where 1 means the lowest probability or severity, and 5 means the highest. Table 4 shows the disclosed threats and the assigned values.

The decision about a threat which needs to be handled is based on the impact matrix shown in Fig. 2 [23]. If the intersection of the probability and severity of the threat lies in cells marked with yellow color, the threat should be handled. However, if it lies in cells marked with blue color, the threat does not need to be handled. Threats belonging to cells marked with white color can be handled, but it is not required. We can see that only one threat requires handling in our system – the server needs to be protected from an SQL injection attack. There are several other issues which we can mitigate or avoid. The physical access is protected by placing the server in a safe room with attested locks, and a firewall can be used to protect the system from malicious connections. The threats to the system can also be further reduced by stripping the data of IP addresses or by using encryption.

VI. CONCLUSION

The paper presents a novel volunteer-based system for collecting network traffic data, which was implemented and tested on 27 volunteers during around 3 months. With relatively long testing, the system has shown to be feasible in terms of resource usage and uptime. Obtained results proved that the system is capable of providing valid detailed information about the characteristics of network

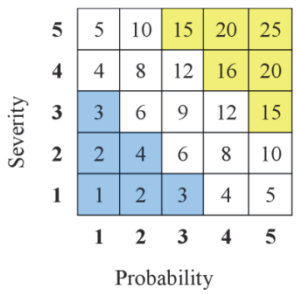

Fig. 2. The impact matrix [23].

TABLE 4: SECURITY LEAKS IN VBS WITH ASSIGNED PROBABILITY AND SEVERITY.

Interface / threat	*Probability (1–5)*	*Severity (1–5)*	*Handle*
The user interface			
A user can delete the local data storage	3	1	No
A user can pollute data in his local data storage	1	2	No
A user can destroy the VBS system	3	1	No
A user can modify the local data storage by adding SQL commands (SQL injection)	1	5	Maybe
An attacker can hijack user's computer and redirect the data to another machine	1	1	No
The server interface			
An attacker can get physical access to the server	1	5	Maybe
The network interface			
An attacker can inject polluted data to the server	3	4	Maybe
An attacker can inject data containing SQL commands to the server (SQL injection)	4	5	Yes
An attacker can perform a Denial of Service attack on the server	3	2	Maybe
An attacker can hack the server and modify or delete the database	2	5	Maybe
An attacker can hack the server and change the file used to upgrade clients, which can result in losing all VBS clients and transferring them into bots in attacker's botnet	1	5	Maybe
The communication between the clients and the server			
An attacker can sniff the communication between the client and the server	1	3	No
An attacker can pollute the data being sent by the client to the server	1	2	No
An attacker can modify the data being sent from the client to the server by adding SQL commands (SQL injection)	1	5	Maybe

traffic. Therefore, we can use the system to create profiles of traffic generated by different applications. The VBS is a field for constant improvements. As we assessed in the previous section, sufficient security needs to be implemented in the system. If it is possible, we should avoid using external tools (Netstat, TCPView), but extract information about open sockets directly from the system API. We need to consider developing an intelligent transfer protocol, which will allow to negotiate link parameters and to schedule transfers in order to effectively use the capacity of the link. A user-friendly installer will be the next step to make the system easier to use by non-qualified users.

VBS requires a valid IPv4 address to listen on a network interface, but IPv6 is also 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 the application name for short flows. Volunteers' privacy also must be protected in a better way, for example by avoiding to store IP addresses in a clear text. Another concern, due to privacy issues, is how to find a large enough group of participating volunteers to be able to receive data for all the relevant applications. This issue has not been resolved so far, but we believe that it will be easier to convince the users to install the software if it can provide some useful information to the user, like statistics about the amount of traffic belonging to a particular group of applications.

Finally, the collected data can be used to create an emulator of different applications, different groups of applications, the Internet traffic under certain conditions, or at selected points of time.

REFERENCES

[1] T. Bujlow, K. Balachandran, T. Riaz, and J. Myrup Pedersen, "Volunteer-Based System for classification of traffic in computer networks," *19th Telecommunications Forum TELFOR 2011, IEEE 2011*, pp. 210–213.

[2] J. Li, S. Zhang, Y. Lu, and J. Yan, "Real-time P2P traffic identification," *IEEE GLOBECOM 2008 PROCEEDINGS*.

[3] R. Alshammari and A. Nur Zincir-Heywood, "Unveiling Skype encrypted tunnels using GP," *IEEE 2010*.

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

[5] 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.

[6] Y. Zhang, H. Wang, and S. Cheng, "A Method for Real- Time Peer-to-Peer Traffic Classification Based on C4.5," *IEEE 2010*, pp. 1192–1195.

[7] R. Alshammari and 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)*.

[8] S. Ubik and P. 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.

[9] J. Cai, Z. Zhang and X. Song, "An analysis of UDP traffic classification," *IEEE 2010*, pp. 116–119.

[10] K. Balachandran, J. Honoré Broberg, K. Revsbech, and J. Myrup Pedersen, "Volunteer-based distributed traffic data collection system," *Feb. 7-10, 2010 ICACT 2010*, pp. 1147–1152.

[11] K. Balachandran and J. Honoré Broberg, "Volunteer-based distributed traffic data collection system," Master Thesis at Aalborg University, Department of Electronic Systems, June 2010.

[12] Volunteer-Based System for Research on the Internet. [Online]. Available: http://vbsi.sourceforge.net/

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

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

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

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

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

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

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

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

[21] T. Bujlow, T. Riaz, J. Myrup Pedersen, "A method for classification of network traffic based on C5.0 Machine Learning Algorithm," *International Conference on Computing, Networking and Communications (ICNC 2012), IEEE 2012*, pp. 244–248.

[22] T. Bujlow, T. Riaz, and J. Myrup Pedersen, "Classification of HTTP traffic based on C5.0 Machine Learning Algorithm," to appear in *Fourth IEEE International Workshop on Performance Evaluation of Communications in Distributed Systems and Wed based Service Architectures (PEDISWESA 2012)*.

[23] Office of Government Commerce, An Introduction to PRINCE2: Managing and Directing Successful Projects, 2009, ISBN: 978-0113311880, London: TSO.