Computer Networks 92 (2015) 41–54

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

User behavior based traffic emulator: A framework for generating test data for DPI tools

Péter Megyesi [a,*], Géza Szabó [b], Sándor Molnár [a]

[a] HSNLab, Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, Budapest, Hungary
[b] TrafficLab, Ericsson Research, Budapest, Hungary

ARTICLE INFO

Article history:
Received 16 July 2014
Revised 29 May 2015
Accepted 23 September 2015
Available online 3 October 2015

Keywords:
Deep packet inspection
User behavior emulation
Traffic generation

ABSTRACT

Deep Packet Inspection (DPI) engines rely highly on the operation environment i.e., the traffic mix they supposed to work with. A well performing DPI engine requires real-world traffic mixes to be tested on. Due to privacy issues real-world traffic is usually only available at the site of the network operator at a secure measurement point. Furthermore, in order to make signature update, performance tweaks, etc. of the DPI engine, real-like measurements are essential. In this paper we present a traffic generation framework that provides up-to-date traffic mixes continuously. The basic idea of the framework is to generate traffic based on automatic user behavior emulation. Real-world traffic measurements are processed to analyze and extract the most typical user behavior scenarios. Our proposed method uses these typical user behaviors for emulation of users on remote controlled hosts while the network traffic of the user equipment is recorded. As a final step, the framework can build high-speed multiplexed traces from the recorded data which mimic the behavior of real traffic. The characteristics of the constructed traffic compared to real world traffic measurements are also evaluated in the paper showing that the framework is able to generate realistic traffic traces that are both suitable for DPI testing and can be publicly distributed without any privacy concerns. The proof of concept implementation of the presented system is open to the public [1].

1. Introduction

In-depth understanding of the Internet traffic profile is a challenging task for researchers and a mandatory requirement for most Internet Service Providers (ISP). To this end, traffic identification helps ISPs in the quest for profiling network applications. With this information in hand, ISPs may then apply different charging policies, traffic shaping, and offer different Quality of Service (QoS) guarantees to selected users or applications.

Deep Packet Inspection (DPI) is a subclass of traffic identification where the method relies on the inspection of packet payload content, instead of only looking at the structured information found in packet headers. Packet payloads are matched against a signature database which contains unique expressions for the different protocols. This method is proven to be the most reliable one among the traffic classification procedures [2,3] and often used as a ground truth for testing other classification methods [4,5]. However, testing DPI tools in terms of both accuracy and performance is still an open issue in the research community.

Many recent publications compare the output of DPI tools in term of accuracy [4–7]. The common method in these studies is the manual creation of ground-truth data by either one of the following two ways: (i) run specific applications one at a time and filter out any background traffic that is unrelated to the given application [5], or (ii) use a third party tool that can associate every generated packet to an

* Corresponding author. Tel.: +3614633110.
E-mail addresses: megyesi@tmit.bme.hu, peter@megyesi.hu (P. Megyesi), geza.szabo@ericsson.com (G. Szabó), molnar@tmit.bme.hu (S. Molnár).

http://dx.doi.org/10.1016/j.comnet.2015.09.026
1389-1286/

application [4,6,7]. Although these methods proven to be an efficient way of testing the accuracy of DPI tools, the manual generation of test data is a highly unscalable process since it should be repeated frequently in order to test that the application-signature database of these tools are still up-to-date.

It is also well known that increasing accuracy by adding more and more signatures to the application-signature database negatively affects performance. The goal of the developers of DPI products is to provide high enough accuracy in real world networks with the highest performance. The most common solution for DPI performance testing is to use traffic simulators which mimic several application level network protocols (e.g., HTTP, SMTP), transport layer network protocols (e.g., TCP/IP, UDP/IP), and also user behavior (e.g., Poisson arrivals of user interaction events). However, simulators are not flexible enough by definition. They can simulate such traffic which is encoded in them. To create realistic traffic with a simulator, the simulator has to be also updated to keep up with the everchanging Internet [8]. This whole process is an overhead for the DPI signature set development which can be saved by collecting measurements with real protocol conversations in real network environment and replayed to the DPI box later. On the other hand, the network data is the property of the operator and plenty of privacy issues may arise if a DPI product vendor takes the measurements to its own site to further develop the DPI signature set.

The lifecycle of a DPI system comprises the steps shown in Fig. 1. After testing a DPI device, the traffic falls into two categories: the recognized traffic part for which the system provides matching signatures and the unrecognized traffic part for which no signature provided any hit. Lack of continuous update of the signatures results in decreased number of signature hits and increased number of non-hits. This effect is due to the inevitable changes in existing protocols and the approach of new ones. Up-to-date active measurements containing latest traffic patterns are needed to update the signature set of the DPI box. The effects of the updates should be tested with traffic mixes containing hints for the new signatures while mimicking realistic network environment at the same time. Our goal is to create a system capable of generating traffic traces for testing purposes of traffic classification systems (especially for DPI tools) both in terms of accuracy and performance. The scope of this framework does not include the process of signature update which is addressed in several papers, e.g., [9] and [10].

In order to reach these goals the following requirements have to be fulfilled by our system:

- The traffic generation has to be automatic to the highest level of extent.
- The generated traffic has to contain up-to-date application level protocol information in the packet payloads similar what can be measured in operational networks.
- The traffic characteristics of the generated traffic (e.g., bandwidth, payload sizes, packet inter-arrival times) have to be similar to what are measured in operational networks.
- The users in the generated traffic (e.g., parallel number of users, used applications and the way they are used) have to be similar what are measured in operational networks.
- The generated traffic should be distributable among DPI testing institutes thus it must not contain user sensitive data.
- The communicating applications of the generated traffic have to be known per packet basis providing the ground truth data.

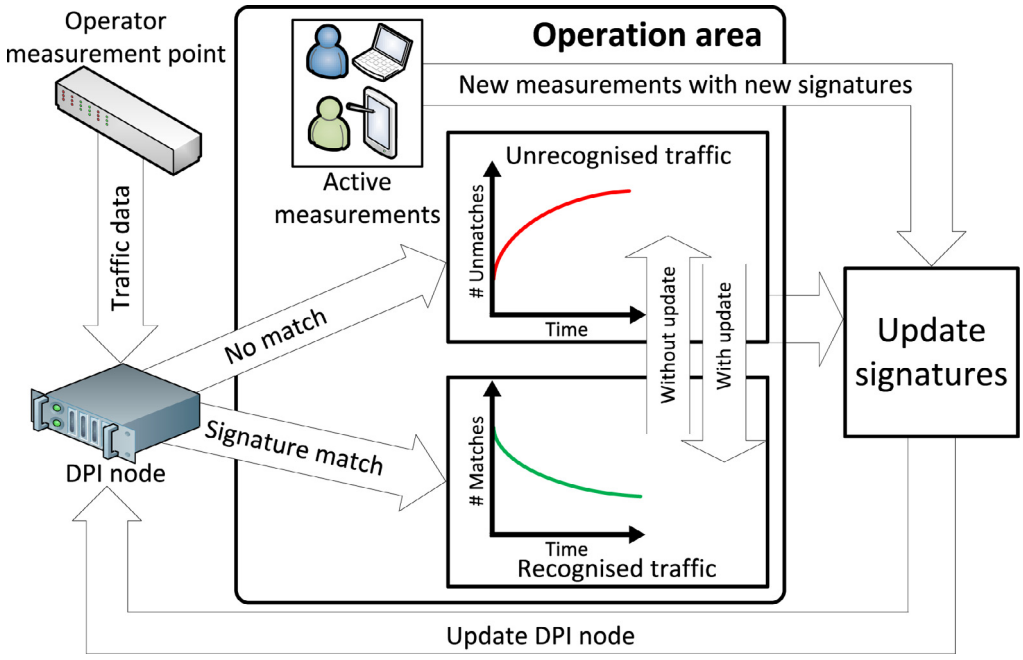


Fig. 1. The lifecycle of DPI systems.

In this paper we propose to record typical user interactions with several applications on the Graphical User Interface (GUI) and construct application specific usage models which can be used later to emulate user interactions on remote controlled computers. From real network measurements we extract typical user scenarios: used applications and their share, usage patterns, etc. With these information elements, anytime when an up-to-date validation trace has to be generated the user actions (e.g., mouse or keyboard events) are replayed according to the emulated user scenarios. The network traffic of the client machine is recorded and stored according to the emulated scenario. Finally, the user base is multiplied and an aggregated traffic is constructed from the recorded network traffic and the real world traffic models. The generated traffic has realistic payload and traffic characteristics both in inter-packet and user level timescales. Furthermore, it does not contain user sensitive data and can be distributed for wide audience. Thus, the presented system can be considered as a network measurement anonymization system to a certain extent as well.

The main contribution of the paper is to thoroughly present our framework for realistic traffic generation in which we imply both realistic payload and inter-packet timing information. Moreover, this is the first time that we evaluate our framework by comparing the characteristics of the generated traffic to traffic characteristics of real measurements recorded in an operational broadband network. We also emphasize that our system is publicly available as a proof of concept [1] to encourage other research parties to contribute to our work.

The paper is structured as follows. In Section 2 a brief overview is given about state-of-the-art traffic generation techniques along with the discussion of their inability to produce realistic output for DPI testing. The basic concept of the User Behavior Based Traffic Emulator (UBE) is presented in Section 3. In Section 4 some highlights in connection with the working mechanism of the presented system is discussed. In Section 5 we present a validation study demonstrating that UBE can construct application mixes in aggregated level similar to those found in real measurements. Finally, Section 6 concludes the paper.

2. Related work

Numerous different traffic generators were proposed in the literature in the last two decades. In this section we mention several generally known solutions which are frequently referred in papers in the field of synthetic traffic generation.

Packet-level generators are usually used for stress testing firewalls and servers or for end-to-end performance testing. The most commonly user-space traffic generator is Iperf [11] which can generate UDP packets at a given rate or TCP packets at maximum speed. BRUTE [12] was later introduced as a kernel-level application for increasing the accuracy of the output speed rate. The same idea has been implemented for specific hardware platform (Intel IXP2400) for archiving further precision and even higher maximum output rate [13]. Other solutions, such as TG [14] or MGEN [15] supports different statistical distributions to be set up for the Inter Packet Times (IPT) and Packet Sizes (PS) information. Furthermore, Ostinato [16] is a very recent generator where users can set up different streams with distinct properties and the output traffic will be the aggregate of them. Since all these solutions generate packets with dummy or random payload they cannot be used for DPI testing purposes.

Replay engines aims to reinject packets to the network as they were previously recorded with as accurate timing as possible. The most common tool for this purpose is Tcpreply [17] which is a user-space application for replying *libpcap* files at arbitrary speeds onto the network. The software package also includes Tcplivereplay which is able to replay stored traffic using new TCP connections and by that adopting for the present network conditions. TCPivo [18] is a kernel-level application for traffic reply which aims to enhance the accuracy of the timing of packets critically when replaying high speed traces (e.g., recorded on OC-48 speed). Another interesting solution is presented in [19] were authors replay OC-48 traces using multiple commodity PCs with Gigabit Ethernet network card. The collective drawback of these generators is that the measurement contains user sensitive information and cannot be distributed to other research groups for further work.

More sophisticated traffic generators are able to mimic the behavior of previously recorded traces by more complex traffic modeling. Harpoon [20] is a flow-based traffic generator that can mimic *netflow* based measurements by analyzing various flow characteristics. Swing [21] is a closed-loop, network responsive traffic generator which is able to extract distributions for user, application, and network behavior of real measurements. Tmix [22] is a traffic emulator for ns-2 based on source-level characterization of TCP connections. Although all these solutions can mimic the behavior of real network traffic in aspect of many different metrics, all these approaches miss to provide realistic packet payloads thus cannot be used as input for DPI devices.

D-ITG [23] is a comprehensive framework for synthetic workload generation. The tool supports both model-based and trace-based traffic generation at the same time. The model-based mode uses Hidden Markov Model approach for modeling the IPT and PS sequence, while the trace-based mode can send packets according to the time order of a previously recorded capture file. The same two problems are present in D-ITG for DPI testing as in the previous cases: the model-based mode generates packet with synthetic payload and the trace-based mode arises privacy issues.

The idea of using GUI testing tools for controlling application in place of a human user was proposed in [24] where authors present a finite state machine model for driving applications. However, their automation only covers basic applications (e.g., Internet Explorer, Outlook and Microsoft Word) using an isolated testbed instead of the Internet and their goal is to present the effect of using anti-virus software on the system's performance. Our goal is to provide repeatable traffic generation in more versatile environments including measurements with various access technologies and smartphone platform as well.

Our framework does not belong to any of these categories since on the one hand, it captures the behavior of real users, and on the other hand, it generates traffic composed of traffic taken from real measurements. It is also able to generate new user level measurement automatically ensuring that our database continuously contains the newest traffic patterns

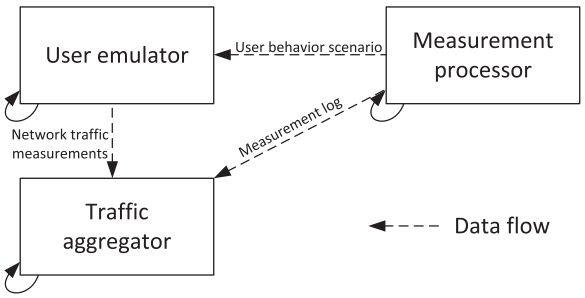


Fig. 2. Abstract structure of the User Behavior Based Traffic Emulator.

of different applications providing the ground truth data as well.

3. The concept of the system

During the maintenance of a DPI box the protocol signature set has to be revised from time to time to check whether some of the signatures become completely obsolete or a new traffic type has emerged and the signature set has to be extended. The unrecognized traffic, i.e., the traffic which has no signature yet does not necessarily originate from a completely new application but a new version of an existing popular one extended with new features. The extension process of the signature set usually starts with active measurements. Selected applications are used one-by-one and regular expressions are constructed [9] on the recorded traffic. After software updates, the active measurements have to be redone. The measurements require the same user interactions with the application GUI from time to time. The basic idea of our system originates from the recognition that the manual repetitive work can be substituted with an automatic mechanism which is feasible due to the practice that the GUI look and feel change less frequently than the underlying network protocol. A good example is Skype [25] which has the same skin from version 1–3 and it changed radically only in version 4. On the other hand, the underlying network protocol changed in several subversions.

For terminology clarification the term *user behavior scenario* indicate a series of actions that a user does to interact with GUI applications. For example, the user opens a web browser, navigates to a torrent site, downloads a torrent file, opens it in a torrent application and five minutes later he or she closes it. Whereas, an *emulated user behavior scenario* means the process constructed by our framework in order to mimic a specific *user behavior scenario*.

In order to clearly present the architecture of the User Behavior Based Traffic Emulator (UBE) we present and discuss three figures in different levels of details:

- Fig. 2: High-level abstract structure of the framework.
- Fig. 3: Detailed functions of the framework.
- Fig. 4: Data flow and database structure of the framework.

Fig. 2 presents the abstract structure of the UBE. The framework is composed of following three main components. The *Measurement processor* is responsible for the definition of typical user behavior scenarios. The *User emulator* can emulate a user behavior scenario on a remote controlled machine and record the traffic generated during the process. The *Traffic aggregator* is able to merge multiple traffic measurements in order to create a high speed aggregated traffic mix. The abstract structure shown in Fig. 2 is detailed in Fig. 3.

Although initial set up of the framework requires some operations detailed below, the main function of the three components in Fig. 2 are repeatable and parallel. For further clarification, Fig. 4 presents the data flow details and Fig. 5 the time flow sequence diagram of operations, respectively. The detailed functions of the three main components of UBE are the following.

3.1. Measurement processor

In the *Measurement processor* the recording of the two necessary inputs are performed:

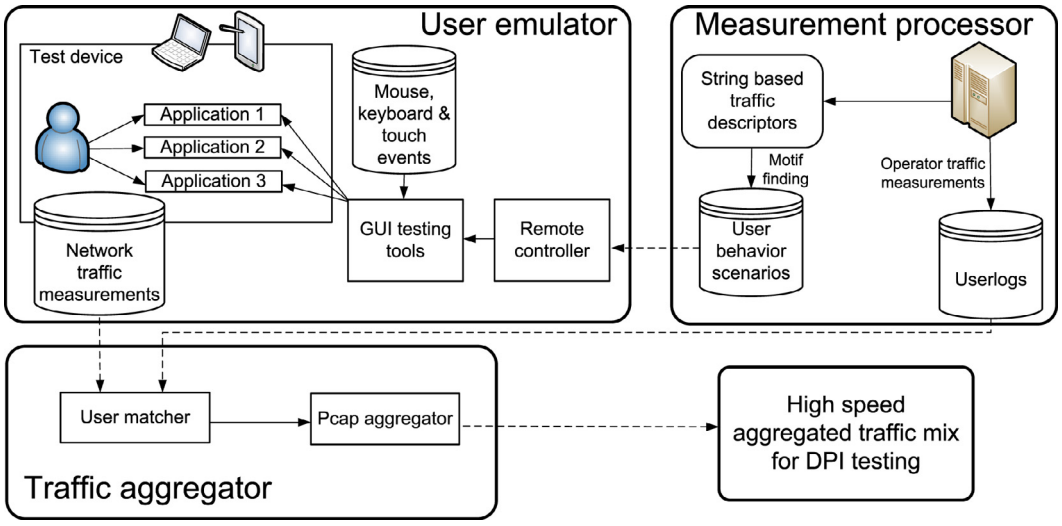


Fig. 3. The architecture of the User Behavior Based Traffic Emulator.

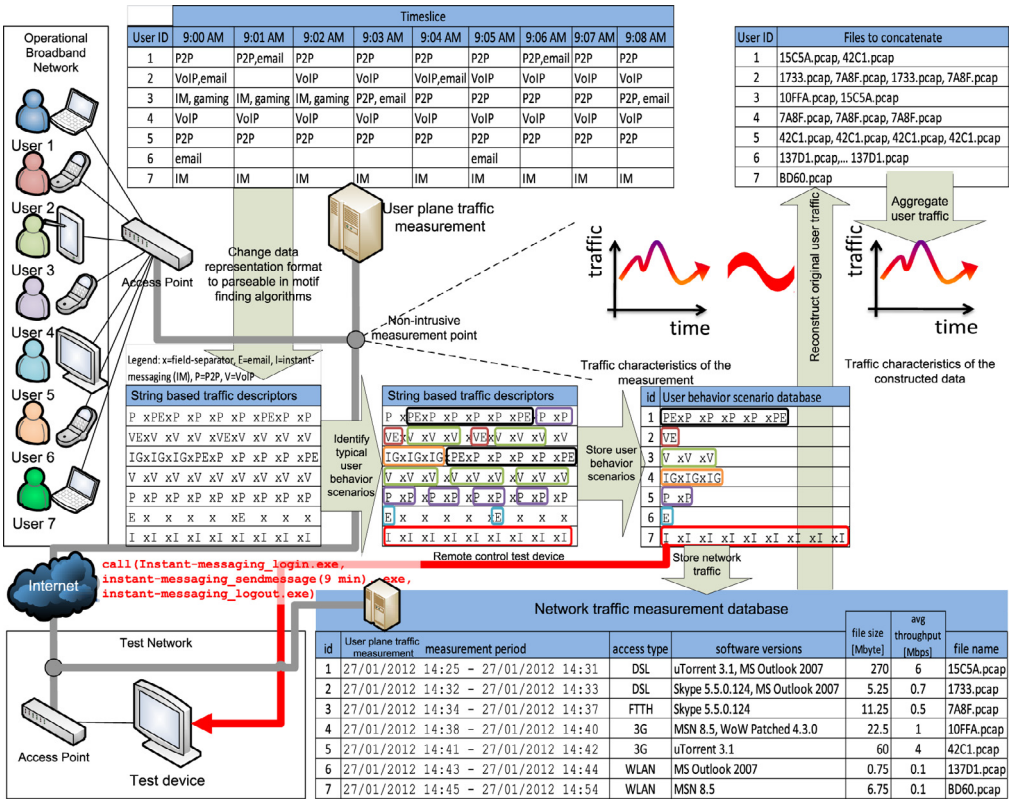


Fig. 4. Data flows in the User Behavior Based Traffic Emulator.

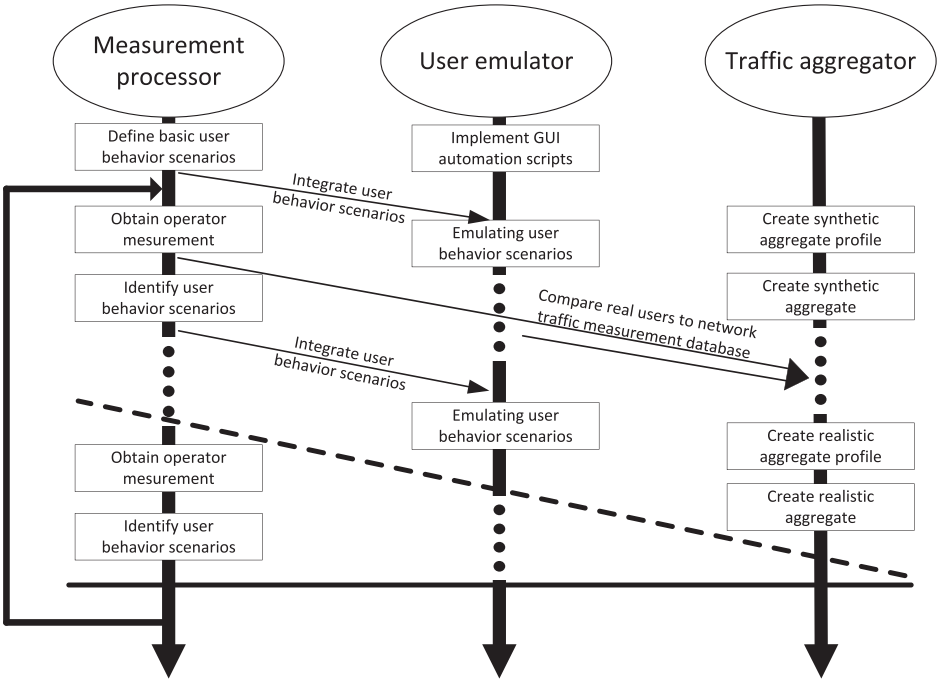


Fig. 5. Time flow sequence diagram of the User Behavior Based Traffic Emulator.

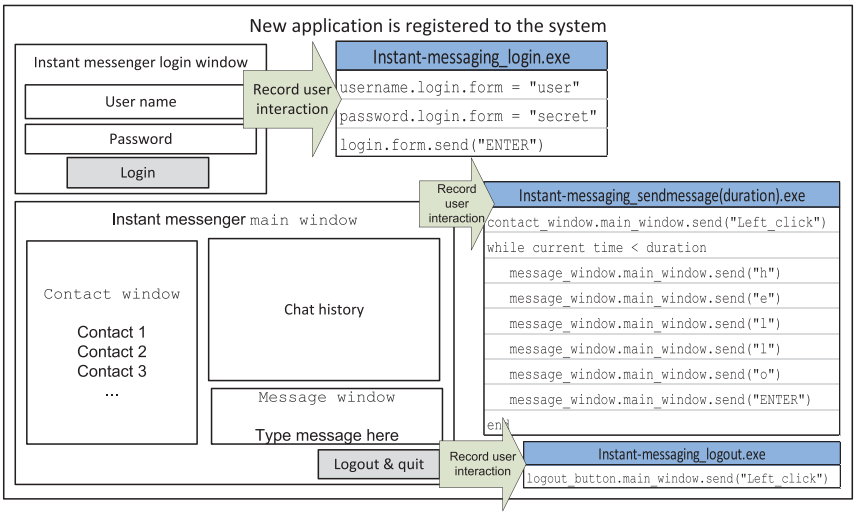


Fig. 6. New application is registered to the framework.

- *Recording of user interactions*: When a new application is added to the system – or one of the GUIs of the applications has changed significantly –, a user will simply use the application while its interaction with the GUI is recorded. This process typically means the naming of the input fields, buttons, etc. – not the exact location of the mouse cursor – as Fig. 6 shows. Object names are rarely changed in a specific application thus this step is robust to version changes. The recorded typical sessions are stored in specific scripts on the test devices.
- *Traffic measurements are taken in operational broadband networks* and typical user behavior scenarios are extracted and stored in a database (see Figs. 3 and 4 from *Operator traffic measurement* to *User behavior scenarios database*, for further details see Section 4.1). User behaviors scenarios can also be defined manually. For example, one can integrate a simple scenario of five minutes of web browsing with P2P at the background via UBE's web interface, and the framework automatically records it to the *User behavior scenario database* by assigning a remote control procedure for this activity.

3.2. User emulator

In the *User emulator* the creation of traffic segments are performed. When new up-to-date validation traffic is needed, the information from the *User behavior scenario database* (see Fig. 3) is grabbed and user actions are emulated by remote controlling computers (see Section 4.3 for details) with the recorded user interactions (see Fig. 6). During remote controlling (see Fig. 3 *Remote controller*), GUI testing tools drive the applications on the client machine and make them to generate real traffic on the network. The generated traffic is recorded and stored in the *Network traffic measurement database*. Note that the scenarios can include such cases when the effects of the applications on each other are emulated, e.g., web browsing with streaming radio and background P2P traffic to consider the effects of the applications on each other's traffic in the transport layer. We installed several test machines on different access network types for further increasing the diversity of the *Network traffic measurement database*. The database can also store the version number of the used clients and later validation traffic for a specific snapshot in the past can be constructed.

3.3. Traffic aggregator

In the *Traffic aggregator* the aggregation of the traffic segments is performed. The number of users is increased and an aggregated traffic is created based on the original traffic measurement and the *Network traffic measurement database* (see Fig. 4 *Reconstruct original user traffic*). The reconstruction of per user traffic implies the arrangement of the proper measurement segments of the *Network traffic measurement database* according to the order defined during the identification of typical user behavior scenarios. As the operational traffic measurement and the measurements in the *Network traffic measurement database* have different measurement periods the packet timestamps have to be modified according to the activity period of the specific user in the user plane traffic measurement. Finally, per user traffic can be aggregated according to the timestamps.

4. Highlights of the implementation

In this section we would like to give some insights on the implementation of the framework. We do not go into the very details in order not to spin out this section but put emphasize on the most interesting elements. (For further implementation details see the help section of the portal [1].) The framework is continuously developed thus it is not limited to the applications or operation systems included in the current paper. Moreover, we mention that previous implementation of our framework had a demo presentation in [26] and the applicability of our system for traffic analysis purposes was shown in [27].

4.1. Creation of user behavior scenarios

In our framework we defined the granularity of user behavior analysis in 1 min scale. This means that we can say for each 1 min period of the user what applications were used, e.g., only email occurred or also P2P file-sharing existed in the background (see Fig. 4 *User plane traffic measurement*).

There are two possible ways to recreate the typical user behavior scenarios. One is a bottom-up approach, when we analyze and store small time slices of the user activity in our *user behavior scenario database*. Later when the original user activity has to be reconstructed there are a limited number of small building blocks in the database and, for example, a 100 min long user activity is constructed from 100 pieces of 1 min long slices. The other is a top-down approach where longest possible building blocks are matched one-by-one for the user activity with decreasing matching length, for example, a 100 min long user activity is constructed from 4 pieces of 20 min long slices, 1 piece of 10 min slice and 2 pieces of 5 min slices. Our goal was to focus on the top-down approach as it reduces the effect of transient states recorded during the communication of applications. An example for such a transient state occurs during P2P file-sharing. At the beginning of the P2P file-sharing session when good seeders are searched for the network traffic has mainly signaling traffic exchange, while later this ratio turns in favor of the content exchange.

To extract typical user behavior scenarios our idea was to utilize algorithms which search for high number of occurrences with tunable soft-limit for hits and non-hits. Such algorithm was applied in [9]. In that scenario the original goal of the algorithm was to find the smallest set of signatures for the biggest coverage ratio for a specific application. Our current goal is to select the smallest set of user behavior segments for the full coverage of the total user behavior sequences. To achieve this we constructed string literals per user from the packet-level network traffic measurement. In the *String based traffic descriptors* in Fig. 4 the used applications are represented with a character while the 1 min granularity is signaled with delimiter characters (‘x’). For example, *PxPWxPEx* describes a three minutes long user scenario where P2P traffic was continuous, web-browsing was occurred in the second minute and e-mail in the third (see [28] for further details on these string based traffic descriptors).

Currently there are 749 entries in the User Behavior Scenario Database which were created after analyzing multiple real measurements. The minimum and the maximum length of these scenarios are 4 and 10 min, respectively. Since the emulation is a real time process running all these scenarios takes about one week measurement on one test machine. Furthermore, we found that using the current entries we can sufficiently cover the available real measurements.

4.2. Types of user traffic

Two main types of generated traffic are identified in our framework. One requires the active attention of the user, thus the generating application is in the user's focus meaning that the specific application is the focused window. The background activities are usually started once and later - after several other actions performed by the user - are switched off. The performed actions of the user behavior emulation consist of three main phases:

- *Starting phase:* This phase usually includes the starting of the application client or navigation to the starting page and login with user credentials. User credentials used by the framework were created solely for testing purposes. 3*rd* party testers can change these information to their own and use them to build a database. However, these changes would not have significant impact on the payloads since credentials usually transferred via encrypted channels.
- *User activity or active phase:* In this phase some user actions are performed, e.g., sending some hotkey actions, mouse actions or other keyboard events.
- *Ending phase:* This phase is responsible of the proper logout and closing of the application.

The two activities are discussed as follows:

4.2.1. User focus is required

In this section we enumerate the implementation tricks of the user behavior emulation of those applications types that require user focus.

- *Gaming*: We use World of Warcraft [29] to generate gaming traffic. The start phase opens the application and enters into the Public Test Realm (PTR) which is a special server used for testing the upcoming patches. Although PTR is not a regular server many players use it to test the upcoming changes in game mechanics. The active part of the emulation performs randomly some movements, spell usages or chatting. The ending phase closes the application. In the future, we would like to extend the gaming scripts to other popular on-line games and also compare the emulation results to relevant studies in this filed such as [30,31].
- *Instant messaging*: We use Skype [25] for traffic generation (former version also included MSN Messenger [32] which has been integrated into Skype). The start phase opens the given application and performs the login of the user. The active phase picks a contact and starts sending messages to it. The messages are typed and sent with the exact timing we measured according to formerly recorded chat logs. The emulation of typing is important due to the working mechanism of instant messaging applications which notifies the parties whether the other communicating party types or erases something. The ending phase logs out and closes the application.
- *Remote access*: In our framework we use two kind of popular remote access applications: the built in Remote Desktop Connection [33] of MS Windows and RealVNC [34]. The starting phase establishes the communication tunnel. The user activity emulation phase performs some simple mouse and keyboard actions, while the ending phase terminates the connection.
- *Social-networking*: To be able to generate social-networking traffic we created a user on Facebook and 'liked' several pages to make the 'wall' full of new comments from time-to-time. The starting phase opens the website and navigates to random links inside the Facebook for the given time. We also switched of the

caching function of the browser to download every data every time the script opens the same link. The script is also able to send messages to a randomly chosen friend according to the same log files we use during instant messaging.

- *Voice over IP*: Note that in some cases the synchronization of two clients is necessary. We need two remote controlled computer for this type of activity and have Skype [25] installed on them. One of them will be the call initiator, the other is the receiver. The call initiator picks the receiver computer user id from the contact list and performs a call with it. The receiver will automatically accept every incoming call. We play audio files containing human communication as input for both the caller and the receiver.
- *Web browsing*: To emulate web browsing activity a link is picked randomly from popular web pages [35] of the specific country the remote controlled computer is situated and a browser is opened with this URL. After loading the page, the active phase waits for a given time, browses the page for a while by rolling down on it and navigates to another either to a randomly chosen link on the current page or a randomly chosen URL from the original pool. The browsing phase can be important in case of AJAX [36] based dynamic web pages in which the separate parts of the page are downloaded on demand, e.g., on the eBay [37] site.

4.2.2. Background activities

In this section we enumerate the implementation tricks of the user behavior emulation of those applications which run in the background and do not require user focus.

- *File download*: To emulate file downloading traffic the starting phase begins file downloading by picking a random file from a formerly defined pool. After finishing, another one is picked and download is started. The ending phase stops the download and deletes every data from the download directory.
- *File sharing*: During file sharing emulation the file sharing client randomly opens a torrent file from a formerly defined pool. The pool contains torrent files in various sizes from different torrent sites. The pool also contains some magnet torrent files which do not use a centralized tracker server but rather other torrent hosts to find the given file to download (e.g., The Pirate Bay now shares only magnet links rather than regular torrent files). The ending phase deletes the downloaded data, thus reopening the torrent file results in restarting the whole file-transfer. Currently we can emulate file sharing using uTorrent, Vuze and BitCommet.
- *Video playback*: Online video playback is either active being in the focus of the user, jumping in the video stream, clicking on new recommended videos, etc. or a completely background activity which plays all videos in a track list. Our framework emulates the later scenarios utilizing the channel function of YouTube. A playlist is loaded first and each of the videos are played one-by-one in the list.
- *Malicious traffic*: DPI devices could also be used for detecting malicious traffic. In order to further extend the functions of UBE we defined malicious traffic as a separate traffic type. We implemented two scripts which are able to download various malicious traffic in the background. The first script downloads the popular Eicar standard anti-virus test file [38] which is a harmless executable but most anti-virus product reacts as it were a virus. The second script uses the Malware Domain List database [39] to download a random malicious file. Since this list is created for security experts, most of the links contains a real harmful program thus we only implemented it to virtual test machines where backup images are available.

4.3. Remote controlling of the GUI on desktop Windows platform

For the emulation of user behavior we used AutoIt [40]. Its primarily goal is to make possible to create automation scripts or macros for Microsoft Windows programs. For every specific application client and for each phase (see Section 4.2) a specific script is constructed. The automation script can be compiled into a compressed and standalone executable file which can be run on computers that do not have the AutoIt interpreter installed. Moreover, AutoIt is compatible with every version of Windows from XP to 8.1 without recompiling the executable files. Also, previous implementations of the automation scripts used AutoHotkey [41] and Watir [42] but we found AutoIt much more flexible. Currently we are also experimenting with Sikuli IDE [43] which uses image recognition to identify and control GUI components. This technique could be useful in case of a version update with radical changes in an application's GUI since updating the automation script would only require replacing some screenshot files. Furthermore, Sikuli IDE works on Windows, Linux and Mac OS systems as well. Adding a new application to the framework (or updating an existing one to recent GUI changes) is fairly simple since both AutoIt and Sikuli IDE are easy-to-use tools, thus an experienced programmer should do it within a day.

The standalone executable files has to be executed in a specific order according to the user behavior scenario we intend to emulate (see Section 4.1). Applications with GUI, e.g., uTorrent [44] or Skype [25] have to be bounded to a graphical session in the Windows system otherwise running them directly from a console session would cause them unexpected errors. Thus the execution is performed from console but via an application called PsExec [45].

PsExec is invoked automatically from an external server by logging into the Windows machine via Telnet. Telnet session can be managed efficiently from the main server containing the user behavior database via Expect [46]. Expect is a simple script language created for automating interactive console based applications such as *telnet* or FTP.

4.4. Remote controlling of the GUI on Android platform

We are also able to emulate user behavior scenarios on Android platform using MonkeyRunner [47]. MonkeyRunner is part of the software development kit of Android and it is commonly used for stress testing applications as it can generate touch, drag and keystroke events on the smartphones GUI. Although, MonkeyRunner only supports touch and drag

events on exact pixels (rather than control buttons), by using the *intent mechanism* we were able to implement most of the emulation scripts for Android platform. *Intents* are abstract descriptions of an operation to be performed. It can be directly sent to an application or broadcasted into the Android system. In the latter case the *global intent filter* will determine which application should get the message [48]. For example, sending an *intent message* with the Uniform Resource Identifier (URI) www.google.com will be directed to the default web browser or the URI *skype:testuser* will automatically open Skype and call the user named *testuser*. We have used these mechanisms to demonstrate the differences in the traffic pattern if the same user behavior scenario is emulated on different access and OS types [27].

The possibility to emulate users on multiple OS platforms using different access types with many applications gives us the opportunity to characterize our measurements in a similar way that was presented in [49]. For example, one could identify mobile users in the operational measurement and only compare them to dump files in our measurement database that was emulated on mobile platform. In [8] authors describe why it would be very complicated (if not impossible) to build a traffic simulator that can cover every possible network scenario. This is the reason why our approach needs a real measurement to mimic its behavior. This way an operator could contract traces similar to the conditions on his/her network.

4.5. Recording of network traffic

For one specific usage scenario multiple measurements are created and stored in the *user behavior scenarios database* (see Fig. 4 *Network traffic measurement database*) on the different test machines and setups. This is practically a link to a network measurement file recorded with *tcpdump* [50] during the emulation of the user behavior scenario. It is important to note that the Windows based traffic generating machines have a special driver (see [6]) installed to create dump files which can be perfectly classified later. This is achieved by a daemon which can track the opened sockets and modify the IP header according the application generated the current packet. Also, in Android platform we use similar approach that can track the opened socket and log it on the device (thus in this platform we do not modify the IP packet headers). This tool is available on Google Play Store [51]. These mechanism provides the ground truth data on per packet granularity for every measurement in the *Network traffic measurement database* thus fulfilling this requirement against our framework. However, we also remark the tool we currently using for ground-truth generation on Windows platform is not open for the public. In order to make our system more open, in the future we would like to to replace this mechanism with publicly available ground-truth generation tools such as GT [7] or VBR [52].

4.6. Guide to deploy the system

In this subsection we give further insight into how one can deploy and use a similar framework. Fig. 7 depicts the main components that have to be installed and also, the main outputs that the system generate. There are two components that need to be deployed: the control server and the test machines. The control server (in our framework a simple Linux machine) handles the databases (the *user behavior scenario database* and the *network traffic measurement database*) and controls the user emulation processes on the test machines. Deploying a test machine requires three steps. First, we have to make sure that the machine is remotely controllable by the control server. For PCs it means an open telnet connection, whereas Android phones have to be attached to server via USB cable. Second, we have to place the GUI control scripts to the test machine which are able to drive the applications. And finally, test machines have to be able to record the network traffic in *libpcap* format. We use *tcpdump* on Linux based

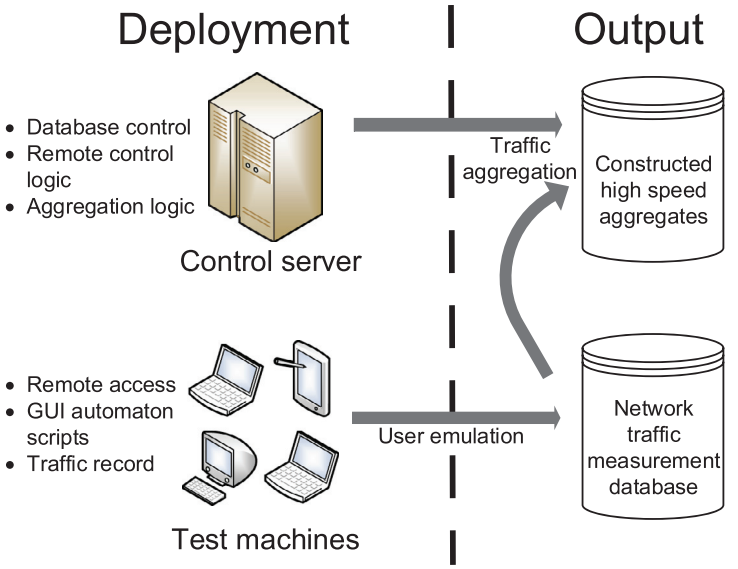


Fig. 7. Main components to deploy the framework.

systems and *windump* on Windows system for this purpose. The logic of the *Traffic aggregator* is also stored in the control server. Most of the above mentioned scripts are available for download on UBE's website [1].

The user emulation processes generate the individual trace files in the *network traffic measurement database* which is one of the fundamental output of the system. The usage of these measurements is twofold. Firstly, by regular updates of these measurements the database can contain the newest application signatures thus it could serve as an input for automatic signature update tool (such system was presented in [9]). On the other hand, the *network traffic measurement database* is also the input for the *Traffic aggregator* part of the system where it is able to generate high speed aggregated traffic mixes which could be used in performance testing of DPI tools. An example for generating such high speed aggregate is presented in the next section.

5. Validation study

We have carried out a performance evaluation study of the User Behavior Based Traffic Emulator to validate that the emulated traffic reflects similar characteristics compared to the traffic generated by users in real measurements. The validation of traffic generators can usually be performed from different points of views and on different time-scales [53]. In this section we summarize our results focusing on four metrics as representative validation metrics from these important traffic characterization dimensions:

- *Traffic components* characterization: traffic shares of applications in the aggregation.
- *Packet-level* characterization: traffic intensity and packet size distribution.
- *Flow-level* characterization: flow size distribution and
- *Scaling-level* characterization: logscale diagram.

Our original database generated by our emulator (all dump files in the *network traffic measurement database*, henceforth *UBE database*) contains about 1800 individual dump files, a total of 165GB data, 200 millions of packets and 3.5 millions of flows. In order to investigate the traffic shares per applications in this database we classified the traffic using nDPI which is an open source Deep Packet Inspection application developed by the nTOP project [54] and the results are presented in Table 1. Also, nDPI is considered to be one of the best performing DPI tools in the literature and it is also frequently upgraded by the developers [4,5]. We used this methodology since later in this section we will show that the real measurement trace and the constructed trace by UBE generate similar amount of application signature matches using nDPI. To know the accuracy of nDPI is of great importance and our future work includes the discussion of this topic. We have preliminary result on accuracy comparison of different traffic classification tools based on our labeling technique [55] but this topic needs a more sound analysis so we consider it as one of our future research topics.

A real operational network measurement (*BME WiFi trace*) was taken at our university campus in a 10Gigabit Ethernet link which aggregates the traffic of WiFi users of two buildings. This measurement is a six minutes long trace containing about 4GB data and 5.5 million packets including the traffic aggregation of about 1970 users, 125k flows and 40 known applications.

We created the *constructed trace* via the *Traffic aggregator* component of UBE using the available individual dump files in the *UBE database* as follows. Firstly, we consider the user level log from the *BME WiFi trace* used by the nDPI classificatory. This log contains the amount of data that were generated by every individual users in the aggregated measurement in a per application basis. After, we find out which individual dump file from the *UBE database* is the most similar to a given user (see Fig. 3 *User matcher*). The most similar measurement file is calculated by the following distance formula which can be considered as a metric in the application space: $\sqrt{\sum_i (O_{app_i} - P_{app_i})^2}$, where O_{app_i} and P_{app_i} is the amount of the i^{th} application data in bytes that were generated by the specific user in the operational measurement and in the specific dump file in our database, respectively. After this step, we had a list of dump files that should be concatenated to get a similar mix to the original operational measurement (see Fig. 4 *Files to concatenate*). To get the final aggregated traffic we performed the reconstruction phase for every user existing in the trace. (For further details about the algorithm refer to [55].) The main packet modifications are the following:

- Adjusting the timestamp of the packets from the measurement date to the date when the user was active. This is a fix shift and the inter-packet timers are not altered.
- Managing the IP addresses in the function of the number of emulated users. We have to alter the IP addresses of the test devices in the IP header. The framework is also

Table 1
Traffic classification results for the *UBE database.*

	Application	# Bytes	% Bytes	# Packets	% Packets	# Flows	% Flows
1	QuickTime	45 G	26.8	44 M	21.7 %	1016	0.03
2	Unknown	42 G	25.4	60 M	29.3 %	787 k	22.7
3	Flash	40 G	24.2	42 k	20.6 %	16 k	0.5
4	Bittorent	23 G	13.9	33 M	16.3 %	1.98 M	57.2
5	HTTP	13 G	7.8	19 M	9.1 %	376 k	10.8
6	SSL	525 M	0.3	830 k	0.4 %	26 k	0.75
7	DNS	55 M	0.03	340 k	0.16 %	163 k	4.7
8	Skype	50 M	0.03	220 k	0.1 %	22 k	0.6
9	Google	24 M	0.01	46 k	0.02 %	1170	0.03
10	ICMP	22 M	0.01	190 k	0.1 %	76 k	2.2
	SUM	167 G		206 M		3.5 M	

Table 2
Traffic classification results comparing the *BME WiFi* trace to the *constructed trace*.

Application	Bytes		Packets		Flows	
	BME	Constr.	BME	Constr.	BME	Constr.
Unknown	2 G	1.95 G	2.9 M	2.8 M	36.3 k	43 k
HTTP	1.17 G	1.12 G	1.2 M	1.4 M	11.8 k	22 k
QuickTime	359 M	250 M	310 k	233 k	162	47
Bittorent	256 M	198 M	575 k	573 k	46 k	115 k
SSL	167 M	138 M	277 k	270 k	5186	9385
Google	42 M	3.7 M	64 k	9200	1025	95
Flash	23 M	94 M	24 k	97 k	97	431
DNS	7.7 M	4.9 M	42 k	27 k	20 k	13 k
Skype	1.9 M	4.1 M	15 k	39 k	1560	6600
ICMP	0.6 M	1.4 M	5441	11 k	2164	6170
SUM	4 G	3.77 G	5.45 M	5.45 M	125 k	217 k

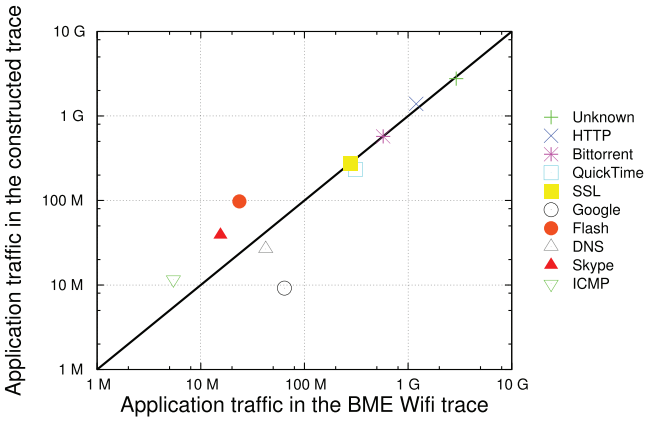


Fig. 8. Application mix of the *BME WiFi* and *constructed traces*. The x-axis represents the traffic volumes of the top 10 applications in the *BME WiFi trace* whereas the y-axis represents the same traffic volumes in the *constructed trace*.

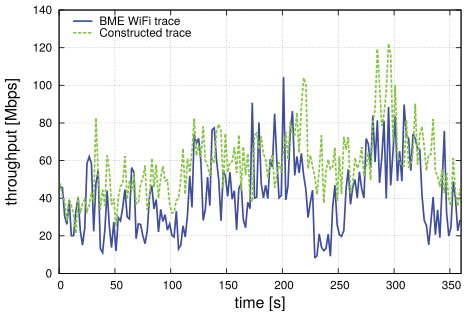
capable of searching the payload of the packet for the IP address in both binary and text format and switches them for the given address. The checksums of the IP/TCP headers are also recalculated.

The *constructed trace* contains about 450 individual dump files, a total 3.8 GB data, 5.4 millions of packets and 217 thousands of flows and Table 2 presents the result of the top 10 applications after the classification for both the *constructed trace* and the *BME WiFi trace*. In addition, Fig. 8 shows a general view about these results where we plotted the traffic volume of the top 10 applications in the *BME WiFi trace* vs. the traffic volume in the *constructed trace*. Furthermore, the names of the applications are analogous to name conversation of nDPI.

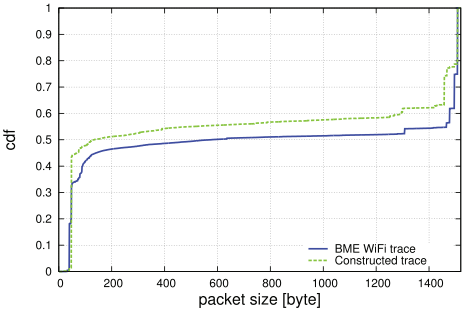
It can be seen from Table 2 and in Fig. 8 that the traffic shares of the top applications in the aggregation are correctly represented and important characteristics are also captured, e.g., Bittorent is the dominant protocol in terms of flows, a few QuickTime flows cause reasonably large amount of traffic, or fairly large number of DNS and ICMP flows cause very small amount of traffic. On the other hand, traffic associated to Google web services is slightly over-represented in the *BME WiFi Trace* than in the constructed aggregate possibly due to the fact that many Android smartphone uses the campus WiFi network. However, this is a good indication that the *constructed trace* is suitable for performance testing of DPI tools since our first goal was to create a traffic mix which can generate similar amount of signature matches than the original trace would.

In order to compare traffic characteristics at packet-level the traffic intensity to downstream direction and the packet size distribution were investigated in Fig. 9a and b, respectively. Although the throughput in the *BME WiFi* and *constructed traces* are not matching, the trends of the two curves in Fig. 9a show similar characteristics. This is further strengthen later in this section by a wavelet scaling analysis. In Fig. 9b the Cumulative Distribution Function (CDF) of packet size shows a good fit between the two curves. The shift between the two curves can be explained by a slight over-representation of small sized of packets in the *constructed trace* (the total traffic of the *constructed trace* is about 5% less than the traffic in the *BME WiFi* trace, whereas the total number of packets is about the same in the two traces). Furthermore, in Table 3 we collected the mean and standard deviation values for the aforementioned curves. Although the values have some deviation to each other we consider these values sufficiently close to approximate the characteristics of the original measurement.

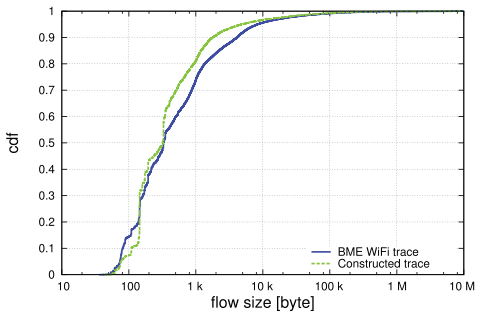
To analyze flow-level characteristics of the two traces we plotted the flow size distributions. The CDF of the flow size is also well captured as depicted in Fig. 9c. We observed an increased number of the torrent request-response activity (53 thousands of 145 byte flows) in our *constructed trace*



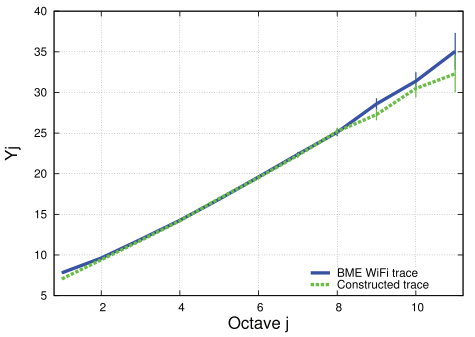
(a) Traffic intensity to downstream direction in the *BME WiFi* and *constructed traces*



(b) Packet size distributions of the *BME WiFi* and *constructed traces*



(c) Flow size distributions of the *BME WiFi* and *constructed traces*



(d) Logscale diagrams of the *BME WiFi* and *constructed traces*

Fig. 9. Comparing the *BME WiFi* and *constructed traces* using different metrics.

Table 3
Statistical significance indicators of the traffic characteristic comparison of the *BME WiFi* trace to the *constructed trace.*

Statistic	Mean		Standard deviation	
	BME	Constr.	BME	Constr.
Traffic intensity [M bps]	38.09	45.48	20.32	17.99
Packet size [byte]	771	680	693	668
Flow size [k byte]	17.4	10.2	905.9	766.2

compared to the *BME WiFi trace* resulting in a vertical jump in the CDF of the *constructed trace*. This is also the main cause for the slightly smaller values in the mean and standard deviation of flow sizes in the *constructed trace* presented in Table 3.

To investigate the scaling characteristic of the traffic we calculated the logscale diagram [56] for both the *constructed trace* and the *BME WiFi trace*. The discrete wavelet transform represents a data series X of size n at a scaling level j by a set of wavelet coefficients $d_X(j,k),\ k = 1, 2, \ldots, n_j$, where $n_j = 2^{-j}n$. Define the q^{th} *order Logscale Diagram* (q-LD) by the log-linear graph of the estimated q^{th} moment $\mu_j(q) = 1/n_j \sum_{k=1}^{n_j} |d_X(j,k)|^q$ against the octave j. Linearity of the LDs at a different moment order q suggests the scaling property of the series, i.e., $\log_2 \mu_j(q) = j\alpha(q) + c_2(q)$, where $\alpha(q)$ is the *scaling exponent* and $c_2(q)$ is a constant. The plot of $\alpha(q)$ against q can reveal the type of scaling [56].

The scaling characteristics for both the *BME WiFi trace* and the *constructed traffic* are presented by the Logscale Diagram related to the moment order $q = 2$ in Fig. 9d. A nearly linear interval of the LD plot at octaves $4 \le j \le 11$ can be observed for both traces revealing the well-known *Long-Range Dependence (LRD)* property of the aggregated traffic [56]. A linear regression to this interval gives an estimation of LRD parameter of $H_{BME\ WiFi\ trace} = 0.875$ and $H_{constructedtrace} = 0.842$ for the original measured and the emulated traffic, respectively. These results clearly indicate that the emulated traffic accurately captures the complex scaling structure of the original measured traffic.

In summary our validation study shows that the emulator is able to reproduce an aggregated traffic which captures the characteristics of the original measurements.

6. Conclusion

In this paper we introduced the User Behavior Based Traffic Emulator (UBE), an automatic traffic emulation framework for constructing database for DPI testing. The system works by recording the traffic of remotely controlled computers and aggregating the traffic segments into multi-user traffic. UBE is able to construct a realistic aggregate

traffic with arbitrary application mix, which is usually difficult to find in real measurements. Moreover, the generated traffic has no privacy restrictions so it can freely be distributed among DPI testing institutes. The emulated traffic contains up-to-date application level protocol information in the packet payloads and the characteristics of the traffic (e.g., application mix, packet sizes, flow sizes, scaling structure) exhibit the traffic characteristics measured in operational networks. The aggregated per user traffic was analyzed and validated by comparing several traffic characteristics with corresponding metrics investigated in traffic taken from real measurements.

References

[1] User Behavior Based Traffic Emulator DEMO portal, Retrieved: May, 2015. URL http://ubetest1.hsnlab.tmit.bme.hu/
[2] A. Callado, C. Kamienski, G. Szabo, B.P. Gero, J. Kelner, S. Fernandes, D. Sadok, A survey on internet traffic identification, Commun Surv Tutor, IEEE 11 (3) (2009) 37–52.
[3] A. Dainotti, A. Pescape, K. Claffy, Issues and future directions in traffic classification, Network, IEEE 26 (1) (2012) 35—40.
[4] Tomasz Bujlow, Valentin Carela-Espanol, Pere Barlet-Ros, Independent comparison of popular DPI tools for traffic classification, Comput. Netw. 76 (2015) 75–89.
[5] S. Alcock, R. Nelson, Measuring the accuracy of open-source payload-based traffic classifiers using popular internet applications, in: 2013 IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops), 2013, pp. 956–963.
[6] G. Szabó, D. Orincsay, I. Szabó, S. Malomsoky, On the validation of traffic classification algorithms, in: Proceedings of the PAM, Cleveland, Ohio, USA, 2008.
[7] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, K.C. Claffy, Gt: picking up the truth from the ground for internet traffic, SIGCOMM Comput. Commun. Rev. 39 (5) (2009) 12–18.
[8] S. Floyd, V. Paxson, Difficulties in simulating the internet, IEEE/ACM Trans. Netw. 9 (4) (2001) 392–403.
[9] G. Szabó, Z. Turányi, L. Toka, S. Molnár, A. Santos, Automatic protocol signature generation framework for deep packet inspection, in: Proceedings of the. Valuetools, Cachan, France, 2011, pp. 291–299.
[10] M. Ye, K. Xu, J. Wu, H. Po, AutoSig-automatically generating signatures for applications, in: Proceedings of the Ninth IEEE International Conference on Computer and Information Technology, 2009 (CIT '09), Kiev, Ukraine, 2009, pp. 104–109.
[11] Iperf, Retrieved: May, 2015. URL http://sourceforge.net/projects/iperf/
[12] N. Secchi R. Bonelli, S. Giordano, G. Procissi, BRUTE: a high performance and extensible traffic generator, in: Proceedings of the SPECTS '05, Philadelphia, PA, USA, 2005, pp. 839–845.
[13] G. Antichi, A. Di Pietro, D. Ficara, S. Giordano, G. Procissi, F. Vitucci, BRUNO: a high performance traffic generator for network processor, in: Proceedings of the SPECTS '08, Edinburgh, UK, 2008, pp. 526–533.
[14] TG, Retrieved:May, 2015. URL http://www.postel.org/tg/tg.html
[15] MGEN, Retrieved: May, 2015. URL http://cs.itd.nrl.navy.mil/work/mgen/
[16] Ostinato, Retrieved: May, 2015. URL http://ostinato.org/
[17] tcpreplay, Retrieved: May, 2015. URL http://tcpreplay.synfin.net/
[18] W. Feng, A. Goel, A. Bezzaz, W. Feng, J. Walpole, TCPivo: a high-performance packet replay engine, in: Proceedings of the ACM SIGCOMM Workshop on Models, methods and tools for reproducible network research, Karlsruhe, Germany, 2003, pp. 57–64.
[19] T. Ye, D. Veitch, G. Iannaccone, S. Bhattacharya, Divide and conquer: PC-based packet trace replay at OC-48 speeds, in: in Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005 (Tridentcom 2005)., 2005, pp. 262–271.
[20] J. Sommers, P. Barford, Self-configuring network traffic generation, in: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, in: IMC '04, 2004, pp. 68–81.
[21] K. Vishwanath, A. Vahdat, Swing: realistic and responsive network traffic generation, IEEE/ACM Trans Netw 17 (3) (2009) 712–725.
[22] M.C. Weigle, P. Adurthi, F. Hernández-Campos, K.J. Kevin, F.D. Smith, Tmix: A tool For Generating Realistic TCP Application Workloads In ns-2, SIGCOMM Comput. Commun. Rev. 36 (3) (2006) 65–76, doi:10.1145/1140086.1140094.
[23] A. Botta, A. Dainotti, A. Pescap, A tool for the generation of realistic network workload for emerging networking scenarios, Comput Netw 56 (15) (2012) 3531–3547.
[24] C.V. Wright, C. Connelly, T. Braje, J.C. Rabek, L.M. Rossey, R.K. Cunningham, Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer Security, in: Proceedings of the 13th international conference on Recent advances in intrusion detection, RAID'10, Ottawa, Canada, 2010, pp. 218–237.
[25] S.A. Baset, H. Schulzrinne, An analysis of the Skype peer-to-peer internet telephony protocol, in: Proceedings of the IEEE INFOCOM, Barcelona, Spain, 2006.
[26] S. Molnár, P. Megyesi, G. Szabó, Multi-functional traffic generation framework based on accurate user behavior emulation, in: Proceedings of the IEEE INFOCOM (Demo), Turin, Italy, 2013, pp. 632–633.
[27] S. Molnár, P. Megyesi, G. Szabó, Multi-functional emulator for traffic analysis, in: Proceedings of the IEEE ICC, Budapest, Hungary, 2013, pp. 2397–2402.
[28] P. Megyesi, S. Molnár, Finding typical internet user behaviors, in: Proceedings of the 18th EUNICE Conference on Information and Communications Technologies, Budapest, Hungary, 2012, pp. 321–327.
[29] World of Warcraft, Retrieved: May, 2015. URL http://www.worldofwarcraft.com/index.xml
[30] A. Dainotti, A. Botta, A. Pescapé, G. Ventre, Searching for invariants in network games traffic, in: Proceedings of the 2006 ACM CoNEXT Conference, CoNEXT '06, 2006.
[31] W. chang Feng, F. Chang, W. chi Feng, J. Walpole, A traffic characterization of popular on-line games, IEEE/ACM Trans Netw 13 (3) (2005) 488–500.
[32] MSN Messenger, Retrieved: May, 2015. URL http://explore.live.com/messenger
[33] Microsoft Remote Desktop Connection, Retrieved: May, 2015. URL http://windows.microsoft.com/en-US/windows7/products/features/remote-desktop-connection
[34] Real VNC, Retrieved: May, 2015. URL http://www.realvnc.com
[35] Alexa: Top 500 Global Sites, Retrieved: May, 2015. URL http://www.alexa.com/topsites
[36] AJAX tutorial, Retrieved: May, 2015. URL http://www.w3schools.com/ajax/default.asp
[37] eBay, Retrieved: May, 2015. URL http://www.ebay.com/
[38] Eicar test virus, Retrieved: May, 2015. URL http://www.eicar.org/86-0-Intended-use.html
[39] Malware domain list, Retrieved: May, 2015. URL http://www.malwaredomainlist.com/
[40] AutoIt, Retrieved: May, 2015. URL http://www.autoitscript.com/site/autoit/
[41] AutoHotkey, Retrieved: May, 2015. URL http://www.autohotkey.com/
[42] Watir, Retrieved: May, 2015. URL http://www.watir.com/
[43] Sikuli IDE, Retrieved: May, 2015. URL http://www.sikuli.org/
[44] uTorrent, Retrieved: May, 2015. URL http://www.utorrent.com/
[45] PsExec, Retrieved: May, 2015. URL http://technet.microsoft.com/en-us/sysinternals/bb897553
[46] Expect, Retrieved: May, 2015. URL http://sourceforge.net/projects/expect/
[47] MonkeyRunner, Retrieved: May, 2015. URL http://developer.android.com/tools/help/monkeyrunner_concepts.html
[48] Intents and Intent Filters in Android, Retrieved: May, 2015. URL http://developer.android.com/guide/components/intents-filters.html
[49] A. Botta, D. Emma, A. Pescape, G. Ventre, Systematic performance modeling and characterization of heterogeneous ip networks, in: Proceedings of the 11th International Conference on Parallel and Distributed Systems, 2, 2005, pp. 120–124.
[50] tcpdump, Retrieved: May, 2015. URL http://www.tcpdump.org
[51] VPN Capture Android Application, Retrieved: May, 2015. URL https://play.google.com/store/apps/details?id=hu.edudroid.measurement_uploader
[52] T. Bujlow, K. Balachandran, M.T. Riaz, J.M. Pedersen, Volunteer-based system for classification of traffic in computer networks, in: Proceedings of the 19th Telecommunications Forum (TELFOR 2011), 2011.
[53] S. Molnár, P. Megyesi, G. Szabó, How to Validate Traffic Generators? in: Proceedings the 1st IEEE Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios (TRICANS 2013), Budapest, Hungary, 2013.
[54] nDPI, Retrieved: May, 2015. URL http://www.ntop.org/products/ndpi/
[55] B. Csatári, Framework for Comparison of Traffic Classification Algorithms, in: Master Thesis, 2011.URL http://www.crysys.hu/szabog/publications/diplomazok/csatari-thesis.pdf
[56] P. Abry, P. Flandrin, M.S. Taqqu, D. Veitch, Wavelets for the Analysis, Estimation, and Synthesis of Scaling Data, Wiley.

Géza Szabó is a Research Fellow at the Traffic Laboratory of Ericsson (Hungary). He has been working there since 2005. He received the M.Sc. degree in Computer Science from the Budapest University of Technology and Economics (BME) in 2006. During 2006-2009, he pursued his PhD studies in the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics and received his PhD degree in 2011 in the topic of traffic classification algorithms. His research interests focusing on mobile broadband networks mainly in connection with traffic measurements, traffic profiling, network management, radio related issues. He coauthored several journals (Journal of Multimedia Tools and Applications, IEEE Communications Surveys and Tutorials, etc.), conferences (ICC, Globecom, PAM, etc.) papers, and patents.

Péter Megyesi received his BSc and MSc in Electrical Engineering from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 2010 and 2012, respectively. Since 2012, he is a PhD student at the High Speed Networks Laboratory at the Department of Telecommunications and Media Informatics, BME. His PhD research is focused on synthetic network traffic generation. His research interests also include traffic measurements, traffic modeling and analysis and traffic identification. Since 2013, Péter is also enrolled in the Doctoral School on Innovation & Entrepreneurship organized by the Information and Communication Laboratory of the European Institute of Innovation and Technology.

Sándor Molnár received his MSc, PhD and Habilitation in Electrical Engineering and Computer Science from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 1991, 1996 and 2013, respectively. In 1995 he joined the Department of Telecommunications and Media Informatics, BME. He is now an Associate Professor and the principal investigator of the teletraffic research program of the High Speed Networks Laboratory. Dr. Molnár has participated in several European research projects COST 242, COST 257, COST 279 and recently in COST IC0703 on 'Traffic Monitoring and Analysis: theory, techniques, tools and applications for the future networks'. He was the BME project leader of the Gold Award winner 2009 CELTIC project titled 'Traffic Measurements and Models in Multi-Service networks (TRAMMS)'. He is a member of the IFIP TC6 WG 6.3 on 'Performance on Communication Systems'. He is a participant in the review process of several top journals and serves on the Editorial Board of the Springer Telecommunication Systems journal. He is active as a guest editor of several international journals such as the ACM Kluwer Journal on Special Topics in Mobile Networks and Applications (MONET). Dr. Molnár served on numerous technical program committees of IEEE, ITC and IFIP conferences working also as Program Chair. He was the General Chair of SIMUTOOLS 2008. He is a member of the IEEE Communications Society. Dr Molnár has more than 170 publications in international journals and conferences (see http://hsnlab.tmit.bme.hu/molnar for recent publications). His main interests include teletraffic analysis and performance evaluation of modern communication networks.