Flow Signatures of Popular Applications

Vladislav Perelman

Computer Science
Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany

Type: Guided Research Final Report
Date: May 9, 2010
Supervisor: Prof. J. Schoenwaelder

Abstract

Recently a lot of research has been done in the area of application identification based on the network activity. However, most of the work has succeeded only in the identification of the type of application used. In this research we were trying to find application signatures within the network flows that could be used to pinpoint certain applications, such as specific web-browser, mail client and so on. For this purpose, manual analysis of the flow records as well as the packet traces had to be done. As a result we have been able to identify several flow signatures which could be used for statistical or network administration purposes.

Contents

1 Introduction **3**

2 Related Work **3**

3 Experimental Setup **5**

4 Browsers **7**
- 4.1 Google Chrome 5.0.335.0 for Linux 8
- 4.2 Opera 10.10 for Linux 9
- 4.3 Firefox 3.5 for Linux 10
- 4.4 Windows versions 10

5 Instant Messaging Clients **10**
- 5.1 Skype 11
- 5.2 Yahoo Messenger 12
- 5.3 MSN Messenger 13
- 5.4 ICQ 14

6 Mail Clients **14**
- 6.1 Mozilla Thunderbird 15
- 6.2 Microsoft Outlook 15
- 6.3 Windows Live Mail 16

7 Media Players **17**
- 7.1 Amarok 17
- 7.2 iTunes 17
- 7.3 Windows Media Player 18

8 Conclusion **18**

1 Introduction

Being able to correctly identify concrete applications utilizing networks' resources is a difficult task which so far no one, to our knowledge, has managed to fully complete. If it were made possible however, it would result in very valuable information for network administrators, designers and other workers in this field. Such information could then be of great help to people who provide security, as it would allow them to immediately see hazardous or malicious software in use. Retrieved data could also be used for statistical purposes to give a precise percentage of various applications used. We propose a way of identifying certain applications by trying to find their network flow signatures.

To fully understand the idea behind the research one has to be familiar with the notion of a network flow. It refers to the sequence of packets that is sent from a host to some destination, be it broadcast domain, multicast group or another host. All of the packets within that sequence share all of the following 7 values: source IP address, destination IP address, source port, destination port, IP protocol, Ingress interface and IP Type of Service [1]. A single machine can receive hundreds and thousands of packets per hour when it is connected to a network; most of them are unnoticed while some carry information that is presented to the user (a web page, e-mail, audio/video streaming, etc.). Flow records summarize the network traffic seen at a network metering point (usually located in a router, but might be also located in the end systems). They show various information about the traffic in a given flow, for instance number of packets sent and received by an end point, timestamps for the flow start and finish time, etc.

Our hypothesis is that some applications generate a specific flow signature, which can be identified among the large data set. Identifying those signatures of popular applications is what this research is focused on. To put it differently, we are trying to find certain patterns in the flow records, which will pinpoint usage of a certain application such as a specific browser, IM client or a mail client.

The idea behind the research is not to understand what exactly each application is doing on a network flow level and why, but rather look over a wide variety of applications and try to find signatures of as many applications as possible.

2 Related Work

The idea of identifying applications by looking at the network activity is not new, it has been addressed for several years now and certain work has already been done in this area. However, in most of the cases the research was focused on identifying the type of the application used, rather than the specific application itself. There has been progress made in detecting whether the network flows belong to the applications of the following categories: web, p2p, data (ftp), mail, news, gaming,

etc. [2, 3] but there is yet to be developed a tool that could pin-point which specific application is being used.

Originally the type of application was identified in a very straightforward and simple manner - by looking at the port numbers used by the end-points. The reason why this method can work in some cases is the fact that some applications permanently use well-known port numbers. For instance, it is widely known that HTTP uses ports 80 and 8080, HTTPS uses port 443, etc. Clearly this method fails whenever applications allow users to change used port numbers or whenever those numbers are generated randomly by the operating system. Not only does this method fail to identify certain applications, but it can also easily misidentify an application if it is running over a well-known port of another application. This happens quite often nowadays, mostly because of the tendency to run applications over the HTTP port 80 to go through firewalls without being blocked.

Seeing how the old way of inspecting port numbers can no longer provide accurate results, new approaches of identifying applications started to appear. Among them one can distinguish between two different methods. One requires labor-intensive inspection of the full packet payload, as being done by A. Moore and K. Papagiannaki in [2]. Their technique approaches 100% accuracy but happens to be a very long and difficult 9-step procedure that could not yet be automated. An example of a completely different approach that operates "in the dark", that is with no access to packet payload or port numbers is BLINd Classification, or BLINC [3], an approach that looks at the patterns of host behavior at the transport layer and uses that information to classify hosts' flows. During its testing phase (monitoring traffic of university campus and biology-related research facilities) BLINC classified approximately 80%-90% of all flows with 95% accuracy.

A great deal of research has been done on identifying specifically P2P applications [4, 5, 6], as it is the most challenging type to identify. That is mainly due to two reasons. Firstly, most P2P applications do not use static, well-known port numbers but rather communicate over dynamically found random ports. Some applications also frequently mask themselves by using port numbers of other applications. The second reason for being so hard to identify is the vast number of various P2P applications that is growing constantly. All of them use different P2P protocols, hence any tool whose task is to identify P2P traffic will have to somehow adapt and be able to discover new protocols on the fly. Karagiannis et al. proposed an approach that is based on the transport layer characteristics and therefore does not rely on the user payload [7]. This method makes use of two heuristics, one of which examines the source-destination IP pairs that use both TCP and UDP to transfer data, another one monitors connection patterns of {IP, port} pairs. According to the developers of this method, it can identify more than 95% of the P2P flows with a false-positives range under 10%.

Another novel approach has been proposed by Couto et al., which is based on the use of neural networks [4]. The neural network is trained using a set of known

traffic values, associated with each application. Once the neural network model is sufficiently trained it can be used to identify Internet applications without looking at the packet content with correct identification percentages being above 90%.

One more direction of the research is identifying malicious software and preventing various attacks through the network. A lot of effort has been put into identifying threat signatures and developing automated detection and prevention systems [5, 6, 8].

Our research, unlike previously mentioned ones, will be focused on identifying specific applications rather than application types. We will try to find signatures of most-commonly used browsers, IM clients, and so on.

3 Experimental Setup

Normally flow records are collected on the router but during our research they have been collected on a laptop using fprobe – a tool that collects network traffic data and emits NetFlow version 5 records. These records contain the following information about the traffic in a given flow, which is used to detect applications' flow signatures:

- Timestamps for the flow start and finish time
- Source and Destination IP addresses
- Source and Destination Port numbers
- Protocol
- Cumulative flags (in case of TCP flow)
- Number of packets
- Number of octets.

The fprobe settings are configured such that a NetFlow collector is located at localhost on port 555. From that port NetFlow v5 records are received and exported to disk via the flow-capture utility one of the programs of the flow-tools collection (flow-tools is a set of programs created by Mark Fullmer, which are used to process NetFlow data). Flow-capture is configured to create one file every 5 minutes, which contains exported NetFlow v5 records.

To view information stored in flow files in human-readable format flow-print tool is used. Since we are interested only in the flows that are either sent by or sent directly to our IP address, we therefore have to filter out all the broadcast, multicast and other irrelevant flow records. To do so, the output provided by flow-print is first piped into the flow-filter tool, which allows to filter out all the unnecessary records leaving only the ones that we are interested in. They are then outputted

into a specified file.

Another program that was used during the research was Wireshark[9] an open source Network Protocol Analyzer (see Figure 1). Wireshark, apart from the information that can be retrieved using flow-tools, also provides full payload of each packet. This information is not needed to find the signatures of application per se, but can help understand reasons behind certain behavior of applications information that can later be used in further research.

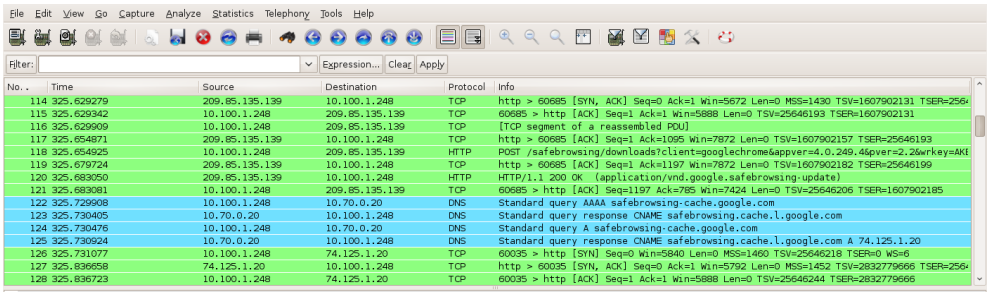

Figure 1. Screenshot of Wireshark. Apart from the basic information such as Source and Destination IP Addresses Wireshark also provides full payload information of each captured packet.

Since we are looking for signatures of certain applications, flow records collected by fprobe and packets collected by Wireshark are examined when only studied applications are running. For example, while looking into the behavior of various browsers on the opening of a blank page, we first waited until a new flow file was created by flow-capture and only then started one browser. Closing this browser within a 5 minute time span, so before the next flow-file was created, ensured that all flow records produced by the examined browser remained in one flow file, its contents then moved to a specified location as clear text as described above.

In order to structure all the obtained files, folders for each examined application were created. Each folder contains both .pcap files provided by Wireshark and flow-tools outputs, which are then manually inspected and checked for repeating patterns.

Throughout the time span of the research we were trying to identify flow signatures of several most popular applications among web browsers, IM clients, mail clients and media players. Moreover, in each category applications were looked at while running under different operating systems.

In order to select the most widely used applications among the huge variety that is currently present to the typical user we have conducted a small study among a group of students from Jacobs university. Participants were asked 12 questions regarding their general computer usage as well as specific preferences in various computer applications. More than 60 people took part in the study; quite expectedly the vast majority of the respondents appeared to use the most popular and

successful on the market applications, however browsers like Elinks and Netscape or email clients like Mutt and Pine still have their users.

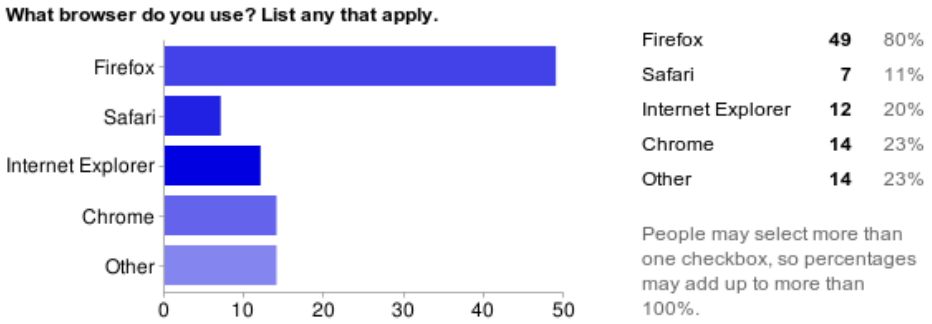

Figure 2. Chart presenting answers to one of the questions in the conducted study.

Based on the answers (See Figure 2 for an example) the following applications have been selected for the study:

Web Browsers	IM Clients
Mozilla Firefox	Skype
Google Chrome	MSN Messenger
Opera	Yahoo Messenger
	ICQ
Mail Clients	**Media Players**
Mozilla Thunderbird	Amarok
Microsoft Outlook	Microsoft Media Player
Windows Mail	iTunes

4 Browsers

The first class of applications that we have decided to start looking at was Internet browsers. The reason for that is simple – browsers provide a link between users and the World Wide Web, they make it possible to easily access information on the Internet and are therefore the most often used applications when it comes to working on-line. The number of currently available browsers is vast, each one with their own advantages and disadvantages, but all of them having one thing in common: they do a lot more than a usual user would expect. We have chosen to investigate into the specific behavior of the following browsers: Mozilla Firefox (Linux/Windows), Google Chrome (Linux/Windows), Opera (Linux/Windows).

All browsers were examined in the same fashion. The first round involved opening

a blank page (*about:blank*) on startup with no RSS feeds, no plug-ins or anything else that could cause network activity. This way it was guaranteed that any network activity detected was due to the start-up of the application and nothing else and can therefore be considered specific for the examined browser. The next step was opening a simple page (such as *http://www.google.com*) and checking for any behavior that is different across various browsers. The third step involved going through several web-pages to check if anything unusual happens that can pinpoint the usage of a certain browser. The fourth and last step was using RSS Feeds and popular plug-ins.

As it has already been pointed out all browsers do more than a mere user would expect. Most of this behavior that remains unseen for anyone without a packets capturing tool can be divided into several categories based on the reasons why certain actions take place: Security, Performance Improvement, Information Gain.
In the following subsections we will discuss findings for each of the examined browsers separately.

4.1 Google Chrome 5.0.335.0 for Linux

Google Chrome for Linux is a relatively new and an ongoing open-source project. This means that updated versions of the browser appear quite often, which increases chances of finding new and unknown behavior in the new versions. For instance the earlier version of Chrome (4.0) showed absolutely no network traffic on start-up. The current version appeared to be quite different from its "silent" ancestor. On every start-up Chrome 5.0 first sends a DNS query to the local DNS server asking for the IPv4 address of the host *clients1.google.com*. It is not at all surprising, since Chrome is a Google product after all. Within a few seconds after the start-up 18 DNS Type A queries are sent to the local DNS server. These queries look up the names $\{X,Y,Z\}$*.students.jacobs-university.de,* $\{X,Y,Z\}$*.jacobs-university.de,* $\{X,Y,Z\}$*.jacobs.jacobs-university.de* twice, where X, Y and Z are 10-letter strings which appear to be randomly generated (for example: rsobazcuyh or hfqfzqpxkt). The suffixes appended in the queries are taken from the search list provided by the DHCP server and are therefore dependent on the local configuration. Clearly, all those requests result in a "No such name" response. After coming up with several ideas on what this could be done for and whether it is a mere bug or not we decided to make use of the fact that Chrome for Linux is an open source project. Reading through the available online code [10] answered all the questions. The observed behavior is caused by an effort to determine whether the user is on a network that redirects requests for intranet hostnames to another site, and if so, tracking what that site is. This in turn is done to avoid erroneously showing an infobar which asks the user if the web address he entered was mistaken for the search query.

Within 5 minutes of start-up Chrome performs another procedure, which is ex-

plained on the web-page of the Chromium project [11]. Google Chrome contacts certain Google servers (*safebrowsing.clients.google.com* and *safebrowsing-cache.google.com*) – therefore asks DNS server for their IPv4 addresses beforehand – for a list of phishing and malware websites. In order to reduce the size of the information that has to be sent from the Google servers to the browser the full URLs of those websites are first hashed using SHA-256 and then the list of first 32 bits of each hash is sent to the user. This list is downloaded in the background and stored on the computer. Every visited website is then checked against this list to increase the security level. If a match is found Chrome asks Google to send a list of all URLs that fall under the matched 32-bit prefix to make sure that the accessed web-site really is considered malicious. The list is redownloaded approximately every half an hour.

To improve page loading time performance Google Chrome uses a trick called DNS prefetching. Since a user is likely to go from one page to another by clicking on the links embedded in the current page, Chrome automatically sends DNS queries of type A to find IPv4 addresses of all links present on the loaded page.

Currently Google Chrome does not support reading RSS Feeds; this feature, however, is under development. Among the plug-ins that were inspected were AniWeather[12] and Adthwart[13]. The first one is a very typical extension that provides information about weather conditions by contacting a server and downloading data every certain period of time. That behavior however is not browser specific and will be identical to that of the AniWeather plug-in for Mozilla Firefox. The AdThwart extension hides ads from the viewed web pages but since it does it only after information is loaded no difference in the traffic can be seen. Throughout the testing phase no traces of AdThwart being installed were found.

4.2 Opera 10.10 for Linux

Starting from version 10.10 Opera decided to give each browser functionality of a server by the means of the so-called Opera Unite technology. Opera Unite, if enabled, allows users to share files between computers without any help from intermediate servers. The interesting part is that even though Opera Unite is disabled by default, discovery of local Opera Unite users is not. This leads to a very specific network activity which cannot be misidentified. In less than a second after the start-up of the browser, Opera uses the Simple Service Discovery Protocol (SSDP)[14] and sends 3 identical multicast messages to 239.255.255.250 to port 1900. This leads to an identification of users on the local network that are using Opera Unite services. It also leads to a non-stop communication between all those users (using mostly TCP over port 2869). The described behavior, which can undoubtedly be considered as a signature of Opera 10.10, will disappear the moment a user changes settings of the browser to disable discovery of local Opera users.

There is also another feature that Opera provides, this time related to the Security

category, which can also be turned off by going to advanced settings, but considering how rare users check those settings, the behavior seen as a result of this feature can be considered a stable signature of this browser. In order to find out whether the web-server which is being contacted can be trusted or not, Opera consults a 3rd part – TRUSTe[15] by sending a GET request of certain form to *sitecheck2.opera.com*, which then responds with an XML-like message providing information about the web-server. As a result start-up of Opera is always followed by a DNS request to the local DNS server to get the IPv4 address of *sitecheck2.opera.com*.

RSS Feeds in Opera are updated by default every 3 hours; this time period however can be easily changed by the user. Opera also allows the user to install various widgets, which firstly do not start automatically on the startup of the browser and secondly do not have Opera specific features.

4.3 Firefox 3.5 for Linux

Firefox appears to be working in a very similar fashion to Google Chrome. Apart from the ISP redirect pages identification, Firefox uses the exact same way of checking for phishing and malware websites. It also prefetches IP addresses of all links on the current page to increase performance. After examining Firefox, we have come to the conclusion that the current version of Firefox has no exact signature that can be used for identification purposes.

4.4 Windows versions

As was mentioned in the beginning of this section both Linux and Windows versions of browsers were examined. It appeared that the only difference in the behavior of browsers under different operating systems was found in Google Chrome which did not send DNS requests for non-existent names during start-up. Both Firefox and Opera had the exact same network activity under Linux, as they did under Windows.

5 Instant Messaging Clients

The second class of applications that we have looked at was IM Clients. Communication over IM is becoming more and more popular every day, especially among teenagers and university students, mostly due to the ease of use, ability to communicate with multiple people at once and ability to transfer files. Using IM clients allows to stay in touch with friends and family and on top of that IM is free, even when you are calling someone on the other side of the world.

It appeared that distinguishing between various IM clients is a much more trivial

task than doing so with Internet browsers. It was clear from the start that ICQ contacts it's own servers and so do MSN Messenger or Yahoo Messenger. We, however, wanted to find some other behavioral patterns that would pin-point usage of a certain IM client apart from checking which server is being addressed. In the following subsections the results of our investigation will be discussed.

5.1 Skype

Created in 2002, Skype has rapidly become one of the most popular and successful VoIP/IM applications worldwide. Good quality of voice and video calls, end-to-end encryption of both calls and chat messages, working across NATs and firewalls – all of that made Skype popular among regular users. Being a close-source program with a far from trivial implementation attracted a lot of research on how Skype actually works [16], [17], [18].

Each time on login, Skype server *"skype.com"* connects to the client and waits for it to send an HTTP 1.1 GET request */getlatestversion* in order to check whether updates are available. After the information has been sent to the client, server tears down the connection. At login client also sends 4 SSDP messages and 4 NAT-PMP [19] Map UDP Request messages. Our original assumption was that the discovery protocol is used to find other Skype Clients on the network, however, the small number or received replies (3) implies that Skype is not trying to find all of them. NAT-PMP new port mapping request is sent over UDP to the port 5351 and according to the specification is resent up to 9 times if no response is received after 250ms, 750ms, 1,75s, etc. doubling waiting time each turn. This request is sent in order to identify whether the client is located behind the NAT and what is the IP address which is visible for the users from outside the network. However not all network routers support NAT-PMP and therefore if the port mapping request does not get a response, Skype switches to another way of finding out needed IP. Most-likely Skype is using a variant of STUN [20] protocol.

At each login Skype obviously has to contact a Skype login server. After logging in on Skype several times with different user names we observed that the client always connects to the server at 212.8.163.94, port 33033 and then the same exchange of messages is observed (See Figure 3). Messages (1) and (2) are always the same, their decimal representations respectively are *22 3 1 0 0* and *23 3 1 0 0*. Messages (3) and (4) were different for each login attempt. They, however, always had the same first four bytes as (1) and (2) respectively. On each successful login messages (3) and (4) had sizes of around 473 and 295 bytes, on a login failure however message (4) was always only 18 bytes.

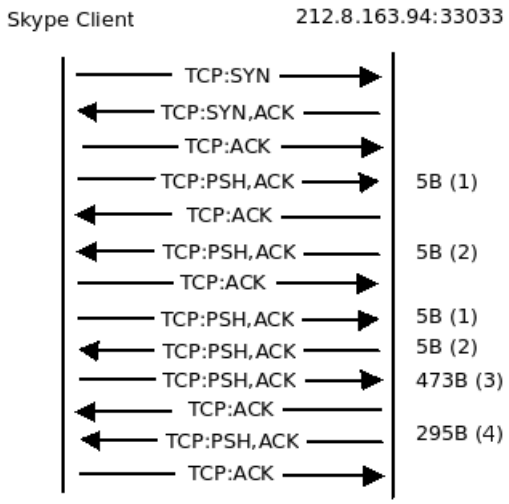

Figure 3. Messages exchanged at the Skype login.

The thorough inspection of the behavior of Skype on login as well as call establishment, media transfer, etc can be found in Salman A. Baset's and Henning G. Schulzrinne's analysis of Skype traffic [18]. This information can be used to easily identify Skype traffic among all the incoming and outgoing traffic.

5.2 Yahoo Messenger

Unlike Skype, Yahoo! Messenger follows a very straight-forward protocol that can be easily traced and is easily distinguishable among other network activity. On startup the client sends two DNS type A queries for *http://vcs1.msg.vip.sp1.yahoo.com* and *http://vcs2.msg.vip.ac4.yahoo.com*. Two HTTP 1.1 GET requests to those servers follow. As a response the client receives an IP address of one of the Yahoo! INC servers. Client application then starts using the YMSG protocol [21] (via TCP/IP connection on port 5050 by default) in order to communicate with that server. Each YMSG message starts with the same sequence of bytes: *59 4d 53 47*(YMSG), followed by the version number *11*, followed by the number representing the length of the message.

The client application first initiates the login procedure, which is the only time when encrypted data (authentication details) is sent to the server; all other data is sent in a binary format with text portions transmitted in clear text. After successful login the client remains logged in as long as the TCP/IP connection is open. Every 60 seconds after the login Yahoo! Messenger sends a specific YMSG Keep-Alive message (Service number 0) to the server, so that it does not teardown the connection. Whenever the user decides to logout, the client sends a Pager Logoff YMSG

message (Service number 2) to the server.

Since all the messages of the YMSG protocol can be easily identified in the network flow, Yahoo! Messenger has a clear, stable signature which can be used to pin-point its usage on the network.

5.3 MSN Messenger

Just like Yahoo! Messenger MSN Messenger uses its own protocol for communication – MSNMS [22], [23] – which makes it relatively easy to trace. During the login, a client first sends DNS Type A queries to find IP addresses of *login.live.com* and a dispatch server *messenger.hotmail.com*. Client then contacts the dispatch server and exchanges VER and CVR messages with it. VER defines version protocol used (MSNP18 for MSN Windows client, MSNP15 for Pidgin [24]). The CVR command sends information about the client version and operating system to the server. For the official Microsoft clients the server then replies with the recommended version of the client. Note, that Pidgin does not send the CVR message to the server – this can possibly be used to differentiate between official and third-party clients. However the server has no way of checking whether the sent information is correct, so technically any client can pretend to be an official MSN Windows client.

After exchanging information with the client, the dispatch server redirects it to the notification server, which is responsible for notifying the user about various events (messages, e-mails, etc.). The client and the notification server first also exchange VER and CVR, then the server sends a GCF message which starts with "GCF 0 4845" every time; the meaning of this message remains unknown to us. A connection to *login.live.com* on port 443 follows and the client receives encrypted data which is then used to authenticate with the notification server. The login procedure ends with the "USR 4 OK" message sent back to the client.

Whenever user A wants to start a conversation with user B, his client sends a message to the notification server, which replies with an IP address of a switchboard server. User B gets a message from his notification server and is then redirected to the same switchboard server. The conversation between two users then is relayed by the switchboard server. This server initiates a teardown of the connection after 5 minutes of inactivity.

All the MSNMS messages to the MSN servers are sent over TCP to port 1863. Chat messages, sent between users are not encrypted and transmitted in clear text.

5.4 ICQ

ICQ, being one of the AOLs products, uses AOLs protocol OSCAR – Open System for CommunicAtion in Realtime [25]. All messages which are related to the communication itself and are not dealing with downloading advertisement for the client interface are sent over https port 443 and are enclosed in the FLAP (Frame Layer Protocol) container [25]. Note that, despite using port 443 for communication, TLS protocol is not used, the client just uses the port which is always open. The first byte of the FLAP container is always 2a, it marks the start of the packet. The 3rd and the 4th bytes show the sequential number of the packet, it is incremented by 1 with every time a packet is sent. The 5th and 6th bytes indicate the length of the packet without the FLAP header.

At the start up, the official ICQ7 client first contacts *update.icq.com* asking it for various updates. The server responds either with HTTP 304 – Not Modified or with HTTP 200 – OK with attached data. When logging in, the client first contacts *api.oscar.aol.com* and tries to login right away. However if the server does not let the client log in, the client contacts *api.screenname.aol.com*, does two HTTP POST requests, receives two responses and then uses the just received data for authentication purposes when talking to the *api.oscar.aol.com*. After successful login, *api.oscar.aol.com* sends the client IP address of the server through which all the communication will take place. This server also sends all the information about the buddy list to the client.

6 Mail Clients

The purpose of each mail client is simple – it has to contact the server where e-mails are stored, retrieve them and display for the user to view. There are several protocols of how e-mails can be retrieved from the server, among them the two most prevalent ones: POP[26] and IMAP[27]. What protocol is being used depends on the particular mail server and therefore is not client-specific.

In order to find flow signatures that could be used to distinguish certain mail client applications we have looked at their behavior while contacting separately POP server and IMAP server as well as their unrelated to e-mail retrieval behavior. Looking at the different protocols separately is done because clients can be configured to check arbitrarily many servers so it is impossible to find a general pattern for their behavior. Following are the results of analyzing Mozilla Thunderbird 3.0.4 for Linux, Microsoft Outlook 2007 and Windows Live Mail when contacting *pop.mail.ru* (POP), *mail.messangingengine.com* (IMAP) and *imap.dfki.de* (IMAP).

6.1 Mozilla Thunderbird

Mozilla Thunderbird is an open-source cross platform e-mail client that was first released in 2004. On startup the client defaults to showing a welcoming page fetched from *http://live.mozillamessaging.com/thunderbird/start*, which therefore results in a DNS type A and type AAAA requests followed by establishing a connection with the server and an HTTP GET request.

When contacting a POP server Thunderbird first resolves the IP address of the server by separately sending both DNS AAAA and A type requests to the DNS server. A TCP connection is then established and communication using POP protocol follows; the server tears down the connection afterwards.

When talking to the IMAP server, the client first resolves its IP address in the same fashion, then opens a connection and most-likely gets the folder structure of the mailbox. Since even when encryption is disabled in the settings, Thunderbird still uses IMAP COMPRESS DEFLATE option it is impossible to know what exactly requests are being sent to the server. This behavior, however, does not influence the network flow, but rather makes it harder to explain it. In approximately 10 seconds another connection to the same server is established by the client (the first one remains open), this time probably to check for any new messages. By default Mozilla Thunderbird is checking for new messages every 10 minutes, during the time in between two checks both connections which were established on startup remain open. Every 10 minutes, the client initiates the check over the first connection. Both connections are closed by the client only when the application is closed.

6.2 Microsoft Outlook

Unlike Thunderbird, Microsoft Outlook on startup proceeds directly with the e-mail retrieval and does not contact any other servers but the e-mail ones. Communication with the POP server, however, is done in the exact same way, with the only difference that Outlook sends only one DNS request of type A. Another slight general difference with Mozilla Thunderbird is that Microsoft Outlook defaults to checking for new mail every 30 minutes.

When communicating with the IMAP server Outlook establishes two connections with this server right away. The first one is used to find out the folder structure of the mailbox, this connection remains open until the application is closed in case new mail arrives before the next scheduled check. Second connection is used to check for new messages in each folder, and it is then closed by the client after approximately 40 seconds. When Microsoft Outlook has to check for new mail again, a new connection is established which is again closed after about 40 seconds.

6.3 Windows Live Mail

On every startup Windows Live Mail contacts *config.messenger.msn.com* at 207.46.96.145. Since the DNS request to receive this IP address was sent only at the first use of the client, Windows Live Mail most-likely stores the address internally. After connecting to the server it sends an HTTP GET request and fetches an XML configuration file of about 20Kb. Checking for new e-mails at the mail servers follows and repeats by default every 30 minutes. Just like Microsoft Outlook and Mozilla Thunderbird, communication with a POP server is done with one TCP connection that is being closed by the server at the end of the dialog.

Communication with an IMAP server starts by initializing one TCP connection with is used to fetch information about the folder structure as well as the total number of messages and number of unseen messages in each folder. After that the client tears down this connection and starts opening and closing new connections one after another as many times as there are folders in the mailbox each time retrieving new e-mails if there are any. This way during one check of new messages on the IMAP server Windows Live Mail initiates *n+1* handshakes and tear-down procedures where *n* is the number of folders in the mailbox.

Figure 4 below illustrates different ways of communicating with IMAP servers used by Mozilla Thunderbird, Microsoft Outlook and Windows Live Mail. Communication is shown from top to bottom, different arrows (solid vs. dotted) describe different connections. IMAP refers to some activity using IMAP protocol. Content in the boxes refers to a repeating action, for example at every mail check.

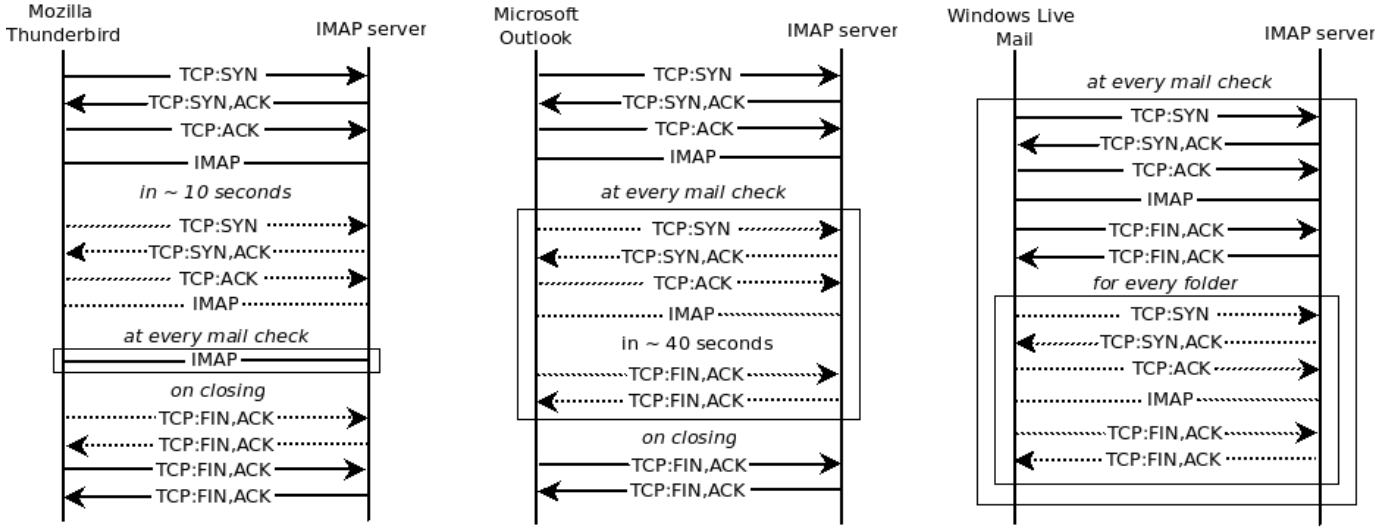

Figure 4. Communication with IMAP server used by Mozilla Thunderbird, Microsoft Outlook and Windows Live Mail.

7 Media Players

Media clients, unlike all previously mentioned types of applications, do not need a network in order to perform their primary task – playing media files, which are stored locally. This, however, does not mean that such applications do not use the network when it is available. On the contrary, most of them have different features or plugins, such as fetching album covers or music sharing, that rely on the network and result in a specific network flow. Following are the results of the investigation of three popular music players – Amarok 2.2.0 for Linux, iTunes 9.1.1 for Windows and Windows Media Player.

7.1 Amarok

Amarok is an open source music player originally created for Linux and is now also available under Windows. On startup Amarok uses the Multicast DNS (MDNS) protocol in order to discover users on the local network that share their music libraries. The clients starts by sending one MDNS query of type PTR for *_daap._tcp.local* to *224.0.0.251* at port 5353. The same query is then sent after one second, next one after two seconds and so on, doubling the waiting period every time. The name DAAP, appearing in the query, stands for Digital Audio Access Protocol[28] – proprietary protocol introduced by Apple in its iTunes 4.0 to share media across the local network. Though the description of the protocol has not been published, it has been reversed-engineered and reimplemented in several music players, Amarok being one of them. After a client discovers shared media libraries on the network it tries connecting to them using DAAP, which leads to TCP connections being established with various IP addresses but always at port 3689.

One of the unique features of Amarok is the opportunity to add various applets to the user interface. For instance, one can add a Lyrics applet, which would automatically download lyrics of a song from an online source like *lyrics.wikia.com* and display them. Applets displaying videos and photos related to the current song would contact hosts like *youtube.com* and *flickr.com*. Since for each new song information to display changes, Amarok with such applets installed would create a TCP connection, retrieve needed data and close the connection every ”song duration” interval, which can be approximated to 3-4 minutes. In this scenario it would be possible to find out whether a user is listening to music just by looking at the network flow.

7.2 iTunes

As was mentioned already, Apple’s iTunes was the original media client that included a feature of sharing one’s music library on the local network. Therefore

the first thing iTunes does on startup is it sends an MDNS query to *224.0.0.251* as was discussed in the previous subsection. Unlike the case with Amarok however, the waiting time between two consecutive queries does not double, but triples up. Apart from gathering information about the libraries on the network, iTunes also contacts *ax.init.itunes.apple.com* on startup (sends a Type A DNS query beforehand) and downloads an XML document that contains a signature and a signing key, which are used by Apple for verification purposes when the client is connecting to the iTunes store.

7.3 Windows Media Player

When using Windows Media Player (WMP), a user has a choice of visiting one of the several available online music stores. Various information about those online stores, such as description of the services they provide, WMP downloads on startup from *onlinestores.metaservices.microsoft.com*. Also when a new song is added to the library WMP automatically fetches information about the corresponding album and the album cover from the microsoft servers – *toc.music.metaservices.microsoft.com* and *images.metaservices.microsoft.com*. Apart from this network activity, however, Windows Media Player does not produce recognizable network flow. In other words, we have not been able to find its flow signature.

8 Conclusion

In this paper we have presented findings of the guided research on the flow patterns of several popular application. These findings lead to the conclusion that it is indeed possible to pin-point usage of certain applications based on their network activity – something that has not been done before. After analyzing web browsers, IM clients, mail clients and media clients, we have been able to identify flow signatures of Opera 10.10, Mozilla Thunderbird, Yahoo Messenger and several others. The process of finding clear flow signatures, however, is time-consuming and requires manual examination of the data provided by Wireshark and flow-tools.

Results of this research can be used by network administrators for statistical and security reasons. However one needs to realize that like any other technique, signature identification has its limitations. Found signatures are valid only for the specific version of the application and there is no guarantee that the signature will stay the same after an update.

In order to put the findings of our research to practice one would need a way of checking the incoming traffic flow against found signatures. Recently, at the CNDS group at Jacobs University, there has been developed and implemented a Stream-based IP flow record query language – Flowy[29]. Writing the queries based on

the signatures and integrating them with Flowy could be seen as a possible continuation of presented work.

References

[1] Network flows. http://en.wikipedia.org/wiki/NetFlow, last visited on May 9, 2010.

[2] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In Constantinos Dovrolis, editor, *PAM*, volume 3431 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2005.

[3] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 229–240, New York, NY, USA, 2005. ACM.

[4] A. Couto, A. Nogueira, P. Salvador, and R. Valadas. Identification of peer-to-peer applications' flow patterns. pages 292 –299, april 2008.

[5] Sen Subhabrata and Spatscheck Oliver. Accurate, scalable in-network identification of p2p traffic using application signatures. In *The 13th International World Wide Web Conference.*, pages 512–521. ACM Press, 2004.

[6] Myung sup Kim, Hun jeong Kang, and James W. Hong. Towards peer-to-peer traffic analysis using flows. In *In Self-Managing Distributed Systems, 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 55–67, 2003.

[7] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc Claffy. Transport layer identification of p2p traffic. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, New York, NY, USA, 2004. ACM.

[8] Piotr Kijewski. Automated extraction of threat signatures from network flows. In *18th Annual FIRST Conference*, June 2006.

[9] Wireshark. http://www.wireshark.org, last visited on May 9, 2010.

[10] Chromium Project Code, January 2010. http://src.chromium.org/viewvc/chrome?view=rev&revision=35807, last visited on May 9, 2010.

[11] Ian Fette. Understanding phishing and malware protection in google chrome, November 2008. http://blog.chromium.org/2008/11/

understanding-phishing-and-malware.html, last visited on May 9, 2010.

[12] Aniweather. http://www.aniweather.com/, last visited on May 9, 2010.

[13] Adthwart. https://chrome.google.com/extensions/detail/cfhdojbkjhnklbpkdaibdccddilifddb, last visited on May 9, 2010.

[14] Simple service discovery protocol. http://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol, last visited on May 9, 2010.

[15] Truste. http://www.truste.com/index.html, last visited on May 9, 2010.

[16] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking down skype traffic. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[17] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. *SIGCOMM Comput. Commun. Rev.*, 37(4):37–48, October 2007.

[18] Salman Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *INFOCOM 2006. The 25th Conference on Computer Communications. IEEE*, 2006.

[19] Nat port mapping protocol. http://files.dns-sd.org/draft-cheshire-nat-pmp.txt, last visited on May 9, 2010.

[20] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs), 2003. RFC 3489.

[21] Yahoo messenger protocol. http://en.wikipedia.org/wiki/Yahoo!_Messenger_Protocol, last visited on May 9, 2010.

[22] Chris Sanders. *Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems*. No Starch Press, 2007. Chapter 6: Common Protocols.

[23] Zhen Xiao, Lei Guo, and John Tracey. Understanding instant messaging traffic characteristics. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, page 51, Washington, DC, USA, 2007. IEEE Computer Society.

[24] Pidgin - the universal chat client. http://www.pidgin.im, last visited on May 9, 2010.

[25] Oscar protocol. http://en.wikipedia.org/wiki/OSCAR_Protocol, last visited on May 9, 2010.

[26] M. Rose J. Myers. Post office protocol - version 3, 1996. RFC 1939.

[27] M. Crispin. Internet message access protocol - version 4rev1, 2003. RFC 3501.

[28] Digital audio access protocol. http://en.wikipedia.org/wiki/Digital_Audio_Access_Protocol, last visited on May 9, 2010.

[29] J. Schoenwaelder V. Marinov. Design of a stream-based IP flow record query language, 2009.