MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Analysis and detection of Skype network traffic

DIPLOMA THESIS

Luboš Ptáček

Brno, spring 2011

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Advisor: RNDr. Jan Vykopal

Acknowledgement

I would like to express my gratitude to RNDr. Jan Vykopal for supervising my work and thank Jan Hapala for advices.

Abstract

This thesis deals with traffic identification of the Skype application. Payload based analysis of the standby traffic and voice calls is done. Basic Skype flow patterns are used to create a plugin for NfSen to display potential voice calls in the network.

Keywords

Skype, voice calls, traffic, payload, analysis, NetFlow, NfSen

Contents

Chapter 1

Introduction

Skype [20, 24] is a software application that allows users to make voice and video calls and to chat over the Internet. Unregulated Skype usage by employees for leisure and private purposes can lead to economic loss. Skype can often circumvent network restrictions like NAT and firewall by traversing them. Therefore enterprises are seeking solutions to regulate Skype activities over their networks. Skype establishes many concurrent connections in a rapid manner, which can be recognized as undesirable behaviour. Skype traffic detection can be categorized into payload-oriented and into nonpayload-oriented approaches.

On the other hand, success of the application comes from user friendly operation, it can operate without manual user configuration. This user friendliness is largely due to ability to detect the current network configuration and use of mechanisms to circumvent many applied network restrictions.

Chapter 2

Skype history

Skype was founded in 2003 by Niklas Zennström from Sweden and Janus Friis from Denmark. The Skype software was developed by Estonians Ahti Heinla, Priit Kasesalu and Jaan Tallinn [13], who were also behind the peer-to-peer file sharing software Kazaa [22]. In April 2003, Skype.com and Skype.net domain names were registered. In August 2003, the first public beta version was released.

One of the initial names for the project was "Sky peer-to-peer", which was then abbreviated to "Skyper". However, some of the domain names associated with "Skyper" were already taken. Dropping the final letter left the current title "Skype", for which domain names were available. Calls to other users within the Skype service are free, while calls to both traditional landline telephones and mobile phones can be made for a fee using a debit-based user account system. Skype has also become popular for its additional features which include instant messaging, file transfer, and video conferencing. Skype has 663 million registered users as of 2010 [15]. The average number of users connected each month was 145 million in the fourth quarter of 2010, versus 105 million a year earlier, while paying customers rose over the same period to an average 8.8 million per month, from 7.3 million. Skype reached a record with 30 million simultaneous online users on 28 March 2011 [14]. The network is operated by Microsoft Skype Division, which has its headquarters in Luxembourg. Most of the development team [13] and 44% of the overall employees of Skype are situated in the offices of Tallinn and Tartu, Estonia. eBay acquired Skype Limited in September 2005 and in April 2009 announced plans to spin it off through an initial public offering in 2010. It was acquired by Silver Lake Partners in 2009. Microsoft agreed to purchase Skype for $8.5 billion on May 2011 and the company is to be incorporated as a division of Microsoft called Microsoft Skype Division.

Some network administrators have banned Skype on corporate, government, home, and education networks, citing reasons such as inappropriate usage of resources, excessive bandwidth usage and security concerns.

Chapter 3

Skype

3.1 Skype entities

According to [10], we can distinguish some entities in the communication framework.

Skype client (SC) End client which places voice calls. Each *SC* maintains a record *host cache* which contains IP addresses and port numbers of *super nodes*. Notation SC_A and SC_B denotes the *SC* of the Caller and Callee respectively.

Supernode (SN) Online nodes that maintain the skype overlaying network. As described in [7], a supernode performs routing tasks such as forwarding requests to appropriate destinations and answer to queries from other *SC*s or *SN*s. A supernode can also forward login requests in case the login server is not directly reachable from a *SC*. Any *SC* with a public IP address can be promoted to an *SN* without the awareness of the *SC* host. This behaviour can be switched off by changing the registry entries.

Skype HTTP Server (HS) The HTTP server of *ui.skype.com*.

Login server (LS) An *SN* that Skype uses to provide authentication services to *SC*s.

Neighbour supernode (NSN) *SN*s that are logically near to an *SC*. An *SC* must establish connection to some *SN*s for Skype communications. The *SC* locates and then binds to its *NSN* for such purpose. An *SN* can be the *NSN* of multiple *SC*s simultaneously.

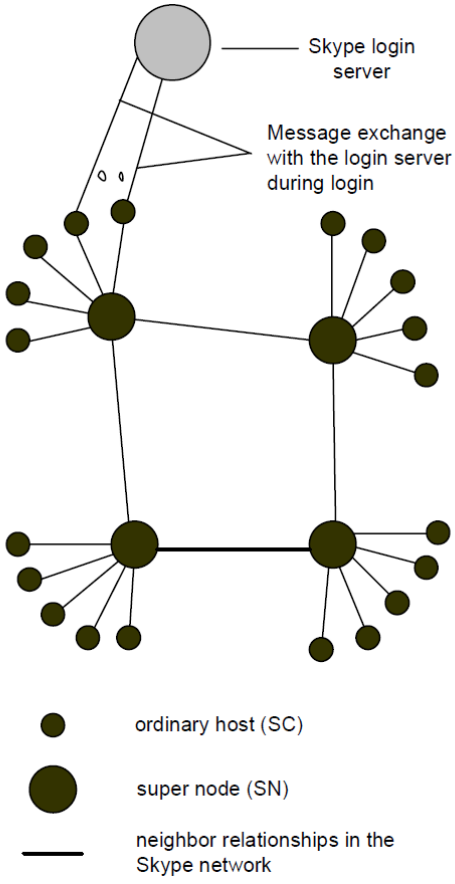


Figure 3.1: Basic Skype network, figure from [4]

3.2 Key components

I have performed traffic analysis and network measurements for the Windows Skype versions 4.1.0.136, 4.2.0.187 and 5.3.0.111 unless otherwise stated.

3.2.1 Ports

Skype uses UDP and TCP protocols for communication. *SC* has normally three listening ports enabled. They are configured in the Connection dialog box. The first one is randomly chosen during the application installation and is higher then 1024 according to [16]. Then there are open listening ports 80 and 443 which can be disabled in the dialog box. *SC* is reachable by the first port number for UDP and incoming TCP traffic. For TCP connection, Skype tries to contact other hosts at ports higher than 1024. If there are network restrictions applied, it tries again to this host on port 443. If this attempt fails, it will try on port 80. I have seldom observed connection on port 80. The reason could be that *SC* tries concurrently to initiate connection with other hosts and there will be a successful connection earlier than the need of making an attempt on port 80. So the ports 443 and 80 serve as a fallback precaution.

3.2.2 Host cache

Skype network is an overlay network and thus each *SC* maintains a table of reachable supernodes. It is called *host cache (HC)* and is stored in a XML file "shared.xml" (for Windows 7 in `C:\Users\<user name>\AppData\Roaming\Skype\`). The *HC* contains a maximum of 200 entries. The file *shared.xml* contains element `<HostCache>` with a hex string (`<HostCache>` with 200 entries has around 12800 characters). An example of the `<HostCache>` element follows.

```
<HostCache>
  41C80105004105020059A9FF2F9DC20001040002DCCDF0DE
  040003DCCDF0DE04000400050041050200
  ...
  04000400050041050200
  D5F0C7F6DE02
  0001010002
  A2A4E5DE
  040003
  D6E2E5DE
```

```
  04000400050041050200
  ...
  040004000500410502005D7B32BFF5930001040002 80CCF0DE
  04000380CCF0DE04000400
</HostCache>
```

From the knowledge that the string contains list of supernodes IP addresses and corresponding ports, I can find some delimiters inside this string. The main delimiter is the substring `04000400050041050200`. It separates the list of supernodes and after it there are 12 characters. Decoding them as IP address and port number we can obtain address 213.240.199.246 and port 56834. Substring `0001010002` is the next delimiter (the middle character with value `1` can have different values). Substring `040003` separates two substrings – `A2A4E5DE` and `D6E2E5DE` – which have sometimes equal value. These two substrings differ for the supernodes, I have not decoded them so their purpose is unknown.

3.2.3 Encryption

All Skype-to-Skype voice, video and instant message conversations are encrypted as described in [20]. Skype uses the Advanced Encryption Standard, also known as Rijndael, which is used by the US Government to protect sensitive information, and Skype uses the maximum 256-bit encryption. User public keys are certified by the Skype server at login using 1536 or 2048-bit RSA certificates.

3.3 Stages in Skype network conversation

3.3.1 Startup and UDP probing

The client application is started at SC_A. It sends UDP messages to multiple *SN*s saved in client's *host cache* HC_{SC_A} till positive response. *SC* initiates two-way UDP handshakes to the supernodes stored in *HC*. The source port is the one stored in Connection dialog box and destination ports are ports corresponding to supernodes in the *HC*. The notation for messages in these handshake probes will be P1 and P4. The purpose of handshakes is to determine possible candidates that allow the *SC* to connect to the Skype network.

We can distinguish different payload sizes for messages – notation for the message X is s_X. If $s_{P4} = 18$ bytes, then we can call this handshake as *positive*. *SC* then tries to establish a TCP connection to the positively

probed supernode. If $s_{P4} = 51$ or 53, then we can call this handshake as *negative*. No TCP connection will be initiated between Skype client and supernode. This UDP probing continues in rapid and continuous manner until positive is present.

Sometimes we can observe handshakes consisting of 4 messages P1, P2, P3 and P4 at the start of the application. It holds for the message sizes that $s_{P2} = 11$ bytes, $s_{P3} = s_{P1} + 5$. Analysing the payload of the messages we can differentiate some byte sequences according to [7] – *session identifiers*, *function parameters*, *IP address exchange*.

Session identifiers P1 is a initiating message so the first two bytes forms the session identifier. The important observation is that the session identifier is equal for P2 and P3 too but not for P4. In fact, this identifier is a number that is increased by two on every new request.

Function parameter The third byte of UDP messages has particular values so we can consider it as some kind of function parameter. It holds that it contains value $0x02$ for P1 and P4. The value is different for P2 and P3 but the lower nibble is always the same (a nibble is an aggregation of four bits so there are two nibbles in a byte – the higher one and the lower one). It is $0x7$ for P2 and $0x3$ for P3. It should be mentioned that for data packets like voice packets the lower nibble equals to $0xd$. There is one more value that is the same through the P3 payloads. The fourth byte possesses $0x01$ value.

IP address exchange We can distinguish four four-byte sequences in the messages. Notation for the payload bytes x to y for the message X is X_{x-y}. $P2_{4-7}$ contains IP address of the *SC*, $P3_{9-12}$ contains IP address of the supernode. The *SC*'s IP address in this message is never private address. It is the publicly visible IP address created by the NAT closest to the supernode. This is called a reflexive transport address according to [9]. Then sequences that are not IP addresses can be observed. I assume them as some unique message identifiers: $P1_{8-11} = P3_{13-16}$ and $P2_{8-11} = P3_{5-8}$.

Byte sequences are depicted in the following tables. SC_1–SC_4 is *SC*'s IP address, N_1–N_4 *SN*'s IP address.

$P1$	$SI\,SI$	02	$xx\,xx\,xx\,xx$	$A_1\,A_2\,A_3\,A_4$	$xx\,xx\,\ldots$

$P2$	$SI\,SI$	$x7$	$SC_1\,SC_2\,SC_3\,SC_4$	$B_1\,B_2\,B_3\,B_4$	

$P3$	$SI\,SI$	$x3$	01	$B_1\,B_2\,B_3\,B_4$	$N_1\,N_2\,N_3\,N_4$	$A_1\,A_2\,A_3\,A_4$	...

$P4$	$xx\,xx$	02	xx ...

We can refer to a *full UDP probe* if P1, P2, P3 and P4 messages are present and to a *partial UDP probe* if only P1 and P4 messages are present. As we can detect particular IP addresses in the full UDP probes we can consider these probes as some part of NAT detection algorithm which operates similar to STUN (Session Traversal Utilities for NAT) [9].

The *SC* continues in the partial UDP probing even after it has been logged into the Skype network.

3.3.2 TCP handshake with supernode

In this step Caller registers to Skype network. A TCP request is sent to the positively responding supernode from the previous step. The TCP connection is established to the same port the UDP probe used.

TCP problems If there is a problem with establishing a connection to the positively UDP probed supernode, then Skype will start initiating connection over ports 443 and 80. The tricky thing is that Skype does not follow communication protocols associated with these well-known ports. Skype employs some modification of TLS protocol [2] for port 443 as described further.

Port 443 It holds for the first message $M1$ sent to the supernode that $s_{M1} = 72$ bytes. The main observation is that the message payload starts with 56 byte sequence:

80	46	01	03	01	00	$2d$	00
00	00	10	00	00	05	00	00
04	00	00	$0a$	00	00	09	00
00	64	00	00	62	00	00	08
00	00	03	00	00	06	01	00
80	07	00	$c0$	03	00	80	06
00	40	02	00	80	04	00	80

Using Wireshark to view details of captured network traffic I can decode this sequence as follows. It is TLS 1.0 `Client Hello` message in SSLv2 record layer.

Values {80 46} mean the length of the message (that is 70), {01} handshake message type (here Client Hello), {03 01} the version of the TLS protocol by which the client wishes to communicate during this session (here TLS 1.0), {00 2d} cipher specification length (here 45), {00 00} session ID length (here 0), {00 10} challenge length (here 16), {00 00 05 00 00 04 00 00 0a 00 00 09 00 00 64 00 00 62 00 00 08 00 00 03 00 00 06 01 00 80 07 00 c0 03 00 80 06 00 40 02 00 80 04 00 80} list of the cryptographic options supported by the client. R1 messages differ only in the last 16 bytes and it should represent the `challenge` attribute.

Response message R2 from the supernode contains always at the first 79 bytes these values:

16	03	01	00	$4a$	02	00	00
46	03	01	40	$1b$	$e4$	86	02
ad	$e0$	29	$e1$	77	74	$e5$	44
$b9$	$c9$	$9c$	$b4$	31	31	$5e$	02
dd	77	$9d$	15	$4a$	96	09	ba
$5d$	$a8$	70	20	$1c$	$a0$	$e4$	$f6$
$4c$	63	51	ae	$2f$	$8e$	$4e$	$e1$
$e6$	76	$6a$	$0a$	88	$d5$	$d8$	$c5$
$5c$	ae	98	$c5$	$e4$	81	$f2$	$2a$
69	bf	90	58	00	05	00	

This start of the message is similar to to the TLS `Server Hello`. Value {16} means content type (here `handshake`), {03 01} means TLS 1.0, {00 4a} means length (here 74), {02} handshake type (here `Server Hello`), {00 00 46} length (here 70), {03 01} means TLS 1.0.

Bytes $R2_{12-15}$ should be `gmt_unix_time`. In the TLS protocol description is `gmt_unix_time` "the current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, GMT)". But the $R2_{12-15}$ bytes have fixed value {40 1b e4 86} and that is "Jan 31, 2004 18:23:18 Central Europe Standard Time".

Then there are 28 bytes in the `random_bytes` field, one byte of `session ID length`, 32 bytes of `session ID`. Field with bytes $R2_{77-78}$ has the value 00 05 and according to TLS it should be "the single cipher suite selected by the server from the list in Client Hello" – here it is `TLS_RSA_WITH_RC4_128_SHA`.

Port 80 In Skype version 2.0 there were noticed value recurrences in the higher nibbles of the R1 message as mentioned in [7]. I have not recorded these recurrences or any other in the messages to the port 80 in Skype versions 4.1 and 4.2.

3.3.3 Authentication

After connection to some supernode with destination port 33033, *SC* tries to authenticate itself to the Skype network. In previous versions of the Skype application there were four TCP messages exchanged in every connection to *LS* in general and after that the connection is closed as mentioned in [7]. I have measured payload pattern independently in all tested Skype versions but only in communication with particular *LS*s like *LS* with IP address `195.46.253.219`. My assumption is that the pattern depends on particular login server or some condition. When the pattern occurs then the first six messages M_1-M_6 exchanged between *SC* and *LS* look like depicted further.

M_1	$SC \rightarrow LS$	16 03 01 00 00
M_2	$SC \leftarrow LC$	17 03 01 00 00
M_3	$SC \rightarrow LS$	16 03 01 00 00
M_4	$SC \leftarrow LS$	17 03 01 00 00
M_5	$SC \rightarrow LS$	16 03 01 00 xx 42 cd ef $e7$ 40 $d7$ 2f 1d ...
M_6	$SC \leftarrow LS$	17 03 01 01 ...

Messages M_1-M_4 have fixed size of 5 bytes as opposite to M_5 and M_6. The fourth byte of the message M_6 does not contain a fixed value. This holds for the connections on ports 443 and 80 too.

3.3.4 Skype latest version check

When the client is first installed, it sends an HTTP request to `ui.skype.com`. The request method is `GET` and we can find in the request URI following patterns: `/ui/` and `/installed`.

If Skype has already been installed, it will check with *HS* the latest version of the application. This check occurs every time Skype is started. Skype client sends an HTTP request and the patterns are `/ui/` and `/getlatestversion?ver=`.

/ui/0/5.3.0.111./en/installed ?info=google-toolbar:notoffered; ienotdefaultbrowser2,google-chrome:notoffered; alreadyoffered

Table 3.1: Example of the `GET` request after Skype installation

/ui/0/5.3.0.111./en/getlatestversion?ver=5.3.0.111 &uhash=1d1bb29ea1f2970757800d8e22b9ce8d6 &google-chrome:notoffered; alreadyoffered

Table 3.2: Example of the `GET` request after Skype start

I have captured this `GET` request regularly every 4 hours and furthermore requests with the empty request URI can occur more often.

3.3.5 NAT and firewall determination

At the transport layer, Skype is performing NAT and firewall detection. NAT traversal is an important function of Skype for determining what kind of NAT settings is the *SC* currently behind. Once determined, the client stores this kind of information in the "shared.xml" file. Each *SC* uses a variant of the STUN protocol to determine the type of NAT the client is behind. There is no global NAT and firewall traversal server. If there was one, the *SC* would have exchanged traffic with it at the start, but no IP address repetition or some specific behaviour at particular stages of the Skype session was observed.

3.3.6 Going online

If the client starts only with the Offline status, no further traffic is recorded. Going Online, UDP packets are sent to multiple *SN*s rapidly and continuously. This kind of UDP pinging messages is detected during the whole Skype session. In this step, *SC* searches for available *NSN* and binds to it with TCP connection.

After positive UDP handshake with some *SN* as described previously, *SC* initiates a TCP connection to this node. If the connection is successful, then *SC* and *SN* exchange messages. If the connection is kept, then *SN* becomes *neighbour supernode* for the Skype client. This binding lasts as long as the *SC* is online but sometimes termination of the connection can

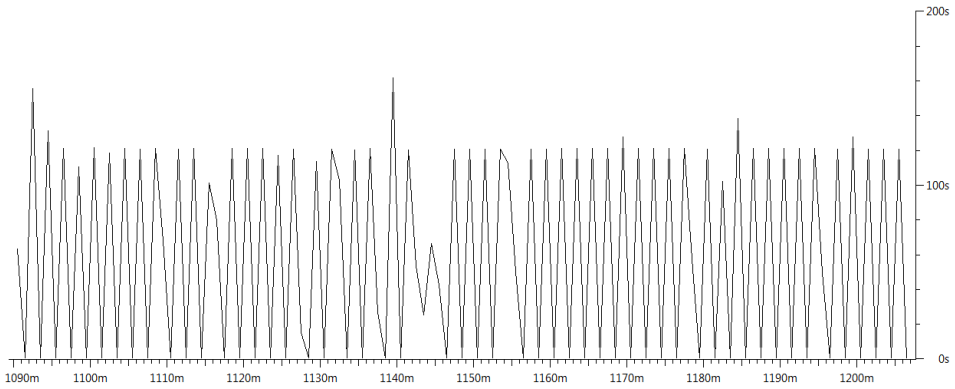


Figure 3.2: At least every 120 seconds *SC* sends TCP packet to its *NSN*.

occur (e.g. I have measured one 39 hours long connection till termination occured). My prediction is that it is caused by unavailability of the *NSN* or the supernode becomes a regular node. As the *SC* is always bind with some *NSN*, a new binding process starts in the same way as before. I did not find any particular pattern that would be common for every startup of *SC*. However, there can be often observed packets with payload sizes 8, 27 (or 28), 4 bytes initiated by *SC* between first 10 packets exchanged between *SC* and *NSN*.

There is probably some timeout counter used on the both sides as we can find regular traffic. I have found out that if any of the parties sends packets containing some signalling information in the 120 seconds time window, then *SC* or *NSN* sends a packet with $s = 2$ and begins a sequence of messages M_1–M_4. They can be some keep-alive messages. Timeouts start to count down for corresponding parties from the moment when M_1 and M_2 are sent.

M_1	$\rightarrow$	$s_{M_1} = 2$	$TCPPush, Ackflags$
M_2	$\leftarrow$	$s_{M_2} = 0$	$TCPAckflag$
M_3	$\leftarrow$	$s_{M_3} = 2$	$TCPPush, Ackflags$
M_4	$\rightarrow$	$s_{M_4} = 0$	$TCPAckflag$

Periodicity in sending packets is shown in Figure 3.2. I have captured almost 135 hours of standby Skype traffic with the longest continuously measured time slot of 67 hours. I have counted up inter packet time gaps for outgoing packets in one minute time slots and that has exposed peaks of 120 seconds almost every two minutes so it holds true. Moreover, it holds for the standby mode that for most outgoing packets with payload greater than 200 bytes the inter packet time gap equals to 15 minutes, Figure 3.3.

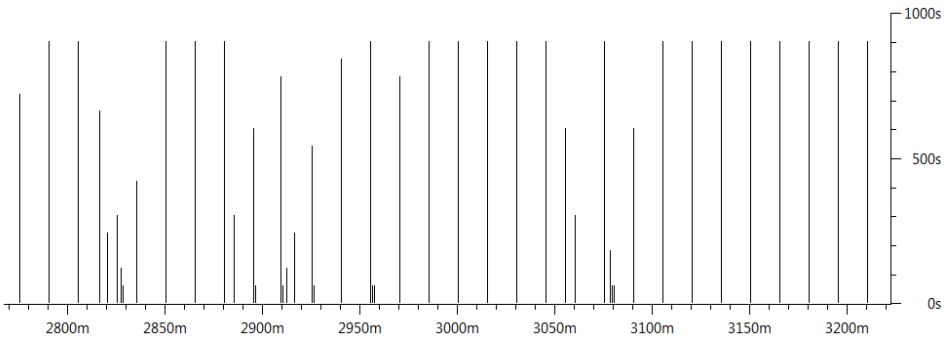


Figure 3.3: Inter packet gap of 900 seconds (i.e. 15 minutes) for outgoing TCP packets to *NSN* is observed every 15 minutes.

This was confirmed only for Skype versions 4.0 and 4.2.

From my measurements, *SC* also establishes in average 5 short TCP connections per hour. UDP probing as checking available peers in the Skype network occurs mainly in bursts. In average, 30 new peers are probed every hour (approximately 1900 per 67 hours) as shown in Figure 3.4, which shows us statistics for the 67 hours lasting Skype standby mode. The most peers are probed at the start of the application as we can see. Every peer is characterized by its ID so if a new peer is probed, then it gains his ID and corresponding dot is plotted to the graph. If some peer sends UDP packet to the *SC*, then corresponding ID is plotted on the negative y-axis. Lot of peers are probed repetitively. Very interesting is that sometimes the *SC* sends UDP packets repetitively (e.g. 700 packets per 50 hours) even to the peers that have never responded.

Perl modul Geo::IPfree was used to determine the originating countries of the probed peers, Figure 3.5. The modul uses a local file-based database to provide basic geolocation services. Countries were aggregated to continents if their country codes are not presented. Most contacted peers are located in the United States.

If UDP protocol is restricted, *SC* utilizes TCP protocol for TCP probes but not in bursts like in UDP case and in lesser scope. Positively probed supernode becomes *SC's NSN*. TCP packets are exchanged continuously between *SC* and *NSN* with average packet rate 1 per 10 seconds.

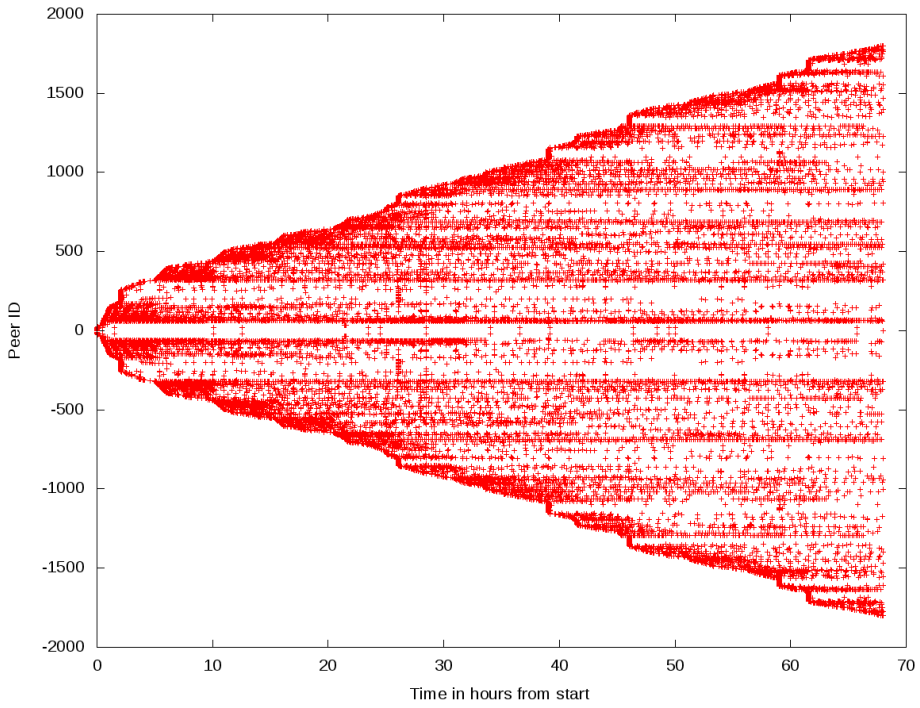


Figure 3.4: Each dot represents packet sent in a given time by *SC* to some other Skype peer whose corresponding ID is represented on the y-axis. Positive ID is for packet from *SC* to the peer, negative ID for packet to *SC*.

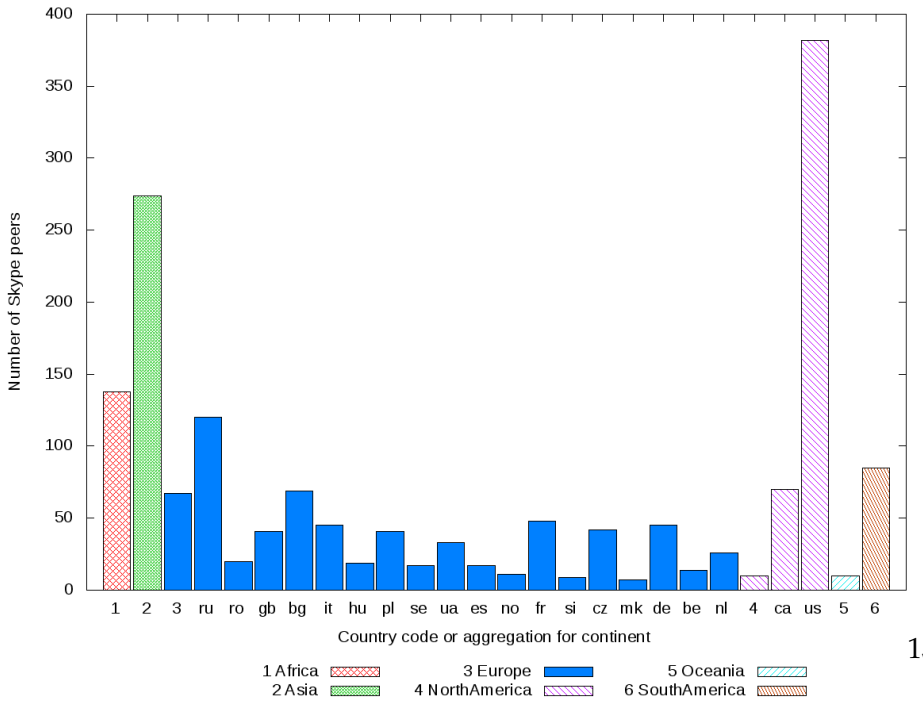


Figure 3.5: Country statistics for contacted peers.

	Sampling Rate (kHz)	Bit Rate (kbps)	CPU (MHz on x86 core)
Narrowband for PSTN gateways and low end devices	8	6 - 20	12 - 30
Mediumband for devices with limited wideband capability	12	7 - 25	16 - 40
Wideband for all-IP platforms	16	8 - 30	20 - 50
Super-Wideband, a new standard in speech quality	24	12 - 40	30 - 80
Key Advantage	Optimize clarity under hardware and network constraints	Adjust to degraded network conditions in real time	Match complexity to CPU resources in real time

Figure 3.6: The average network bit rate for corresponding audio bandwidths, figure from [19]

3.3.7 Going offline

When the client is switched to Offline status, every open TCP connection is finished. TCP connection with the *NSN* is always ended. After sign out from the Skype application, TCP connection with address like `78.141.181.242` or `213.146.188.16` is established. These connections occur sometimes after startup of the *SC* too. These addresses are associated with Skype Communications according to WHOIS database. I suppose that this connection can be somehow important for creating their own statistics about users. It is worthwhile to mention that I have outgoing ports 13392 and 12350 during all these connections observed.

3.3.8 Voice codec

SILK is a speech and audio codec developed internally at Skype which is used as the default codec for all Skype to Skype calls. It is highly scalable in terms of audio bandwidth, network bit rate, and complexity, making it the codec of choice for multiple modes and applications as described in [12, 8]. SILK is a replacement for the SVOPC codec [25] used firstly in Skype version 3.2. The SILK codec was a separate development branch from SVOPC and the final version was introduced in Skype version 4.0 (February 2009).

The SILK speech and audio codec is highly scalable in terms of audio bandwidth, network bit rate, and complexity. SILK supports four different audio bandwidths: 8000 Hz, 12000 Hz, 16000 Hz and 24000 Hz sampling frequency as shown in Figure 3.6.

Narrowband mode should only be used to interface to PSTN networks or on low end devices that do not support greater than 8000 Hz sampling frequency, mediumband mode for lower end devices that do not support greater than 12000 Hz sampling frequency or are under severe network bandwidth constrains (e.g. wireless devices). Wideband mode should be used for all-IP platforms that do not support greater than 16000 Hz sampling frequency. Super wideband mode should be used on all platforms that support 24000 Hz and greater sampling frequency.

The internal frame size of SILK is 20 ms. The SILK encoder can be set to join up to five internal frames into a single frame output. That means we can get 20, 40, 60, 80 or 100 ms frames of encoded speech or audio data. It is mentioned that SILK operates at a very low algorithmic delay, consisting of packetization delay, i.e. 20, 40, 60, 80 or 100 ms plus 5 ms lookahead delay.

The internal sampling frequency of the encoded speech or audio signal of SILK may change during the duration of a transmission. The average bit rate target can be adjusted on a per frame basis. This allows support for congestion control and network load management.

3.3.9 Call placement

The *SC* needs TCP connectivity for signalling information as described in [20]. It strongly prefers UDP connectivity for voice and video communication. If UDP is unavailable, it can utilize TCP for the media stream but with the additional overhead due to TCP being stateful. Before a user places their call, the client communicates with the peer network to test connectivity. It checks whether the outgoing UDP port is available and the type of address translation used by network. Status checking and updating is also carried out through P2P architecture to identify online status of our contacts. I have observed that if there appears status change, then *SC* is informed by 2 up to 4 TCP message exchange.

There are several possibilities how the voice or video communication is transfered through the network. If both caller (SC_A) and callee (SC_B) are on machines with public IP addresses, then upon pressing the Call button in the application interface, then UDP probing to different peers occurs, SC_A establishes a TCP connection to the SC_B and some signalling information is exchanged. During the voice or video call UDP packets are then exchanged, TCP connection is alive and is kept for sending signalling information.

IF SC_A is behind NAT, it is able often to traverse NAT and negotiate networking parameters (remote IP address and source port) through other nodes and then initiate direct UDP connection. If SC_A was not able to com-

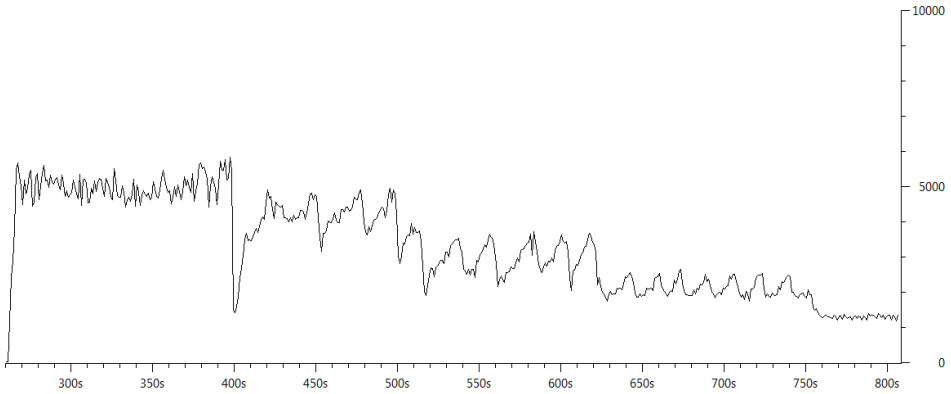


Figure 3.7: Bit rate during a voice call under decreasing available bandwidth every 120 seconds.

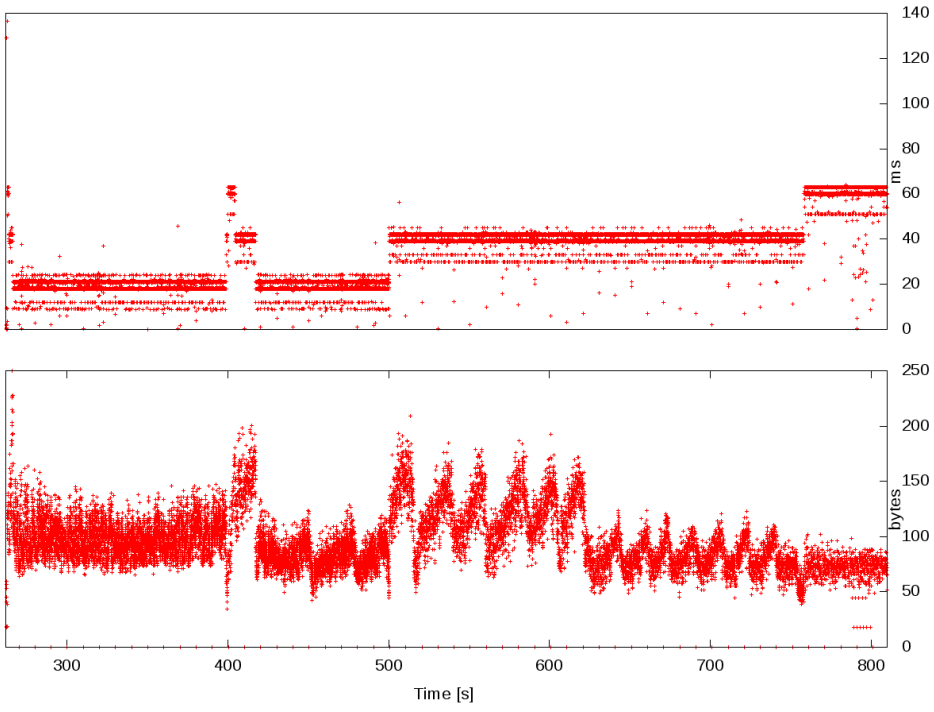


Figure 3.8: Inter packet gaps and bytes per packet during a voice call under decreasing available bandwidth every 120 seconds.

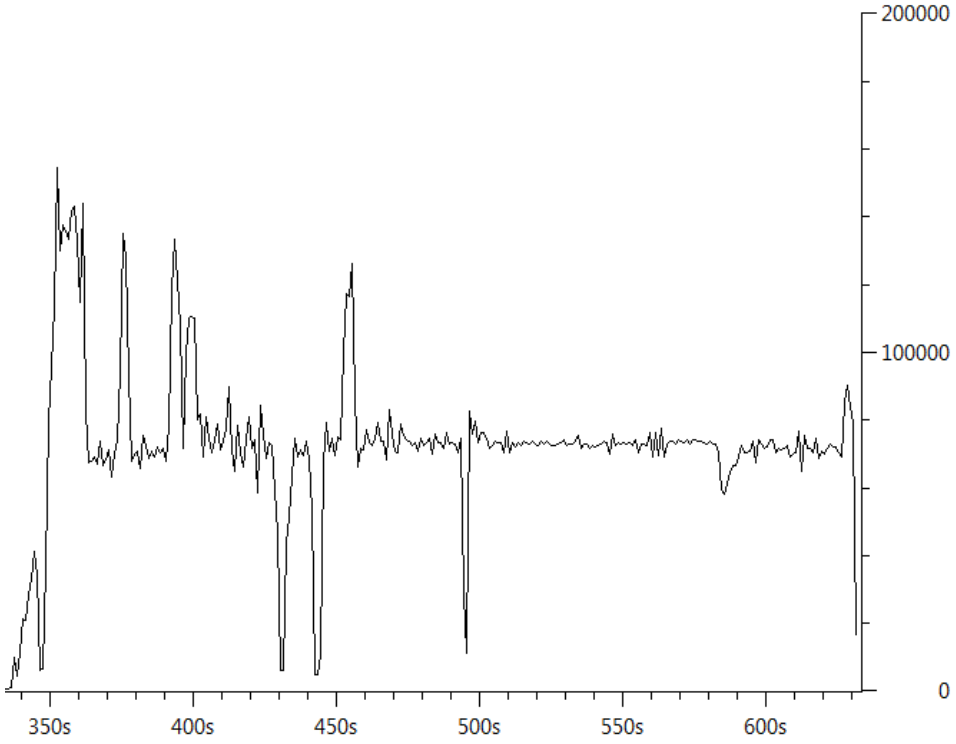


Figure 3.9: Bit rate during a video call.

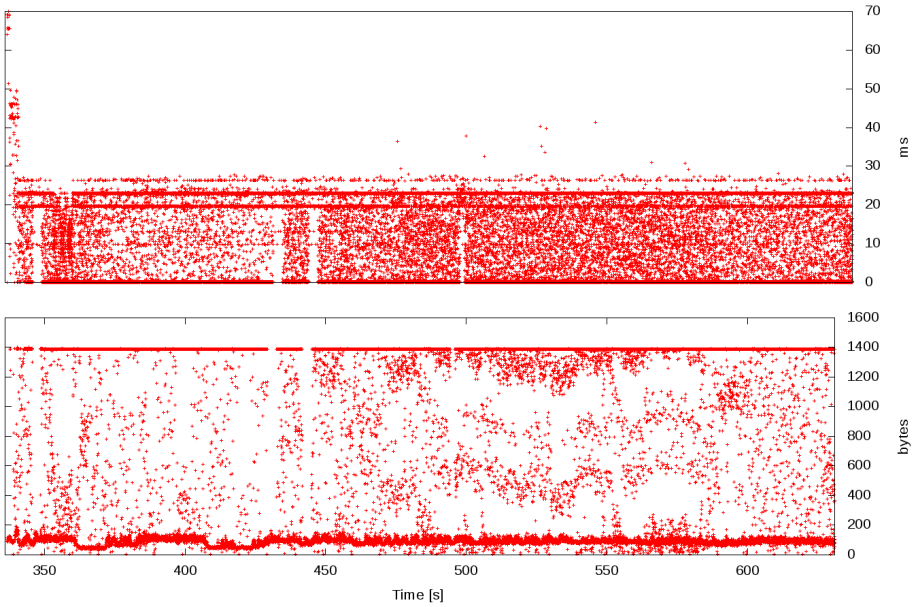


Figure 3.10: Inter packet gaps and bytes per packet during a video call.

municate directly, then it will find the appropriate *relays* for the connection. Relays are supernodes that relay media traffic and signalling information between clients that are not able to reach each other. SC_A and SC_B will then try to connect directly to the relays. For signalling and communication are then used several relays, probably for some fault tolerance or backup purposes. If SC_A is behind UDP restricted firewall, then for TCP connection relays are used too. Not only one relay is used but several relays as I have confirmed by testing. How would voice call properties change, I have placed a UDP call inside local network under decreasing available bandwidth, Figures 3.7 and 3.8. Every 120 seconds the available bandwidth was decreased. The bandwidth was unlimited for the first period, then it was changed to 4000, 3000, 2000 and 1000 bytes per second. We can see how 20, 40 and then 60 msec frames of encoded speech were used and how payload of frames was changed. The idea of possible voice call characteristic we can get from the fact that Skype since version 4 uses one particular voice codec for calls between Skype clients and that the parameters of the codec are available. On the other hand, this is not true in the case of video calls. Skype uses a TrueMotion VP7 video codec developed by On2 Technologies as described in [11]. The description of the codec is not available. Worth to mention is that Skype can operate on bare minimum bandwidth including video as low as 4 kbps [11]. I have placed a video call to visualize if there is particular pattern e.g. in inter packet time gaps. Figures 3.9 and 3.10 are showing a regular video call with 30 frames per second and resolution 320 x 240. As we can see, there is no clear distribution between inter packet time gaps. Dots around the value 20 shows us that not all voice packets are included in the video packets. Video packets are sent in bursts so the distribution around value 0. This can be also visible in the payload bytes per packet. Voice packets have payload around the value of 150 bytes whereas video payload around 1380 bytes. As described in [21], Skype can use from 100 kbps up to 1.5 Mbps in the case of HD video call (in group video calling up to 8 Mbps for download).

Chapter 4

NetFlow protocol

NetFlow is a network protocol developed by Cisco Systems for collecting IP traffic information. It has become an industry standard for network traffic monitoring and is widely used measurement solution today. NetFlow provides network administrators with acces to IP flow information from their data networks. A flow is defined as a unidirectional sequence of packets with some common properties that pass through a network device. Network elements (e.g. routers) export these collected flows to an external device – the NetFlow collector. Exported data is used for a variety of purposes, including ISP billing, network, user and application monitoring, capacity planning, security analysis.

During development of NetFlow there were several protocol versions presented. The first implementation was version v1 in 1996. It was restricted only to IPv4 and e.g. did not support IP masks. Versions v2, v3 and v4 were developed for internal CISCO purposes and have been never released. Version v5 came in 2009 and is the most common and is supported by different brands of network devices. Next versions till the version v9 are not widely used. So after v5 the next commonly used version on recent network devices is v9 [6]. IPv6 is supported and it uses templates to provide access to observations of IP packet flows in a flexible and extensible manner. A template defines a collection of fields, with corresponding descriptions of structure and semantics. The advantages are that it allows export of only required fields from the flows and that new fields can be added to the flow records without changing the structure of the export record format. Protocol IPFIX (Internet Protocol Flow Informatin eXport) becomes a successor to v9. It is IETF standardized NetFlow v9 with several extensions.

Netflow architecture consists of several NetFlow exporters. These devices monitor packets entering a particular location in the network and creates flows from these packets. The information from the flows is exported in the form of flow records to the NetFlow collector. Usually there is one collector which parses the incoming flow records and stores them. NetFlow monitoring using standalone NetFlow probes is an alternative to flow col-

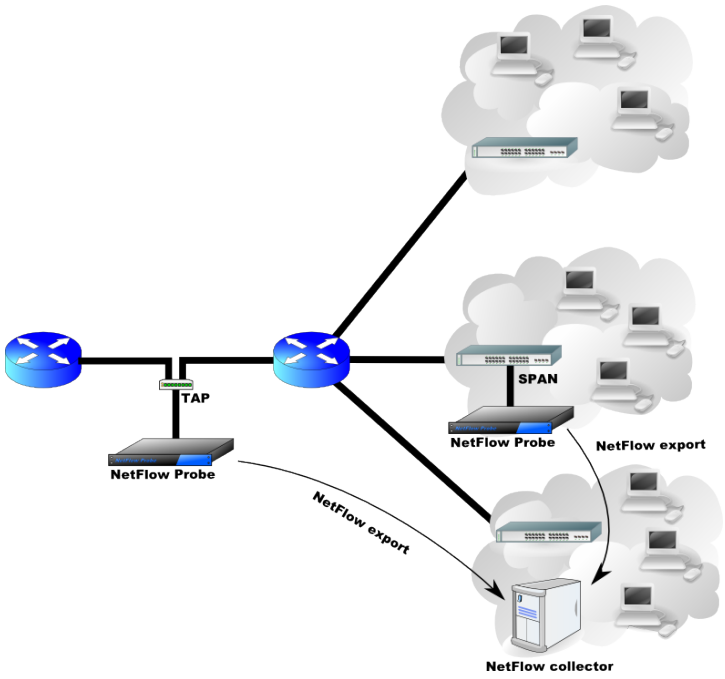


Figure 4.1: NetFlow architecture using standalone probes, figure from [23]

lection from routers or switches. This approach can overcome some limitations of router-based NetFlow monitoring, e.g. routers can be under heavy traffic load and must create flows simultaneously. The probes are transparently connected to the designated location and do not influence the packets passing this location as depicted in Figure 4.1.

The common properties that the sequence of packets in the flow share are according to the traditional Cisco definition following 7 values:

- Source IP address
- Destination IP address
- Source port
- Destination port
- IP protocol
- Input interface
- Type of Service

The exporter will output a flow record under several conditions:

- If the exporter cam detect the end of the flow, e.g. the `FIN` and `RST` bits in a TCP connection.
- If the flow has been *inactive*, i.e. no packets belonging to the flow have been observed for a certain period of time (*inactive timeout*).
- Long-lasting flows are exported on a regular basis. That means that flow records are exported continuously after a certain period of time (*active timeout*) for the capturing flow.

The exported record can contain a wide variety of information about the traffic in a given flow. NetFlow v5 record contains version number, sequence number, input and output interface, timestamps in milliseconds for the flow start and end, number of bytes and packets associated with a flow, source and destination IP addresses and ports, IP protocol, Type Of Service, for TCP flows the aggregation of TCP flags, routing information like IP address of the next-hop along the route to the destination and source and destination IP masks.

4.1 NfSen and NFDUMP

NfSen and NFDUMP are tools that are distributed under the BSD licence. The NFDUMP tools collect, process and store NetFlow data on the command line, NfSen is a graphical web based front end for the NFDUMP.

4.1.1 NFDUMP

NFDUMP contains several tools that support NetFlow protocol in versions v5, v7 and v9.

- **nfcapd** (NetFlow capture daemon) – captures the NetFlow records from the exporter and stores them into files. Automatically rotate files every 5 minutes. There is one nfcapd process running for each exporter.
- **nfdump** (NetFlow dump) – reads and displays the NetFlow data from the files stored by nfcapd. It can create statistics, make flow data aggregations or filter stored data.
- **nfprofile** (NetFlow profiler) – reads the NetFlow data and filters it according to the profiles and stores it into files for later use.

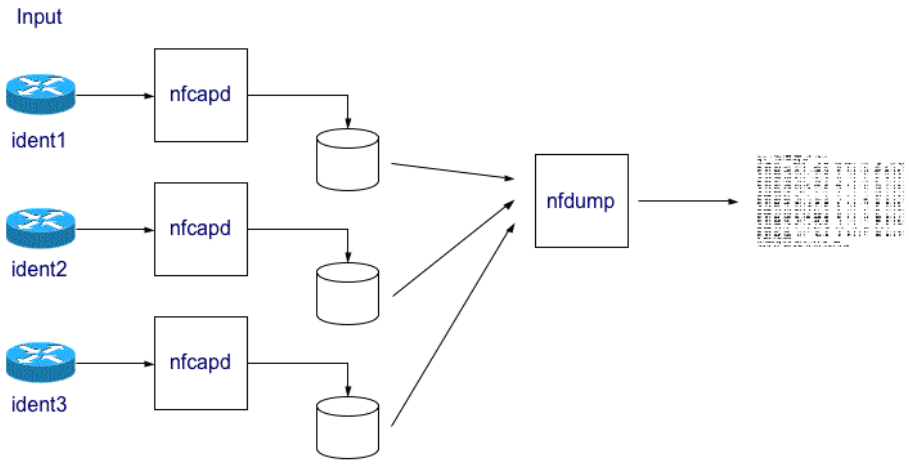


Figure 4.2: Capture daemons store NetFlow data that is read by nfdump, figure from [17]

- **nfreplay** (NetFlow replay) – reads the NetFlow data and sends it over the network to another host.

- **nfclean** (old data cleanup) and **ft2nfdump** (data convertor)

The goal of the design is to able to analyze netflow data from the past as well as to track interesting traffic patterns continuously. The amount of time back in the past is limited only by the disk space available for all the netflow data. The tools are optimized for speed for efficient filtering.

All data is stored to disk before analyzing. This separates the process of storing and analyzing the data (Figure 4.2). The data is stored in the *time slot* based fashion. Each file contains flow records captured during the configured 5 minutes time slot. The output file format corresponding to the time slot is `nfcapd.YYYYMMddhhmm`, e.g. `nfcapd.201104271330`. If there are several exporters from which the NetFlow data are stored, then the data is organized in the corresponding directories. One can choose its own subdirectory structure, but in the main configuration it is year, then month and day.

Flows can be read either from a single file or from a sequence of files as depicted in Figure 4.3.

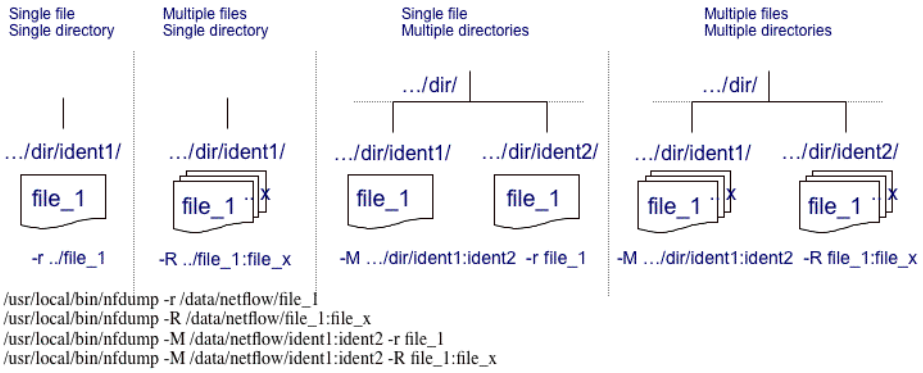


Figure 4.3: How the files are stored by nfcapd and read by nfdump, figure from [17]

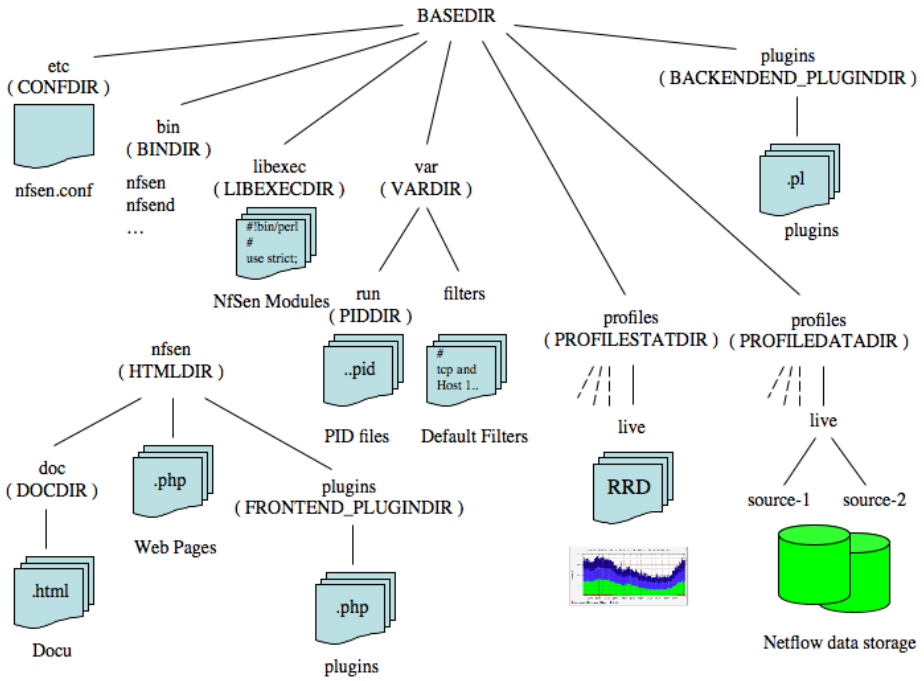


Figure 4.4: Directory structure, figure from [18]

4.1.2 NfSen

NfSen is graphical web based front end utilizes NFDUMP tools as mentioned before. It manages to display statistics about number of flows, packets and bytes in graphs using RRD (Round Robin Database) where graphs data is stored. As human need different point of views on the world around us, they need also different interfaces to the stored data. NfSen preserves benefits of the command line interface and brings concurrently graphical view. It allows easy navigation through stored NetFlow data, processing data from a single time slot or from a time window, creating history or continuous profiles to make specific view on the NetFlow data. NfSen also provides an option to execute specific actions (alerts) based on user defined conditions. One of the important functions is the option to extend NfSen with plugins. This affords us sufficient ways how to fit our additional needs and allows us to modify NfSen. Plugins may be selected from the NfSen navigation bar.

NfSen has a very flexible directory layout that means that the administrator can configure NfSen to fit his own needs. Default layout is shown in Figure 4.4. All NetFlow data is stored under `PROFILEDATADIR`. NfSen is reachable by web interface after the installation. But there is also possibility to use the command line interface if it is needed.

4.1.3 Plugins for NfSen

Plugins allow to add additional functionality in the area of statistical processing and output presentation. Plugins structure are divided into two types, namely *frontend plugins* and *backend plugins*. Backend plugins are intended to process stored NetFlow data or to prepare them for output or brings functionalities like alerting conditions and actions to the NfSen. They are run periodically or they can be run from the frontend plugin. Frontend plugins may display any kind of data resulting from backend plugin processing. The backend plugins are Perl modules. The frontend plugins are defined as PHP scripts with the same name as the backend plugin. Both plugins may exchange relevant data over nfsend.comm socket as shown in Figure 4.5. Over this communication channel any number of scalar and array values can be exchanged. Communication functions for the frontend plugin are defined in `nfsenutil.php` file and communication functions for the backend plugin are defined in the Perl module `Nfcomm.pm`. Both plugins must implement specific functions to be correctly integrated to NfSen.

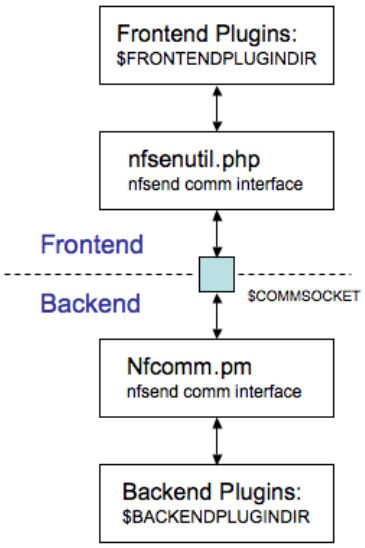


Figure 4.5: Communication concept for backend and frotend plugins, figure from [18]

Chapter 5

Creation of DetectSkype plugin for NfSen

As a result of my measurements I have created a plugin for NfSen. I have developed it for NfSen collector situated on `nftest.ics.muni.cz`. It processes flow records from 3 channels:

- 10GE connection between the Masaryk University network and CESNET
- 1GE from/to the Faculty od Informatics
- 1GE from/to the Vinařská dormitory

Real traffic is monitored but the data is anonymized. That means that IP addresses are anonymized but the characteristic of the flow records remains untouched.

The activity of the plugin can be divided into two ways. One is automatic and periodic processing of stored data and creation of statistics, the other one is processing user commands and presentation of results.

5.1 Backend plugin

Backend plugin is written in Perl language. The main part is by NfSen demanded function `run`. It is run periodically with relevant parameters (profile, profilegroup, timeslot) every 5 minutes by NfSen. Data processed by the plugin are stored in six database files. The database is simple Berkeley DB, which is easily usable in Perl environment.

The question was what is characteristic for Skype activity from the flow point of view. As Skype client initiates every startup a TCP connection to specific address, it was first rule that was considered for implementation. The *SC* sends an HTTP request to the HTTP server `ui.skype.com` to check the latest version of the application. This request is sent every four hour and under normal conditions (e.g. no TCP `RST` flag is present in the

flow) 5 TCP packets are sent in both directions. Filter for nfdump that will gain flow records representing this rule follows:

```
proto tcp and $srcnet and dst ip $uiskypeaddress
and dst port 80 and packets > 3
```

`$srcnet` is set to `src net 147.250.0.0/16` as the university addresses are anonymized to this mask. `$uiskypeaddress` is set to 203.233.225.202 as this is the actual anonymized address for `ui.skype.com`. Used in real network, it can be set to `ui.skype.com` because fully qualified hostnames are supported by nfdump.

We will maintain IP addresses that were in connection with Skype HTTP server for the last four hours and corresponding start time of the flow.

Next characteristic deals with voice call properties. As we can filter flow records in respect to bit rate and other useful fields, the filter for obtaining flows with voice characteristic will be:

```
proto udp and duration > 296000 and pps > 15
and pps < 70 and bps > 13000
and bps < 99000 and bpp < 300
```

As TCP voice calls use several connections to relays, I have focused on UDP calls. Since the characteristic of the voice call does not hold true for short periods of time like at the beginning and at the end of calls when e.g. signalling packets are sent after the end of call and influence calculated fields of the flow records, I have considered flow records that are spanned through the entire time slot – and that are 5 minutes long flow records. As the frames of sizes 20, 40, 60 msec are generated by SILK voice codec in the rate 50, 25, 16 frames per second, I filter records with packets per second value in this manner with some overhead to capture possible video call in higher pps rate. The restriction for bit rate is similar. Video call properties intersect the ones of voice call. So not to gain only isolated records of video calls sometimes, I have increased the tolerance for bit rate over the maximal value for voice bit rate. Bytes per packet are set according to my measurements. During processing of flows with given properties there is also check for the identical flow records. As outside trafic e.g. from CESNET to the Faculty of Informatics should be captured at two channels, there will be almost *identical* flow records for a given flow. They are eliminated by the check for almost identical start time field (threshold difference 3 seconds). Each flow with given properties is stored to the database. Every new flow in the next time slot is checked against these stored flows. If positive match is found i.e. source address, source port, destination address, destination port match, then these flows are merged together. If some flow is not merged with some new flow, then it is considered that the voice call has ended and

this flow is exported to the history database with additional values like UDP port (values are explained further in the frontend plugin section).

Next characteristic for the Skype client is the used UDP port. It is used as the source and incoming port for the UDP protocol. It is set in the Connection dialog of the client so it is not dynamically changed during the run of application. I have searched for this port in two situations. The first one is the moment when the connection to the HTTP server is detected. Then in the next time slot, the IP addresses are searched for the corresponding UDP ports. Skype initiates UDP probes during the whole Skype session and, from the flow point of view, the flow consists of one packet because in the future generated probes to the same peer will belong to the new flow. The filter is:

```
   proto udp and src port > 1024 and bpp < 80
and bpp > 40 and packets = 1
and src ip in $filter_addresses
```

Bytes per packet are limited thanks to the packet length statistics I have gained from my Skype standby measurements. I have also tested the proposed plugin implementation and when filtering processed data I have recognized BitTorrent activity. I have done several measurements to find out how similar BitTorrent UDP probing activity is to the one of Skype. Most of the UDP probe packets have sizes greater than 90 bytes. For UDP packets of Skype standby measurement (from 67 hours lasting log) I can see following packets statistics:

$Packetlengths$	$Count$	$Percent$
$40-79$	8472	52.98%
$80-159$	522	3.26%
$160-319$	6835	42.75%
$320-639$	161	1.01%

So I have limited bytes per packet according to these values to get better results. As there will be lot flows, flows are aggregated by IP source address and port and the aggregated flows are sorted according to aggregation number. Record with most flows is chosen as the representative and corresponding source port is selected as possible UDP port of the Skype application as concurrently with detected Skype activity there must be UDP probes present.

Second situation, as mentioned before, when the UDP port is searched. With the start of potential voice call it is time to check for the UDP port as at the beginning of voice call there is a burst of UDP probes always.

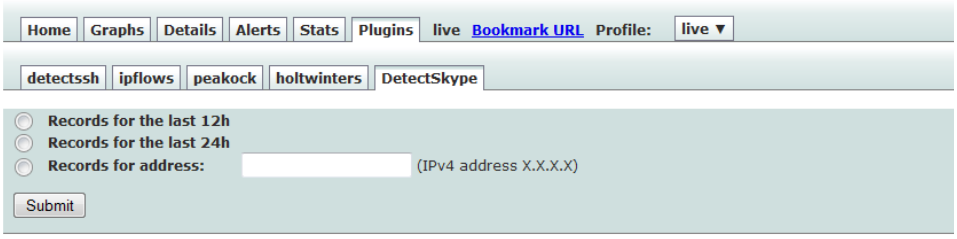


Figure 5.1: DetectSkype plugin frontend.

As we check UDP ports at two different times, we increase the probability that the port does not belong to some other network activity at a particular IP address.

As voice calls are bidirectional flows, I wanted to consider this characteristic too. I have find out that sometimes there are missing flow records on the collector side. To avoid dividing some continuously monitored potential call I have implemented another approach. The plugin calculates number of incoming flows to our designated IP address and port at the start of the potential call. Characteristic of the flows allows us to detect possible video calls with high bit rates. This characteristic is used because outgoing voice call can be detected correctly but in the opposite direction huge video traffic will not be recognized as Skype flow. This could lead to not recognizing our flow as Skype traffic as there would be missing bidirectional pattern.

```
  proto udp and duration > 296000 and pps > 10
and pps < 210 and bps > 8000
and bps < 2000000 and $filter_addr_port
```

Aggregation on destination address and port is used and records are arranged according to number of flows.

5.2 Frontend plugin

Frontend plugin is a component of the NfSen web interface. It implements two mandatory functions `DetectSkype_ParseInput` and `DetectSkype_Run`. `DetectSkype_ParseInput` is intended to to parse possible form data and is called after selection in the plugins tab. `DetectSkype_Run` is the main function, it is up to it what will be displayed in the web browser and parameters will be sent to the backend plugin.

There are several options what scope of values to display in the web

browser. User can choose to view stored records for the last 12 or 24 hours. If selected, then a table with corresponding statistics for records is displayed.

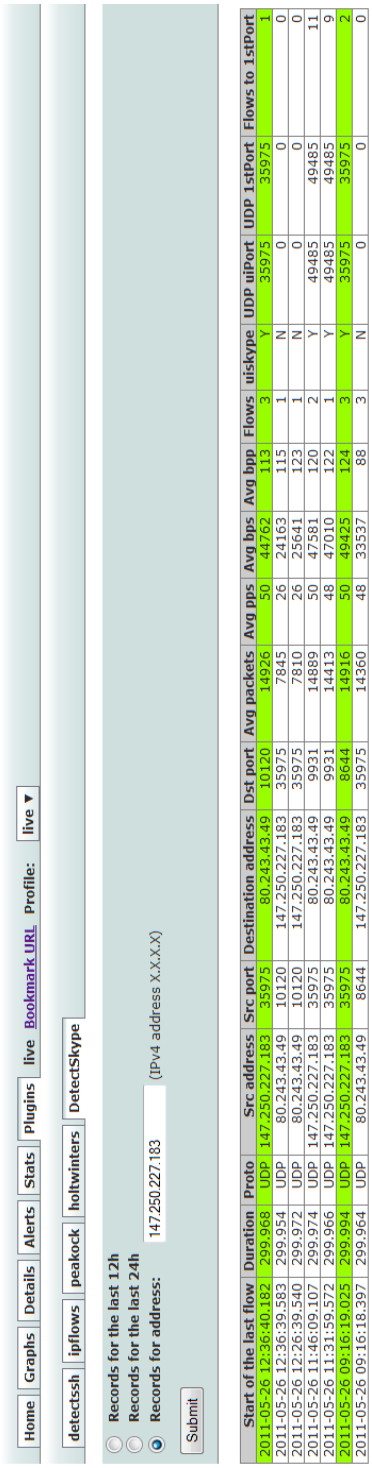
- **Start of the last flow** – Start time of the last record from the continuously captured flow records.
- **Duration** – Duration of the last captured flow record.
- **Src address**
- **Src port**
- **Destination address**
- **Dst port**
- **Avg packets** – Average number of packets calculated from the merged records.
- **Avg pps** – Average packets per second value.
- **Avg bps** – Average bits per second statistics.
- **Avg bpp** – Average bytes per packet statistics.
- **Flows** – Number of continuously captured and after that merged flows.
- **uiskype** – "Y" if connection from Src address to ui.skype.com captured.
- **UDP uiPort** – UDP port that was detected around the time of uiskype activity.
- **UDP 1stPort** – UDP port that was detected around the time the first flow record was captured.
- **Flows to 1stPort** – Number of flows with previously defined characteristic to the UDP 1stPort.

If uiskype equals "Y" and the source port equals to UDP uiPort and UDP 1stport, then the record is marked as green as the potential Skype voice call. Column Flows to 1stport gives us information if there exists bidirectional characteristic. If the value is 0, then bidirectional pattern is missing. Then this flow is not Skype voice call (or there was missing corresponding record from the channels). If the value is between 1 up to 6 then it is most probably a voice call. This value up to six is given by the fact that the flows are

Records for the last 12h
Records for the last 24h
Records for address: (IPv4 address X.X.X.X)
Submit

Start of the last flow	Duration	Proto	Src address	Src port	Destination address	Dst port	Avg packets	Avg pps	Avg bps	Avg bpp	Flows	uiskype	UDP uiPort	UDP 1stPort	Flows to 1stPort
2011-05-27 08:34:46.535	299.984	UDP	61.169.82.175	41320	147.250.73.122	59258	15000	50	80004	200	1	N	0	0	0
2011-05-27 08:34:46.513	299.999	UDP	147.250.73.122	59258	61.169.82.175	41320	14979	49	79888	200	1	N	0	0	0
2011-05-27 08:33:14.097	299.953	UDP	84.123.165.208	4390	147.250.228.222	39478	14318	47	22443	58	1	N	0	0	0
2011-05-27 08:31:38.627	299.809	UDP	101.111.5.145	12316	147.250.95.25	32925	12370	41	17081	51	2	N	0	0	0
2011-05-27 08:27:06.560	299.971	UDP	76.209.107.78	33929	147.250.82.3	16065	15833	52	41587	98	1	N	0	0	0
2011-05-27 08:26:59.919	299.967	UDP	60.251.52.51	4367	147.250.215.92	4314	6459	21	20880	121	1	N	0	0	0
2011-05-27 08:24:46.528	299.983	UDP	61.169.82.175	41320	147.250.73.122	59258	15000	50	80004	200	1	N	0	0	0
2011-05-27 08:24:46.487	299.987	UDP	147.250.73.122	59258	61.169.82.175	41320	14495	48	77305	199	1	N	0	0	0
2011-05-27 08:20:32.918	299.961	UDP	147.250.74.81	60937	61.169.82.11	41656	15001	50	80007	200	5	N	0	0	0
2011-05-27 08:20:32.913	299.983	UDP	61.169.82.11	41656	147.250.74.81	60937	15000	50	80005	200	5	N	0	0	0
2011-05-27 08:17:06.538	299.953	UDP	76.209.107.78	33929	147.250.82.3	16065	14895	49	40137	101	1	N	0	0	0
2011-05-27 08:16:53.522	298.256	UDP	147.250.236.90	58089	83.56.188.53	30361	16058	53	22399	52	1	N	0	58089	251
2011-05-27 08:15:53.312	299.971	UDP	147.250.236.90	58089	91.92.216.180	53093	19994	66	27539	51	1	N	0	58089	251
2011-05-27 08:15:25.543	299.960	UDP	147.250.236.90	58089	88.237.207.49	57944	11880	39	16319	51	1	N	0	58089	251
2011-05-27 08:11:37.671	299.969	UDP	83.159.192.185	20410	147.250.95.25	52270	8628	28	21345	92	1	N	0	0	0
2011-05-27 08:08:08.515	299.949	UDP	147.250.236.90	58089	85.69.235.36	55034	14206	47	19739	52	1	N	0	58089	349
2011-05-27 08:07:53.740	299.985	UDP	61.169.82.175	24816	147.250.87.247	27726	15000	50	80004	200	5	N	0	0	0
2011-05-27 08:07:53.740	299.988	UDP	147.250.87.247	27726	61.169.82.175	24816	15000	50	80000	200	5	N	0	51216	0
2011-05-27 08:07:02.767	299.992	UDP	95.147.223.66	44576	147.250.95.25	32925	12429	41	17258	52	3	N	0	0	0
2011-05-27 08:06:52.499	297.229	UDP	147.250.236.90	58089	87.57.159.193	57014	13564	45	19000	52	1	N	0	58089	349
2011-05-27 08:06:02.361	296.382	UDP	147.250.236.90	58089	73.156.170.163	20877	19590	66	27153	51	1	N	0	58089	349
2011-05-27 08:06:01.961	299.557	UDP	13.151.139.169	17482	147.250.5.84	57190	15483	51	79889	193	2	N	0	0	0
2011-05-27 08:05:08.718	299.968	UDP	147.250.73.40	44866	86.78.160.41	16014	12064	40	16354	51	2	Y	0	0	0
2011-05-27 07:49:55.692	299.704	UDP	147.250.236.90	58089	150.105.157.48	44714	16644	55	23220	52	1	N	0	58089	58
2011-05-27 07:42:58.295	299.958	UDP	79.216.23.185	57882	147.250.87.99	55195	14999	50	80005	200	1	N	0	0	0
2011-05-27 07:42:58.267	299.997	UDP	147.250.87.99	55195	79.216.23.185	57882	15001	50	80001	199	1	N	0	0	0
2011-05-27 07:40:42.730	297.211	UDP	147.250.193.58	57337	85.177.66.28	44811	10604	35	34126	119	1	Y	57337	57337	3
2011-05-27 07:38:47.710	299.953	UDP	147.250.193.58	57337	37.220.46.115	19388	14736	49	75264	191	2	Y	57337	57337	2
2011-05-27 07:32:50.800	299.979	UDP	81.238.105.141	27841	147.250.95.25	32925	11704	39	16141	51	1	N	0	0	0

Figure 5.2: DetectSkype plugin frontend, first rows of statistics for the the last 24 hours.



Start of the last flow	Duration	Proto	Src address	Src port	Destination address	Dst port	Avg packets	Avg pps	Avg bps	Avg bpp	Flows	uiskype	UDP uiPort	UDP 1stPort	Flows to 1stPort
2011-05-26 12:36:40.182	299.968	UDP	147.250.227.183	35975	80.243.43.49	10120	14926	50	44762	113	3	Y	35975	35975	1
2011-05-26 12:36:39.583	299.954	UDP	80.243.43.49	10120	147.250.227.183	35975	7845	26	24163	115	1	N	0	0	0
2011-05-26 12:26:39.540	299.972	UDP	80.243.43.49	10120	147.250.227.183	35975	7810	26	25641	123	1	N	0	0	0
2011-05-26 11:46:09.107	299.974	UDP	147.250.227.183	35975	80.243.43.49	9931	14889	50	47581	120	2	Y	49485	49485	11
2011-05-26 11:31:59.572	299.966	UDP	147.250.227.183	35975	80.243.43.49	9931	14413	48	47010	122	1	Y	49485	49485	9
2011-05-26 09:16:19.025	299.994	UDP	147.250.227.183	35975	80.243.43.49	8644	14916	50	49425	124	3	Y	35975	35975	2
2011-05-26 09:16:18.397	299.964	UDP	80.243.43.49	8644	147.250.227.183	35975	14360	48	33537	88	3	N	0	0	0

Figure 5.3: DetectSkype plugin frontend, statistics for one IP address.

counted for more then one time slot and if the flow record is captured at by two channels at the same time, we can get value up to six for continuous voice call. Values higher are most probably not voice calls. But when Skype client is at conference call, then more incoming flows can be detected. So there is third option for displaying data statistics, namely for a particular IP address. Records for this option are stored for two weeks and records in which given address figures as source address or destination address are displayed. In Figure 5.3, there is visible how BitTorrent was filterd out by better choosing parameters for searching UDP ports. The most upper row shows us a voice call that has lasted for three time slots. That means that the start of the voice call was detected approximately at `12:26:40`. At that time we can observe flow has bidirectional character to our voice call. Here we can also see that the flow from address 80.243.43.49 has missing flow record for the start time at `12:31:39` but the flow was present. This also clearly visible by the fact that two displayed flows have start time as multiplication of 5 minutes.

Chapter 6

Conclusion

I have studied network traffic of the Skype application in this thesis. I have confirmed some detected traffic or payload patterns presented by others for earlier versions of Skype client. My contribution was verification of proposed papers related to this field for newer versions of application – 4.1, 4.2 and 5.3 – and discovering new facts that were not published so far as I know. I have proposed e.g. new facts about TCP signalling traffic. From the presented facts it is possible to create data payload analyzer that will recognize Skype traffic quite efficient as similarly mentioned in [1]. There are several other techniques how to reveal Skype traffic. One method based on Pearson's Chi Square test is used to detect Skype's fingerprints from the packet framing structure, exploiting the randomness introduced at the bit level by the encryption process [5]. There can also be used some machine learning algorithms like AdaBoost or C4.5 [3].

Further in this thesis I have created plugin for NfSen with its backend and frontend part. As NetFlow records do not offer data payload for inspection, approach different to payload analysis was used. Limitations are that it is quite difficult to recognize TCP relayed traffic as several connections with no constant traffic and with dynamical source ports is present, so it was not possible to consider this kind of detection. Limitation for proposed plugin is that we do not have calculation for the probability the Skype voice call was recognized correctly.

Bibliography

[1] Davide Adami, Christian Callegari, Stefano Giordano, Michele Pagano, and Teresa Pepe. A Real-Time Algorithm for Skype Traffic Detection and Classification. In Sergey Balandin, Dmitri Moltchanov, and Yevgeni Koucheryavy, editors, *Smart Spaces and Next Generation Wired/Wireless Networking, 9th International Conference, NEW2AN 2009 and Second Conference on Smart Spaces, ruSMART 2009, St. Petersburg, Russia, September 15-18, 2009. Proceedings*, volume 5764 of *Lecture Notes in Computer Science*, pages 168–179. Springer, 2009.

[2] C. Allen and T. Dierks. RFC 2246 – The TLS Protocol Version 1.0. `http://tools.ietf.org/html/rfc2246`, January 1999.

[3] Duffy Angevine and A. Nur Zincir-Heywood. A preliminary investigation of skype traffic classification using a minimalist feature set. In *ARES*, pages 1075–1079. IEEE Computer Society, 2008.

[4] Salman Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *INFOCOM*. IEEE, 2006.

[5] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. In Jun Murai and Kenjiro Cho, editors, *Proceedings of the ACM SIGCOMM 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, August 27-31, 2007*, pages 37–48. ACM, 2007.

[6] B. Claise. RFC 3954 – Cisco Systems NetFlow Services Export Version 9. `http://tools.ietf.org/html/rfc3954`, October 2004.

[7] Sven Ehlert and Sandrine Petgang. Analysis and Signature of Skype VoIP Session Traffic. In *Franunhofer FOKUS Technical Report NGNISKYPE-06b*, July 2006. Berlin, Germany.

[8] H. Astrom, J. Spittka and K. Vos. RTP Payload Format and File Storage Format for SILK Speech and Audio Codec. `http://developer.`

skype.com/resources/SILK_RTP_PayloadFormat.pdf, August 2010.

[9] J. Rosenberg, R. Mahy, P. Matthews and D. Wing. RFC 5389 – Session Traversal Utilities for NAT (STUN). http://tools.ietf.org/html/rfc5389, October 2008.

[10] Chun-Ming Leung and Yuen-Yan Chan. Network Forensic on Encrypted Peer-to-Peer VoIP Traffics and the Detection, Blocking, and Prioritization of Skype Traffics. In *WETICE*, pages 401–408. IEEE Computer Society, 2007.

[11] Jonathan Rosenberg. Skype and the Network, Technical Advisory Process Workshop on Broadband Network Management. http://www.openinternet.gov/workshops/docs/ws_tech_advisory_process/Skype-FCC.PPTX. Visited 27. 5. 2011.

[12] S. Jensen, K. Soerensen and K. Vos. SILK Speech Codec draft-vos-silk-01. http://developer.skype.com/resources/draft-vos-silk-01.txt, March 2010.

[13] Andreas Thomann. Skype - A Baltic Success Story. https://infocus.credit-suisse.com/app/article/index.cfm?fuseaction=OpenArticle&aoid=163167&coid=7805&lang=EN, September 2006. Visited 21. 2. 2011.

[14] Skype – The Big Blog – 30 million people online on Skype. http://blogs.skype.com/en/2011/03/30_million_people_online.html. Visited 27. 4. 2011.

[15] Skype grows FY revenues 20%, reaches 663 mln users. http://www.telecompaper.com/news/skype-grows-fy-revenues-20-reaches-663-mln-users, March 2011. Visited 27. 4. 2011.

[16] IT Administrators guide. http://download.skype.com/share/business/guides/skype-it-administrators-guide.pdf. Visited 21. 2. 2011.

[17] NFDUMP. http://nfdump.sourceforge.net. Visited 25. 5. 2011.

[18] NfSen – Netflow Sensor. http://nfsen.sourceforge.net. Visited 25. 5. 2011.

[19] SILK Data Sheet. http://developer.skype.com/resources/SILKDataSheet.pdf. Visited 21. 2. 2011.

[20] Free Skype calls and cheap calls to phones – Skype. http://www.skype.com. Visited 21. 2. 2011.

[21] Help for Skype: How much bandwidth does Skype need? https://support.skype.com/en/faq/FA1417/How-much-bandwidth-does-Skype-need. Visited 7. 4. 2011.

[22] Kazaa – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Kazaa. Visited 21. 2. 2011.

[23] NetFlow – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Netflow. Visited 20. 5. 2011.

[24] Skype – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Skype. Visited 12. 5. 2011.

[25] SVOPC – Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/SVOPC. Visited 20. 5. 2011.