VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

A TOOL FOR VOIP AUDIO EXTRACTION

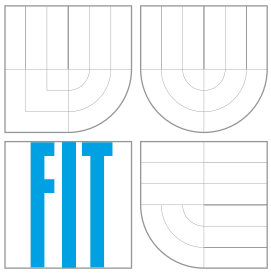
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB RUŽIČKA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁSTROJ PRO ZÍSKÁVÁNÍ HLASOVÝCH DAT VOIP

A TOOL FOR VOIP AUDIO EXTRACTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB RUŽIČKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2012

Abstrakt

Cílem práce je vytvořit systém, který dokáže rekonstruovat audio data z VoIP komunikace. Systém rozpozná v záznamu síťového provozu proudy VoIP paketů a na základě jejich obsahu sestaví přenášený audio signál. Kromě rozšířeného RTP protokolu je podporován také IAX protokol používaný Asterisk ústřednou, který nabízí zajímavé možnosti a není plně či vůbec podporován dostupnými nástroji. Systém je implementován jako knihovna s minimálním rozhraním.

Abstract

In this thesis, we describe VoIP protocols and design of a system to reconstruct audio data from VoIP communication. The system is able to detect VoIP packet streams in an IP network traffic and assemble an audio signal they carry. RTP and IAX VoIP protocols are supported. Unlike widespread RTP protocol, IAX is not fully supported by available tools although it is used by increasingly popular Asterisk communications project and offers interesting features not found in RTP. The system is implemented as a library with minimal frontend.

Klíčová slova

získávání VoIP audia, analýza síťových proudů, IAX, RTP

Keywords

VoIP audio extraction, network stream analysis, IAX, RTP

Citace

Jakub Ružička: A Tool for VoIP Audio Extraction, bakalářská práce, Brno, FIT VUT v Brně, 2012

A Tool for VoIP Audio Extraction

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Matouška, Ph.D.

.....
Jakub Ružička
May 13, 2012

Poděkování

Rád bych poděkoval autorům svobodného a otevřeného software, díky kterému mohla tato práce vzniknout, panu Petru Kopeckému za možnost projekt zveřejnit a doktorovi Petru Matouškovi za inspirativní vedení této práce.

Contents

Chapter 1

Introduction

In this chapter, motivation for this thesis is explained, objectives of the project are stated, available solutions to the problem are presented and contribution of this work is discussed.

1.1 Motivation

Motivation for a VoIP audio extraction tool arose with development of a complex IP network monitoring and analysis system developed by LinuxBox.cz[1]. This system called Foxstat can decode IP traffic on network, transport and application layers, identify individual data streams and analyze their protocol specific content. For VoIP protocols, call quality metrics like packet loss or network jitter are interesting but the ability to reconstruct the actual voice data enables wide range of quality assessment options. Listening to the extracted audio is the simplest way to subjectively evaluate the call quality but there are also automated methods like Perceptual Evaluation of Speech Quality (PESQ, ITU-T recommendation P.862), its successor Perceptual Objective Listening Quality Assessment (POLQA, ITU-T recommendation P.863) or alternative solutions like Audio Quality Analyzer (AQuA) from Sevana[2].

1.2 Objectives

The objective of the project is to design and implement a system for VoIP stream detection and audio reconstruction as a library with minimal command line frontend running on linux based operating system. The system provides:

- A convenient command line tool for VoIP audio extraction.
- An interface for external applications.

The ability to listen to the transmitted audio provided by the command line tool enables manual subjective audio quality evaluation while automated quality evaluation systems working on audio signal level may be built on top of the provided interface.

The system supports RTP [4] protocol used extensively in VoIP telephony and IAX [6] protocol used by increasingly popular Asterisk open source communications project[3].

[1] `http://linuxbox.cz`
[2] `http://www.sevana.fi/voice_quality_testing_measurement_analysis.php`
[3] `http://www.asterisk.org/asterisk`

Unlike widespread RTP protocol, IAX is not fully supported by available tools even though it offers interesting features not found in RTP. To the best of our knowledge, no existing software is able to reliably reconstruct audio from IAX and provide convenient interface on target platform. The system must be able to:

- Read packets from network traffic record files.
- Decode IP, UDP, RTP and IAX packets.
- Detect individual RTP/IAX streams in IP traffic.
- Reconstruct the audio data carried by VoIP packets.
- Write the reconstructed audio data into files.

1.3 Available solutions

There are several tools available that can extract VoIP payload. We introduce here the most promising ones which we evaluated. The requirements are as follows:

- Reconstruct audio data carried by RTP protocol [4].
- Reconstruct audio data carried by IAX protocol [6].
- Provides interface that allows usage from other applications.
- Runs on linux.
- Open source is strongly preferred.

For more information about RTP and IAX protocols see chapter 2.

1.3.1 Application `rtpbreak`

`rtpbreak`[4] is a free and open source command line utility for RTP payload extraction by Michele Dallachiesa released under GNU General Public License version 2.

It can read packets either from live network interface or from a pcap file and save raw RTP payloads into files. It does not require presence of RTCP packets and works independently of a signaling protocol (SIP, H.323, etc.).

`rtpbreak` is a great tool with beautiful source code, but handles RTP protocol only, there is no IAX support.

1.3.2 Application Wireshark

Wireshark[5] is a well known network sniffing and analysis tool that can decode huge number of protocols and includes a powerful GUI. It is free and open source under GNU General Public License and runs on most operating systems.

In addition to ability to display a structure of binary RTP and IAX packets, Wireshark can also perform a Stream Analysis on both protocols that displays a list of packets belonging to a VoIP stream with relevant information about each packet and overall stream

[4] `http://dallachiesa.com/code/rtpbreak/`
[5] `http://www.wireshark.org`

metrics. A sample RTP Stream Analysis window is shown in Figure 1.1. There is an option to save a stream payload to a file, which is exactly what we need. However, after investigation we found two problems. First, unlike GUI independent packet decoders (dissectors) which can be used both from main Wireshark GUI and command line tool `tshark`, Stream Analysis code is tightly integrated with its interactive GUI and it does not provide command line or any other interface so it can not be used from external programs. Second, IAX Stream Analysis is unable to process IAX trunk packets, which are essential part of the protocol and one of the main advantages of IAX over RTP.

These two problems are still present in Wireshark version 1.6.3 making it unsuitable for the expected task. Fixing them would require considerable amount of work. However, we did write patches to properly decode IAX trunk packets which were committed into upstream Wireshark. See Appendix B for more information.

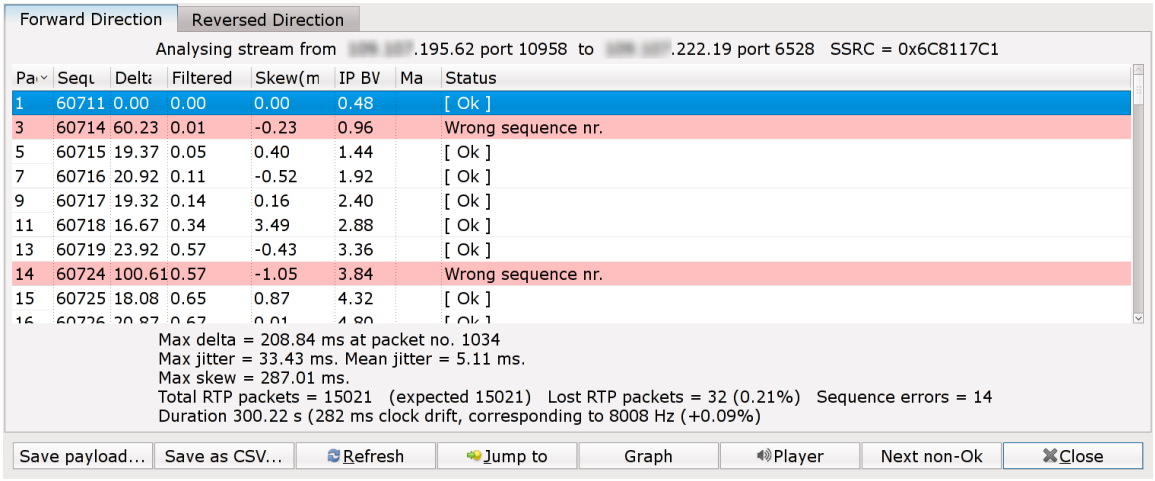


Figure 1.1: Wireshark RTP Stream Analysis

1.3.3 Application Cain and Abel

Cain and Abel[6] is a freeware password recovery and network sniffing tool for Microsoft Windows. Although it does not run on target platform and it is not open source, we include it here for the sake of completeness as it is powerful and widely used software.

Cain and Abel version 4.9.43 is able to detect RTP stream and save its payload to a file, but it does not detect IAX streams at all. VoIP analysis window is shown in Figure 1.2.

[6] `http://www.oxid.it/cain.html`

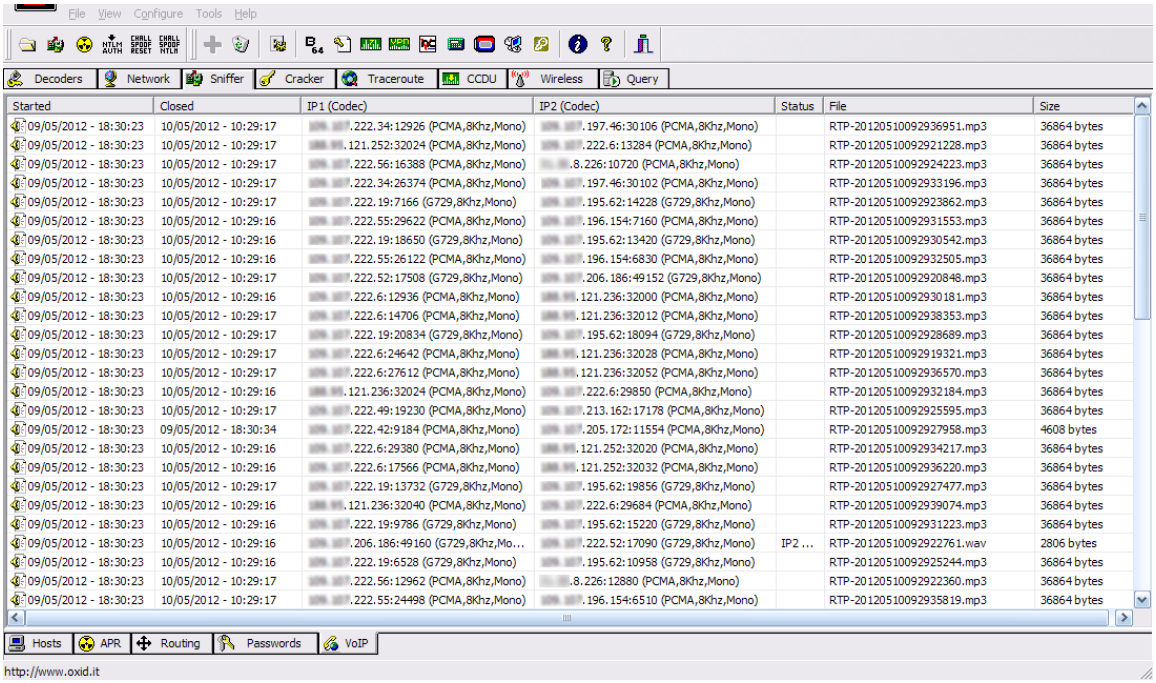


Figure 1.2: Cain and Abel VoIP analysis

1.4 Contribution

Although there are several tools available to reconstruct audio data from the widespread RTP protocol, none of them fully supports increasingly popular IAX protocol that has many advantages described in section 2.4. Our system can reconstruct audio data carried by both IAX and RTP and provides convenient command line tool as well as a simple interface that enables development of applications for VoIP audio quality evaluation.

Chapter 2

VoIP protocols

In this chapter, principles and operation of Voice over Internet Protocol (VoIP) are explained including introduction to Internet Protocol Suite. Usage and packet formats of Real-time Transport Protocol (RTP) and Inter-Asterisk eXchange protocol (IAX) are described.

2.1 Overview

Voice over Internet Protocol (VoIP) is a family of technologies for transmitting voice communication over a packet-switched Internet Protocol (IP) networks. Unlike traditional circuit-switched Public Switched Telephone Networks (PSTN), there is no dedicated communications channel between VoIP nodes, voice data are sent in small chunks called packets. Each packet can travel a different route through an IP network and arrive with variable delay or not at all. VoIP protocols provide formats and rules for originating and signaling VoIP sessions (calls), transcoding, packetization and transmission of voice and media data over IP networks.

2.1.1 Internet Protocol Suite

The Internet Protocol Suite is a set of communication protocols used on the Internet and similar networks described in RFC 1122 [1]. It is also known as TCP/IP because Internet Protocol (IP) and Transmission Control Protocol (TCP) were the first protocols in the suite which remain most important although many other protocols were added. The suite is structured into four abstract layers:

- *Application Layer* is a top layer of the Internet Protocol Suite in which applications create and communicate user data to other applications. VoIP protocols like RTP, IAX, SIP[2] and H.245[5] are application layer protocols.
- *Transport Layer* provides end-to-end communication services for applications. Reliability, flow control, congestion control, byte orientation, multiplexing and connection-oriented communication services may be provided on transport layer. There are two primary transport layer protocols. Transmission Control Protocol (TCP) is a reliable connection oriented transport service that provides end-to-end reliability, resequencing, and flow control [1]. TCP is used by signaling and control VoIP protocols such as SIP and H.245. User Datagram Protocol (UDP) is a connectionless transport service

for sending messages (datagrams) between hosts. It does not provide mechanisms for reliability, ordering, or data integrity so datagrams may arrive out-of-order, duplicated or not at all. UDP is used in real-time data transmission VoIP protocols like RTP and IAX because they are sensitive to delays and dropped packets are preferred to delayed packets.

- *Internet Layer* provides principal Internet Protocol (IP) used for transmitting packets (datagrams) across interconnected networks, such as the Internet. The task of IP is to deliver packets from one host to another based on their IP addresses, routing the packets through network nodes and across network boundaries. It includes provision for addressing, type-of-service specification, fragmentation and reassembly, and security information but there are no delivery guarantees (they are provided on Transport Layer).
- *Link Layer* or media-access layer is a bottom layer of the Internet Protocol Suite that contains protocols for communication between adjacent network nodes using physical and logical network components. This layer is of little interest from VoIP perspective.

VoIP protocols work on the application layer as seen in Figure 2.1.

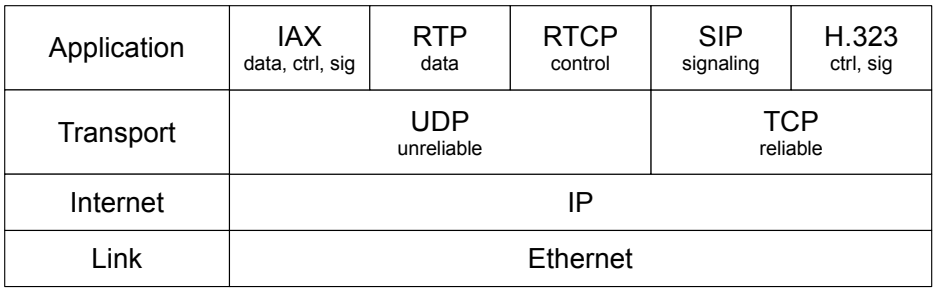


Figure 2.1: VoIP protocols in TCP/IP model

2.1.2 VoIP operation

First, a session consisting of one or more audio streams must be initiated. Each audio stream transmits an audio signal from a sending host to one or more receiving hosts. A typical VoIP call between host A and host B is a session of two streams, a stream carrying audio signal from A to B and an opposite direction stream carrying audio signal from B to A. Originating, modifying and terminating VoIP sessions is called signaling.

After session initiation, audio signal is transmitted in streams between desired hosts of IP network. On sending host, the audio signal is split into small (usually 20 ms) frames which are encoded to selected audio format, encapsulated into VoIP packets and sent separately over an IP network. On target host, the audio data are extracted and decoded from received VoIP packets and reconstructed into original audio signal. The process of audio signal packetization is illustrated in Figure 2.2.

Transmission of audio signal in packets over IP network introduces several negative effects:

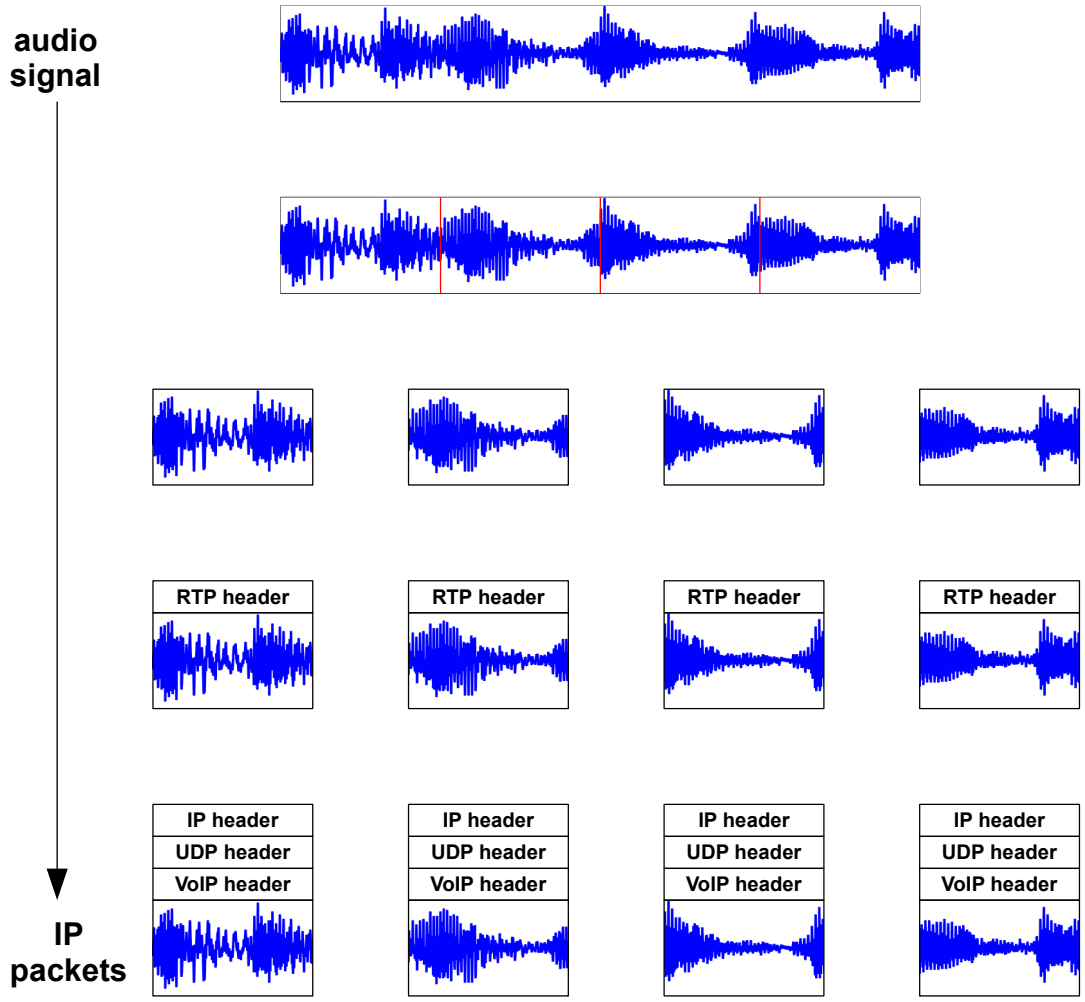


Figure 2.2: Audio signal packetization

- *Delay* before a packet reaches its destination (also called network latency) depends on large number of factors from connection quality to network load. Large delay drastically degrades VoIP quality as speakers have to pause during call which breaks the natural conversation.
- *Jitter* is a deviation from mean delivery delay caused by variable network latency. Although packets are sent with constant period, they arrive with variable delay. With large jitter, packets may even arrive out-of-order.
- *Packet loss* may occur on unreliable lines, during network congestion and many other occasions.

To compensate jitter, packet loss and too late packets, a jitter buffer on receiving side is required. Incoming packets are inserted into jitter buffer according to their sequence and timestamp information so that audio signal is correctly reconstructed even with out-of-order and late packets. However, jitter buffer introduces additional delay so its size must be chosen carefully to balance jitter tolerance with call delay.

2.2 RTP - Real-time Transport Protocol

Real-time Transport Protocol (RTP) is a protocol defined in RFC 3550 [4] which *provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video. These services include payload type identification, sequence numbering, timestamping and delivery monitoring* [4]. A RTP stream is accompanied by a separate RTP Control Protocol (RTCP) stream which provides statistics and basic control information about a RTP flow. RTP can be unicast to a single recipient or multicast to multiple recipients (i.e. audio/video conference).

2.2.1 RTP usage

On IP networks, RTP usually runs on top of the UDP protocol on even UDP port with RTCP on following odd port. RTCP control packets are sent periodically to all participants in the session and perform these functions:

- Primarily provide feedback on the quality of RTP data distribution.
- Carry a persistent canonical name to keep track of each participant.
- Control the rate of RTCP packets in order to scale up to a large number of participants.
- Optionally convey minimal session control information.

RTCP may not support all the control communication requirements of an application, in such case a higher-level session control protocol such as SIP[2] may be used.

RTP stream may be modified between end systems by intermediate RTP systems called "mixers" and "translators" which may be designed for various purposes. For example, RTP mixer can be used to transcode audio streams into lower-bandwidth encoding for transmission over a low-speed link.

2.2.2 RTP packet format

RTP features a simple packet header format shown in Figure 2.3.

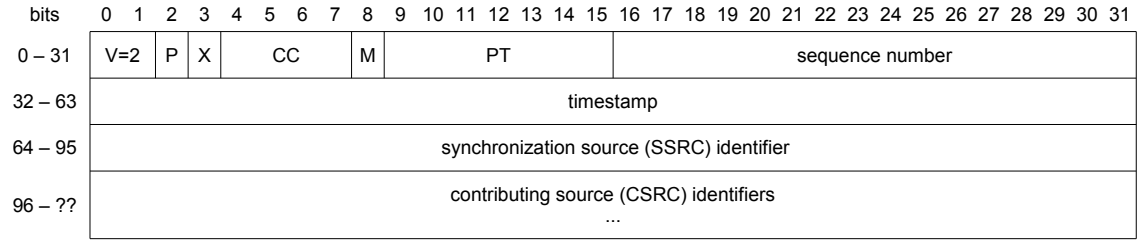


Figure 2.3: RTP packet header

First twelve octets are present in every RTP packet. Following 0 to 15 CSRC identifiers are present only when inserted by mixers.

Short description of fields follows, for full description see RFC 3550 [4].

- **version** (V): 2 bits
  Version of RTP, current version is 2.

- **padding** (P): 1 bit
  If the padding bit is set, the packet contains one or more additional padding octets at the end which are not part of the payload.

- **extension** (X): 1 bit
  If the extension bit is set, the fixed header must be followed by exactly one header extension.

- **CSRC count** (CC): 4 bits
  The CSRC count contains the number of CSRC identifiers that follow the fixed header.

- **marker** (M): 1 bit
  The interpretation of the marker is defined by a RTP profile.

- **payload type** (PT): 7 bits
  This field identifies the format of the RTP payload and determines its interpretation by the application.

- **sequence number**: 16 bits
  The sequence number increments by one for each RTP data packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.

- **timestamp**: 32 bits
  The timestamp reflects the format dependent sampling instant of the first octet in the RTP data packet. It allows jitter calculations and proper stream reconstruction.

- **SSRC**: 32 bits
  The SSRC field identifies the synchronization source. No two synchronization sources within the same RTP session will have the same SSRC identifier.

- **CSRC list**: 0 to 15 items, 32 bits each
  The CSRC list identifies the contributing sources for the payload contained in this packet. The number of identifiers is given by the CC field.

2.3 IAX - Inter-Asterisk eXchange

Inter-Asterisk eXchange protocol (IAX) is *an application-layer control and media protocol for creating, modifying, and terminating multimedia sessions over IP networks. IAX was developed by the open source community for the Asterisk PBX*[1] *and is targeted primarily at VoIP call control, but it can be used with streaming video or any other type of multimedia.*

IAX is an "all in one" protocol for handling multimedia in IP networks. It combines both control and media services in the same protocol. In addition, IAX uses a single UDP data stream on a static port greatly simplifying Network Address Translation (NAT) gateway traversal, eliminating the need for other protocols to work around NAT, and simplifying network and firewall management. IAX employs a compact encoding that decreases bandwidth usage and is well suited for Internet telephony service [6].

IAX is an open binary protocol described in Informational RFC 5456 [6] designed to reduce overhead, especially in regards to voice streams.

[1] `http://www.asterisk.org/asterisk`

2.3.1 IAX protocol usage

IAX includes media and control functions, no additional signaling protocol is needed. Signaling and media streams between two hosts are multiplexed over a single UDP stream, usually on UDP port 4569.
IAX call origination is explained in Figure 2.4 and Figure 2.5 shows a complete IAX media exchange. Note that voice mini frames are always preceded by a voice full frame.

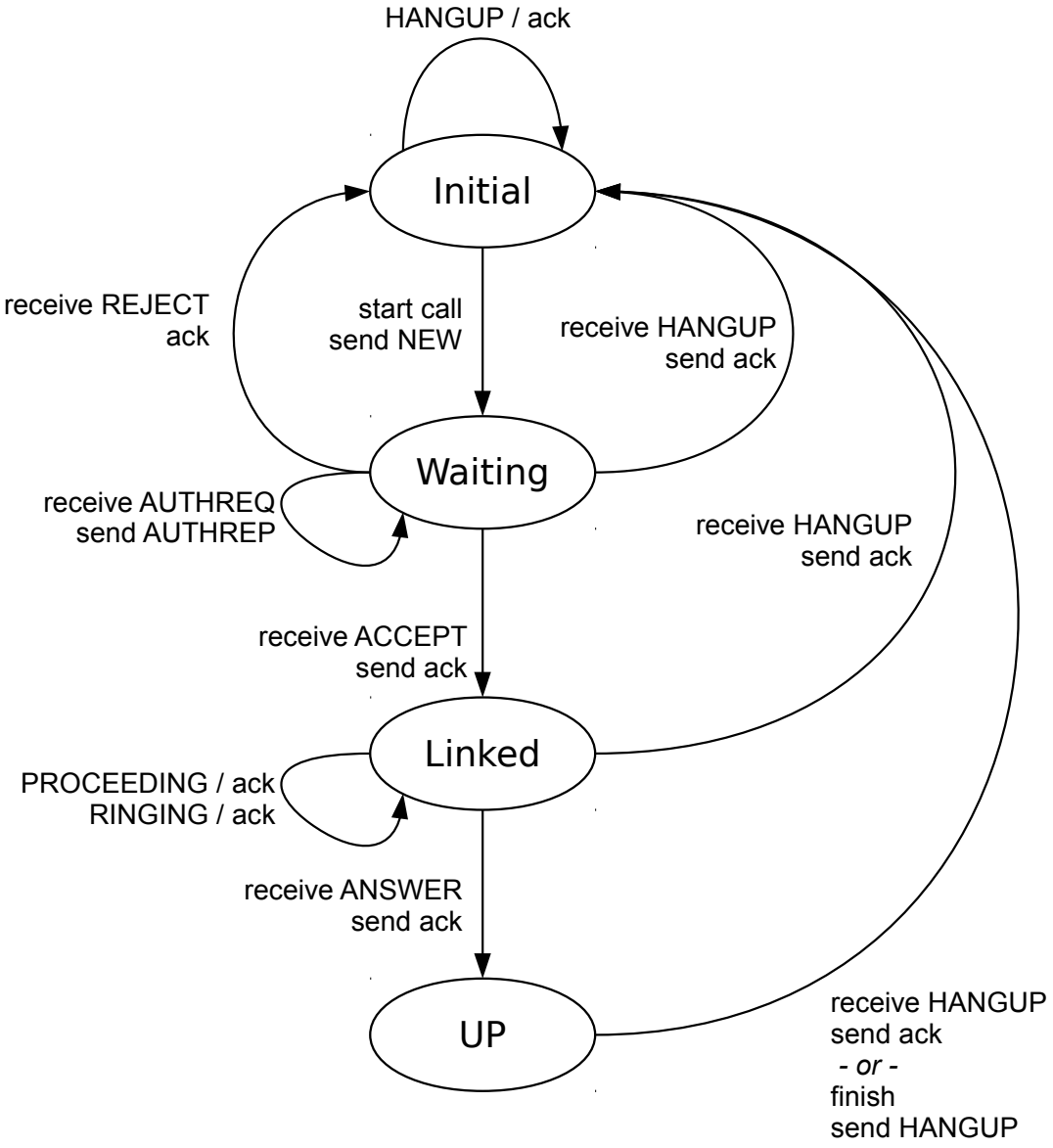


Figure 2.4: IAX call origination state diagram

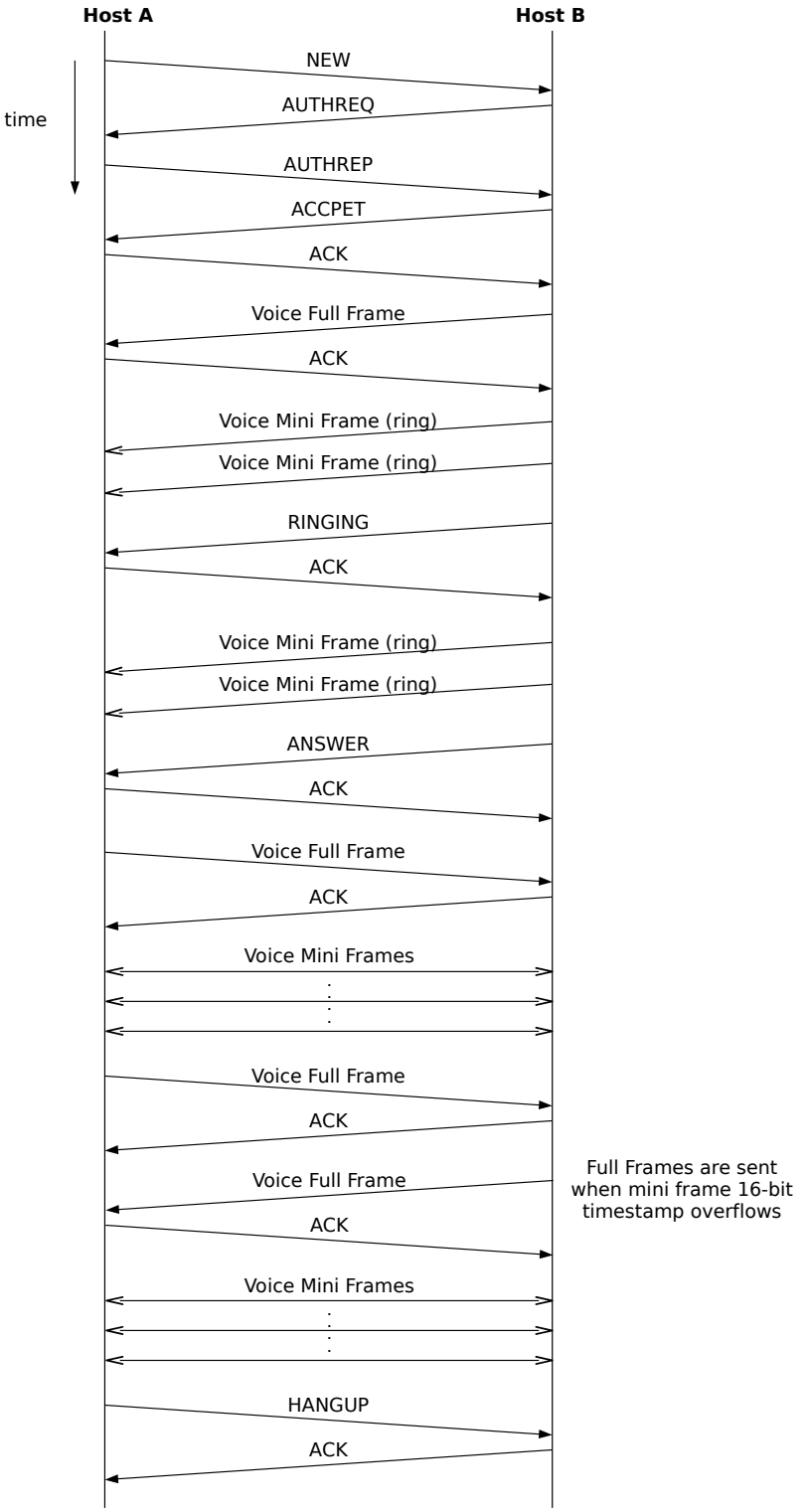


Figure 2.5: Complete IAX end-to-end media exchange

2.3.2 IAX packet format

IAX defines several packet (frame) types and formats:

IAX Full Frame

Full Frames can send signaling or media data. Generally, Full Frames are used to control initiation, setup, and termination of an IAX call, but they can also be used to carry stream data (though this is generally not optimal) [6].

Full frames are sent at the beginning of a session and also to resync the 32-bit timestamp when the 16-bit timestamp (used by mini frames) overflows.

Format of Full Frame is illustrated in Figure 2.6.

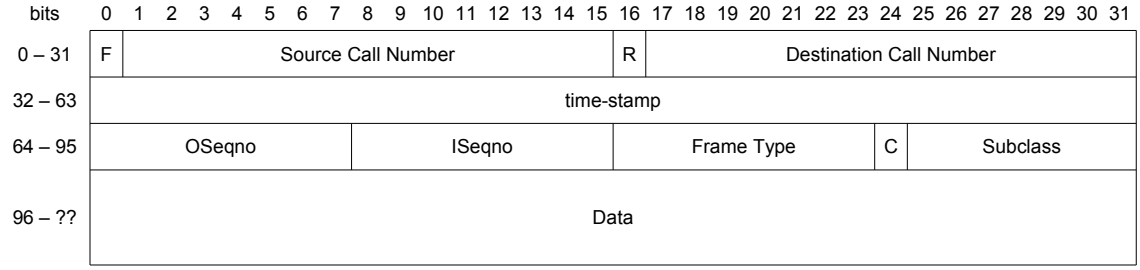


Figure 2.6: IAX Full Frame format

The standard Full Frame header length is 12 octets.
Short description of fields follows, for full description see RFC 5456 [6].

- **'F' bit**:
  Specifies whether or not the frame is a Full Frame. 1 means Full Frame.
- **Source call number**: 15 bits
  Used by transmitting host to identify the current call. It can not be used by another call on the same client at the same time and thus uniquely identifies the call on the local host.
- **'R' bit**:
  Specifies whether or not the frame is being retransmitted.
- **Destination call number**: 15 bits
  Used by transmitting host to reference the call at the remote host. The destination call number equals the remote host's source call number and uniquely identifies a call on the remote host.
- **Time-stamp**: 32 bits
  Time-stamp is an incrementally increasing representation of the number of milliseconds since the first transmission of the call.
- **OSeqno**: 8 bits
  Outbound stream sequence number starting at 0 upon initialization of a call and increased by 1 with each Full Frame sent. Silently overflows to 0.

- **ISeqno**: 8 bits
  Inbound stream sequence number starting at 0 upon initialization of a call and increased by 1 with each Full Frame received. It represents the next expected inbound stream sequence number. Silently overflows to 0.

- **Frame type**: 8 bits
  Identifies the type of a message carried by Full Frame.

- **'C' bit**:
  Determines the coding of remaining 7 bits of Subclass field are coded. 1 means the Subclass value is interpreted as a power of 2, otherwise it's interpreted as a simple 7-bit unsigned integer.

- **Subclass**: 7 bits
  Interpretation of this field depends on frame type.

Frame type determines the interpretation of carried data and the subclass field. Following 10 frame types are defined (shortened from RFC 5456[6]):

- **Control Frame** carries session control data, i.e., it refers to control of a device connected to an IAX endpoint. The subclass describes the device control signal.

- **IAX Frame** carries control data that provides IAX protocol-specific endpoint management. This frame type is used to manage IAX protocol interactions that are generally independent of the type of endpoints. The subclass describes an IAX event.

- **Voice Frame** carries voice data. The subclass specifies the predefined audio format of the data.

- **Video Frame** carries video data. The subclass specifies the predefined video format of the data.

- **DTMF Frame** carries a single digit of DTMF[2]. The subclass is the actual DTMF digit carried by the frame.

- **Text Frame** carries a non-control text message in UTF-8 format. Subclass is set to 0.

- **HTML Frame** carries HTML data. The subclass contains HTML Frame specific code.

- **Image Frame** carries a single image. The subclass describes the format of the image.

- **Comfort Noise Frame** The frame carries comfort noise. The subclass is the level of comfort noise in -dBov.

- **Null Frame** must not be transmitted.

[2]Dual Tone Multi-Frequency is an analog signaling method working in voice-frequency band.

IAX Mini Frame

Mini Frames carry no control or signaling data; their sole purpose is to carry a media stream on an already-established IAX call [6].

Mini Frame is implicitly defined to be of type "voice frame". The format of carried data is defined by the format of the most recent full voice frame.

Mini frames carry only a lower 16 bits of transmitting host's full 32-bit timestamp. A full voice frame must be sent before transmission of mini frames and full frame is also used to resync the 32-bit timestamp when the 16-bit timestamp overflows.

Format of Mini Frame is illustrated in Figure 2.7.

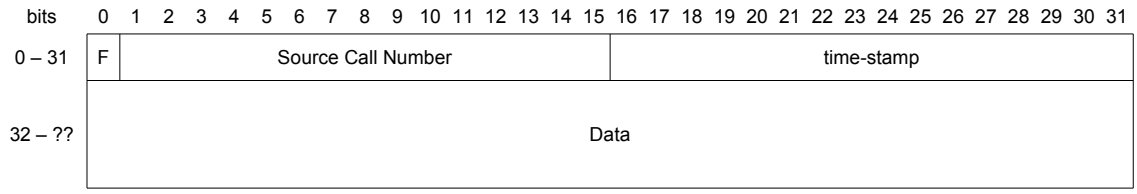


Figure 2.7: IAX Mini Frame format

Mini Frame features a minimal 4 octets header:

- **'F' bit**:
  Set to 0 to to specify it is not a Full Frame.
- **Source call number**: 15 bits
  Used by transmitting host to identify the current call.
- **Time-stamp**: 16 bits
  Lower 16 bits of the transmitting host's full 32-bit time-stamp for the current call.

IAX Trunk Frame

Meta frames serve one of two purposes. Meta trunk frames are used for trunking multiple IAX media streams between two peers into one header, to further minimize bandwidth consumption. Meta video frames allow the transmission of video streams with an optimized header. They are similar in purpose to mini voice frames [6].

IAX Trunk Frame is an interesting IAX feature that allows trunking multiple media streams between two hosts into a single association. There are two methods of trunking:

- *Without trunk time-stamps*: Each trunk media frame header contains only call number and data length. 32-bit time-stamp of trunk meta frame is used.
- *With trunk time-stamps*: Each trunk media frame has a full mini frame header including 16-bit time-stamp. This increases the header size by 2 octets but provides more accurate timing information.

Meta Trunk Frame header format is illustrated in Figure 2.8
Short description of fields follows:

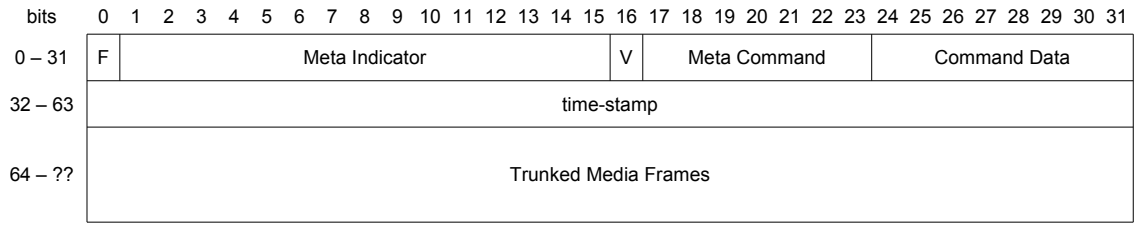


Figure 2.8: IAX Meta Trunk Frame format

- **'F' bit**:
  Set to 0 to to specify the frame is not a Full Frame.
- **Meta Indicator**: 15 bits
  All zeroes to indicate Meta Frame. Meta Frames are identifiable because the first 16 bits are always zero.
- **'V' bit**:
  Set to 0 to specify that the frame is not a meta video frame.
- **Meta Command**: 7 bits
  Set to '1' to indicate Meta Trunk Frame. All other values are reserved for future use.
- **Command Data**: 8 bits
  Options that apply to a trunk call. The least significant bit is the 'trunk time-stamps' flag indicating the type of trunk media frames. Other bits are reserved for future use.
- **time-stamp**: 32 bits
  Represents the actual time of transmission of the trunk frame.

Figure 2.9 illustrates IAX Trunk Frame without timestamps and Figure 2.10 illustrates IAX Trunk Frame with timestamps.

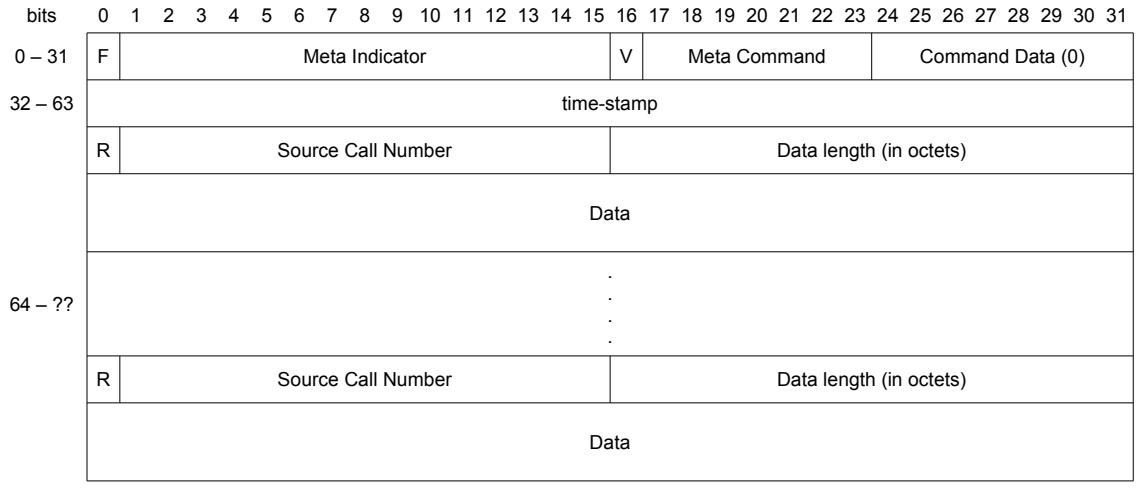


Figure 2.9: IAX Meta Trunk Frame without timestamp

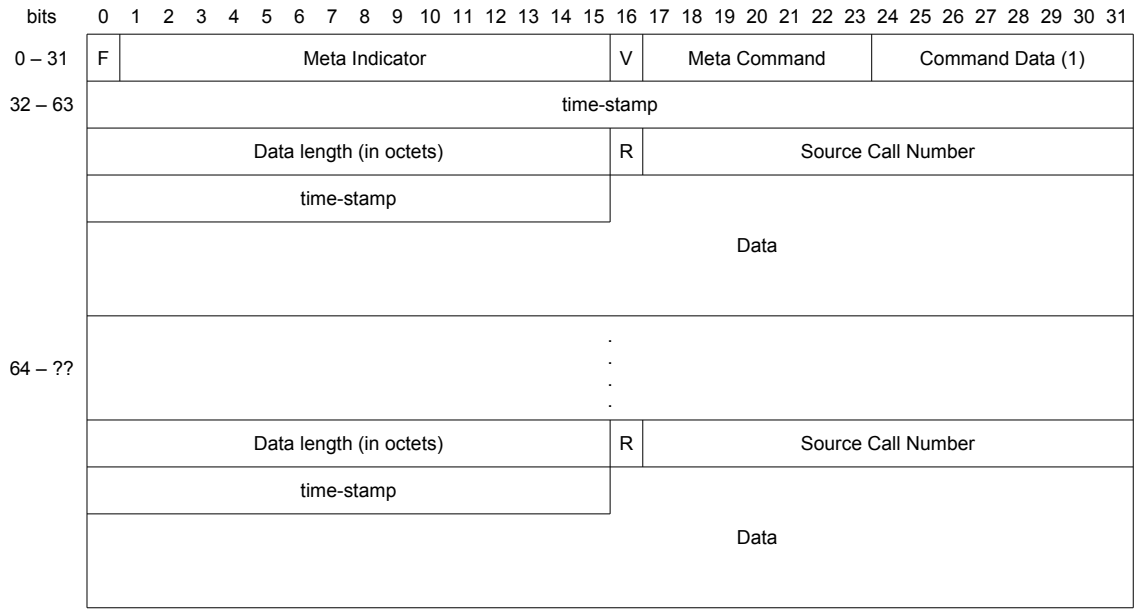


Figure 2.10: IAX Meta Trunk Frame with timestamp

2.4 RTP and IAX comparison

RTP is a straightforward protocol for end-to-end data delivery with minimal control functionality provided by RTCP, additional signaling protocol such as SIP is needed in most scenarios. A simple packet format makes it easy to implement a software and hardware working with RTP such as streaming servers, VoIP end devices, RTP mixers, traffic analysis systems etc.

IAX is a full-fledged all-in-one VoIP solution created for Asterisk open source communications project. Although IAX is best suited for communication between VoIP servers, it can be efficiently used to communicate with end devices as well. IAX provides several advantages over RTP:

- IAX trunk frames allow transmission of multiple calls in one UDP/IP packet decreasing bandwidth usage, especially between servers with many streams.
- Compact IAX mini frames have only 32-bit headers in contrast with 96-bit RTP, saving 8 bytes per packet.
- Usage of a single UDP port for both signaling and media makes IAX traffic easy to shape, prioritize, and pass through firewalls.
- No additional signaling protocol is needed.

On the other hand, IAX has many different packet formats described above, which make it harder to parse and analyze.

The essential differences between RTP and IAX are summarized in Table 2.1.

	RTP	**IAX**
packet format	simple	complex
signaling	minimal (RTCP)	full-fledged
SW support	very good	poor
trunk packets	no	yes
required UDP ports	2 (+ signaling)	1
port numbers	dynamic	4569
timestamp units	samples (codec dependent)	ms (codec independent)
minimal header size	32 bits	96 bits

Table 2.1: Summary of differences between RTP and IAX

Chapter 3

Detection and Reconstruction of VoIP audio

In this chapter, methods of VoIP stream detection and audio reconstruction are explained. Audio formats used in VoIP communications are briefly introduced.

3.1 VoIP stream detection

First of all, it is necessary to recognize RTP and IAX packets in IP traffic. IAX usually uses a registered 4569 UDP port, so if an IP packet has either source or destination port number 4569, it is likely to be an IAX packet. That is not the case with RTP which may use any UDP port making stream auto detection difficult. There are two basic approaches to identifying VoIP streams:

- *Manual*: Explicit rules to identify VoIP streams are given. Packet filters matching against IP and UDP headers (for example BPF[1]) can be used. It is possible to describe even VoIP calls composed of multiple streams that way. This approach produces consistent results but it is not always desirable, especially when the required information about streams is difficult to obtain or not available.
- *Automatic*: The software attempts to identify any VoIP streams in IP traffic. RTP and IAX are binary protocols and it is not possible to reliably recognize them in UDP packet. Usually, IAX streams use 4569 UDP port, but it can be changed. For RTP, it might possible to decode signaling protocol (i.e. SIP) if it is used. Alternatively, heuristic approach can be used - RTP usually runs on even UDP port, RTP header can contain only certain combinations of bits and a size of payload depends on payload type. Although this approach may work under certain conditions, it is not universal and may produce false positives.

To classify a packet into a VoIP stream, IP source and destination address, UDP source and destination port, and protocol specific source identification must be obtained from packet headers:

- In RTP, the source identifier is Synchronization source, SSRC. It is a 32-bit numeric value uniquely identifying a source of a stream of RTP packets within a particular RTP session.

[1] Berkeley Packet Filter

- For IAX, the source identifier is a 15-bit source call number, which corresponds to destination call number for a remote host as illustrated in Figure 3.1. IAX full frames carry both source and destination call number, mini and trunk frames carry source call number only.

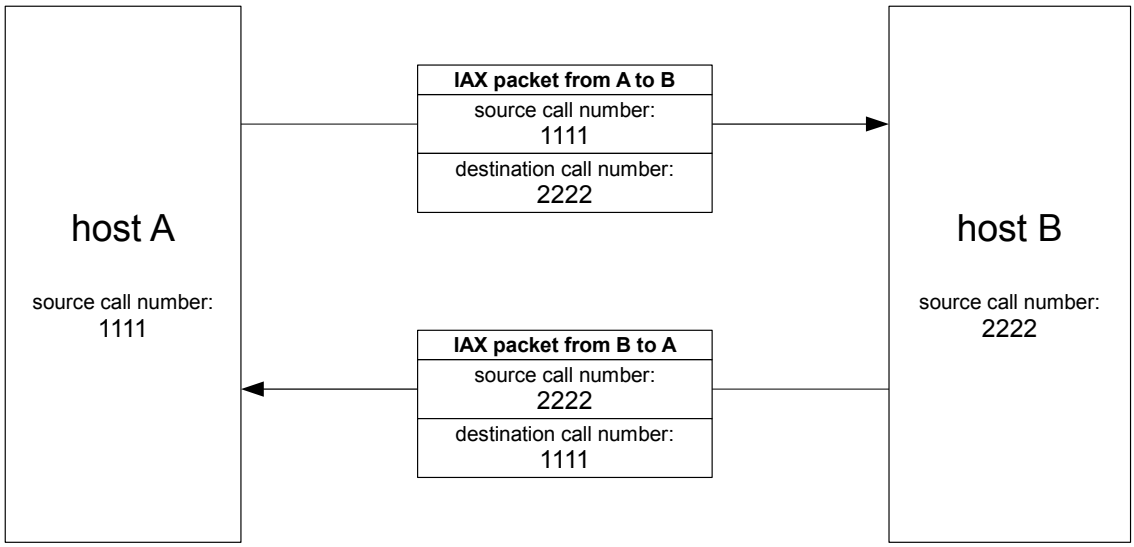


Figure 3.1: IAX call and destination numbers

3.2 Audio Reconstruction

Once VoIP packets are obtained, it is possible to reconstruct the audio signal based on sequence and timing information. Both RTP and IAX provide a timestamp with each packet, but it has different meaning in each protocol:

- In RTP, timestamp depends on payload format and indicate a sampling instant of first octet of payload. In other words, it is the number of *samples* since the first transmission of the call.
- In IAX, timestamp is independent on payload format and indicates a number of *milliseconds* since the first transmission of the call.

  Note that the timestamps may overflow and start from zero.

Both protocols also provide sequence number starting at zero:

- In RTP, sequence number increments by one with each packet sent.
- In IAX, sequence number increments by one with each full frame sent, mini and trunk frames do not contain or increment sequence number.

Even though a VoIP packet usually carries 20 ms of audio signal in practice, it may generally carry payload of arbitrary length. Therefore, it is necessary to recognize payload audio format and compute the duration of the payload from its size.

For example, 10 ms of audio are encoded to 80 octets by G.711 codec and 10 octets by G.729. 160 octets of G.711 payload imply duration of 20 ms, while 160 octets of G.729 payload imply duration of 160 ms.

The simplest naive way to reconstruct the audio signal carried by a stream of VoIP packets is to sort the packets by their ascending RTP/IAX timestamp and concatenate their payloads. This approach does not account for late packets which were dropped by the receiver, reconstructed signal would contain parts that were not actually available. Also, dropped packets create a gap in playback. Receiver filled this gap with silence, last transmitted payload or some other way, but reconstructed signal would skip such gap entirely, producing shorter audio signal.

To accurately reconstruct the audio signal, receiving host behavior must be emulated, taking into account RTP/IAX timing information, payload duration and packet transmission time. More info about the specific method implemented in our system is available at section 4.3.

3.3 VoIP audio formats

In the beginning of a VoIP call, an audio format must be negotiated which all participating hosts are able process. Audio data are encoded into the selected format by the sender, transmitted, and decoded by the receiver. Individual audio formats differ in many aspects such as audio transmission quality, required bandwidth, processing complexity, licensing, sample rate and more.

For example, G.711 is a free simple audio format with high audio quality and very low processing complexity but with high bandwith usage (8 kB/s). On the other side, G.729 is a patent burdened format with good audio quality and low bandwith usage (1 kB/s) but with high processing complexity.

Decoding G.729 and other proprietary formats requires not only a complex software but also a license. Even if open source libraries to transcode these codecs existed, a paid license would still be required to actually use them.

For more information about VoIP audio formats see RFC 3551 [3] and/or RFC 5456 [6] section 8.7 Media Formats.

Chapter 4

Design of the Application

In this section, application implementation is decomposed into subproblems. Details of RTP/IAX detection and audio reconstruction are explained. Command line tool usage and system interface are presented. Source code structure of the implemented system is briefly described and source code metrics are provided.

4.1 Problem Decomposition

The system was named `voipstream` and it is written in `python`[1] programming language. High level system overview is presented in Figure 4.1. The system must be able to perform following tasks:

- *Read IP packets*: `libpcap` library is used because it is widespread and well supported and provides the ability to read packets directly from network interface or file in pcap format which is a de facto standard for storing network traffic (not only) on linux systems.
- *Detect RTP and IAX packets*: Described in section 4.2.
- *Extract information form IP, UDP, RTP and IAX packet header*: `scapy`[2] python library is used with custom decoders.
- *Classify RTP/IAX packets into individual streams.*
- *Insert RTP/IAX payloads into individual streams.*
- *Reconstruct audio signal from payload streams*: The the audio data are reconstructed in memory on the fly. In the end, raw audio data are written, one file per stream. For G.711 codec, WAVE audio format output is supported, including dual channel audio for bidirectional streams.

4.2 RTP and IAX detection

Packets with source or destination port 4569 are assumed to be IAX. Other packets are assumed to be RTP if following condition are met:

[1] `http://www.python.org`
[2] `http://www.secdev.org/projects/scapy/`

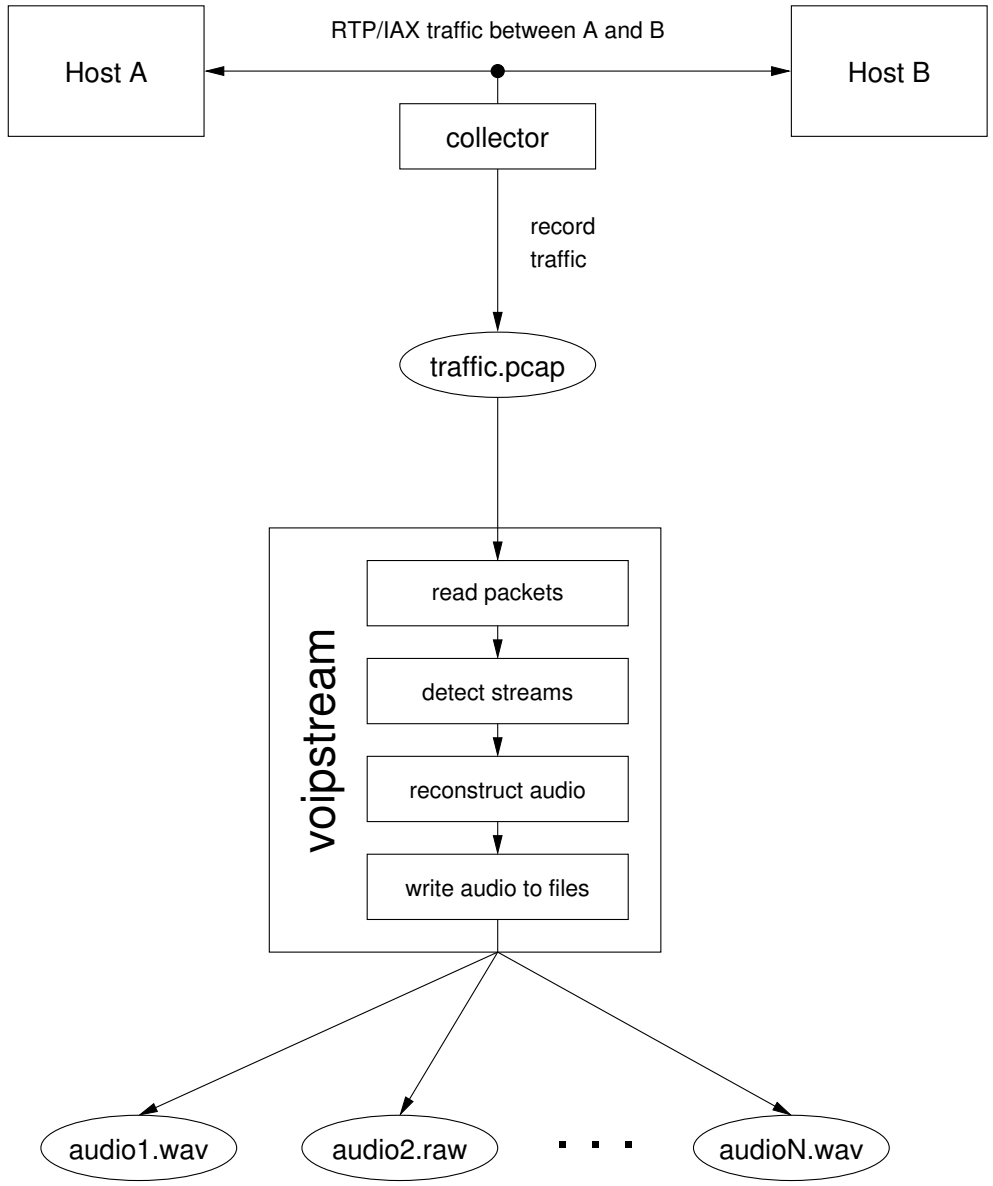


Figure 4.1: High level system overview

- Both source and destination UDP ports are even and unprivileged (greater or equal to 1024).
- Packet is long enough to contain RTP header and data.
- RTP version (2 bits) is 2.
- RTP payload type (7 bits) has a known value.

This is a rough heuristic that works on testing samples of traffic between two VoIP servers using both RTP and IAX but it is far from universal.

4.3 Audio reconstruction

A simple static jitter buffer emulation is supported to reconstruct audio as realistically as possible. Simply ordering packets by timestamp does not reproduce errors created by packets loss which may be undesirable for quality assessment. Maximum delay in timestamp units is defined as d_{max}. Jitter buffer stores largest timestamp seen t_{max} and compares it with each inserted packet's timestamp t_i. When a packet satisfies condition

$$t_i + d_{max} < t_{max}$$

it is considered to be late and dropped. Otherwise, it is inserted into jitter buffer based on its timestamp. Jitter buffer with $d_{max} = 50$ is illustrated in Figure 4.2.

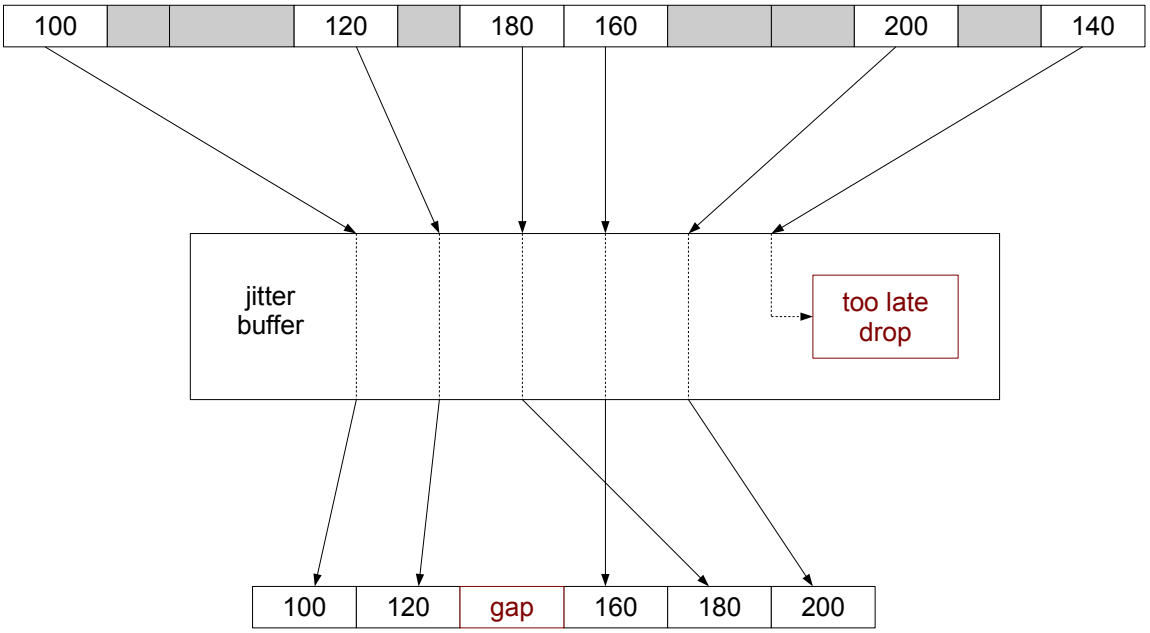


Figure 4.2: Jitter buffer illustration. On top, stream of packets is displayed. White packets are VoIP packets with numeric timestamp. Gray packets are other traffic.

Two advanced features are implemented for supported codecs. First, gaps in audio caused by packet loss can be filled with silence or optionally with beep sound. This prevents time warping and in case of beep helps to identify lost packets in audio signal. Second, bidirectional streams between two hosts are saved in dual channel WAVE file including padding to properly align them in time, both sides of a call can be listened to simultaneously. These are codec dependent features that work only on G.711 A-law and μ-law. They are mostly a proof of concept to demonstrate reconstruction possibilities as decoding of other VoIP codecs is not trivial, especially the popular patent burdened G.729.

4.4 Usage and Interface

`vget.py` command line frontend is provided which takes a `pcap` file as an argument. Typical invocation involves `-d` option specifying output directory, for example

```
./vget.py -d out_dir input.pcap
```

`vget.py` is also an example of how to use the `voipstream` interface. To extract audio from `pcap` file, create an instance of `StreamSeeker` class and call its `seek_and_dump_pcap()` method.

See Appendix A for more information including software requirements.

4.5 Source Code Structure and Metrics

`voipstream` python package is split into following files (ordered by relevance):

`vget.py`	Fronted, VoIP audio extractor.
`voipstream.py`	High level voipstream public interface.
`jb.py`	voipstream jitter buffer - payload stream reconstruction.
`payload.py`	Payload specific data and functions.
`rtp.py`	scapy RTP layer (decoder).
`iax.py`	scapy IAX layer (decoder).
`gap.py`	Audio gap related functions.
`dumper.py`	Functions for writing audio to files.
`wave.py`	Classes for writing WAVE files.
`utils.py`	Utility functions.
`config.py`	A simple container for configurable options.

Figure 4.3 shows a UML class diagram of relevant `voipstream` classes.

The `voipstream` source files span 1231 lines and `SLOCCount`[3] software metrics tool reports 917 physical source lines of python code. See Table 4.1 for per source file metrics.

file	**lines**	**characters**
payload.py	383	8142
jb.py	192	5336
iax.py	156	3315
voipstream.py	105	2597
wave.py	88	2131
vget.py	78	1662
gap.py	70	1387
dumper.py	46	934
rtp.py	41	1453
loc.py	39	772
utils.py	24	350
config.py	9	174
total	1231	28253

Table 4.1: Numbers of lines and characters per source file

[3]SLOCCount homepage: `http://www.dwheeler.com/sloccount/`

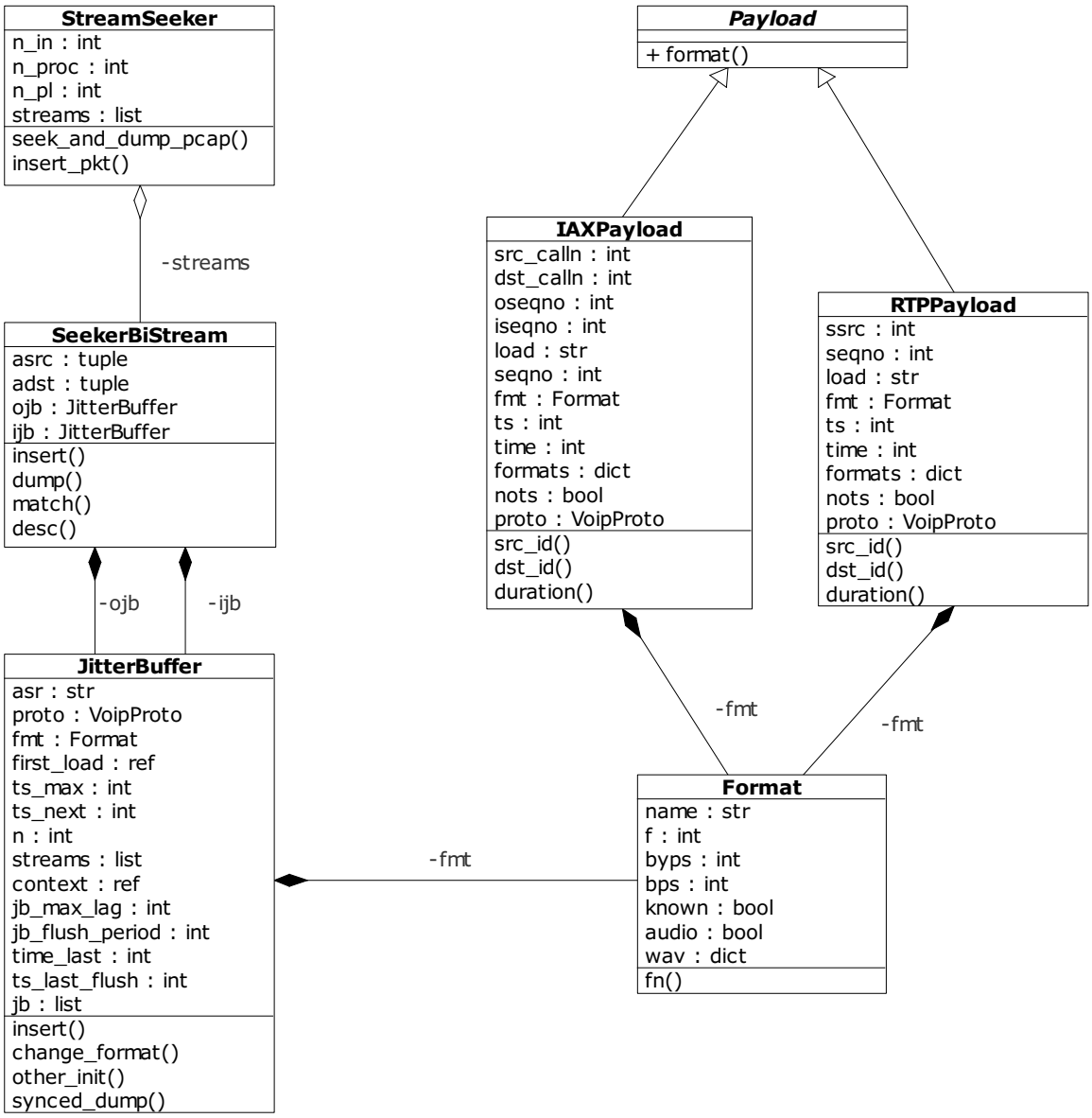


Figure 4.3: UML class diagram

Chapter 5

Testing

In this chapter, the implemented system is tested on laboratory, real and random data sets and results are presented. System performance is evaluated, explained and compared with other tools.

5.1 Testing Setup and Terminology

All tests were executed on HP ProBook 4510s notebook with dual core Intel Core 2 Duo T5870 CPU and 4 GB RAM on linux kernel version 3.0.26.

A *call* consists of a voice packet stream between two hosts and an opposite direction voice packet stream between them (if available).

5.2 Laboratory Data

Laboratory data set was created by recording traffic between two Asterisk servers using `tcpdump` utility during short testing calls with various settings. Each `pcap` record contains exactly one call. Table 5.1 describes the laboratory data set.

record file	**protocol**	**codec**	**# used/all packets**[1]	**trunk**
`RTP-G722-bi.pcap`	RTP	G.722	754/826	-
`RTP-GSM-bi.pcap`	RTP	GSM	705/748	-
`IAX-trunk-ts-G711-bi.pcap`	IAX	G.711	126/263	TS[2]
`IAX-trunk-nots-G711-bi.pcap`	IAX	G.711	315/368	no TS[3]

Table 5.1: Laboratory data set

Laboratory data were used during development to test the correct operation of the system and thery are available on attached CD in `data/lab` directory.

[1] number of packets used for audio reconstruction / number of all packets in capture files
[2] IAX trunk frame with individual timestamps
[3] IAX trunk frame without individual timestamps

5.3 Real Data

Real data set consisted of 600,000 packets of real VoIP network traffic captured in 96 seconds between two Asterisk servers with 100 MB Ethernet connection using `tcpdump`[5] utility. Testing setup is illustrated in Figure 5.1. The set was stored and read from 108 MB `pcap` file and contained both RTP and IAX streams including IAX trunk frames. We do not have permission to publish these data, only testing results are included.

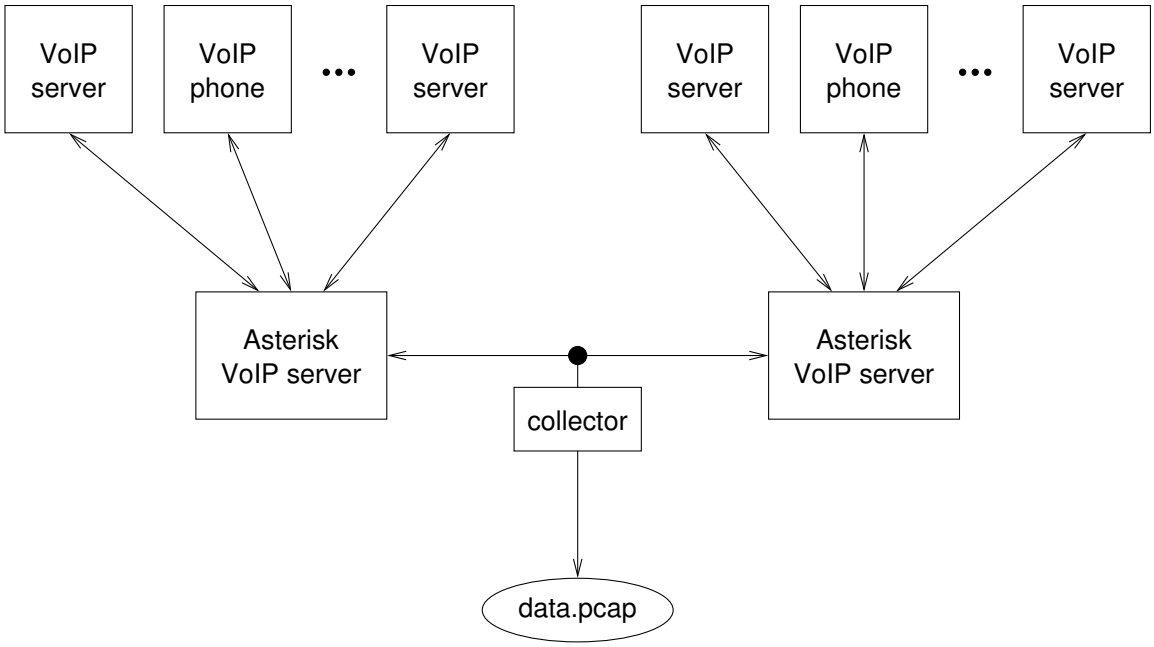


Figure 5.1: Real data capture setup

packets	600,000
duration	96 s
size	108 MB

Table 5.2: Real data set summary

The test was run using command:

```
./vget.py -d dump real.pcap
```

Time was measured using `time` unix utility. After 14 minutes, 143 calls were detected and written to 181 files. There are more files than calls because each direction is written separately for not supported codecs such as G.729. Correct function was verified by listening to output audio in WAVE format. Result are summarized in Table 5.3.

[5] http://www.tcpdump.org

detected RTP/IAX packets	555,458
detected calls	143
RTP calls	128
IAX calls	15
G.711 calls	97
G.729 calls	46
elapsed time	14 minutes
output audio size	64 MB

Table 5.3: Real data set test results

5.4 Random Data

Random data set consisted of 200,000 packets of traffic captured using Wireshark[6] on the testing notebook during daily usage including HTTP, SPDY, XMPP and BitTorrent traffic. Random data set are available on the attached CD as a `data/random.pcap` file.

packets	200,000
duration	345 s
size	149 MB

Table 5.4: Random data set summary

The test was run using command:

```
./vget.py -d dump random.pcap
```

After 3 minutes, the program exited with 0 calls found.

5.5 Performance

The system was written in `python` programming language which is dynamic and known to be much slower than static compiled languages such as C. Examining CPU usage with `cProfile` python profiler revealed that most CPU time is consumed by `scapy` library. That is to be expected, as `scapy` allocates and populates new python object for each packet.

Figure 5.2 shows the relation between number of packets and processing time.

An implementation of the system in C programming language using only `libpcap` library which was started later (it is not part of this work) is able to process the real data set approximately *650 times faster*. `rtpbreak` and Cain tools described in section 1.3 crash when processing the real data set.

Although `python` and `scapy` library allowed fast and elegant implementation, resulting system performance might be insufficient for processing of large amounts of traffic.

[6] `http://www.wireshark.org`

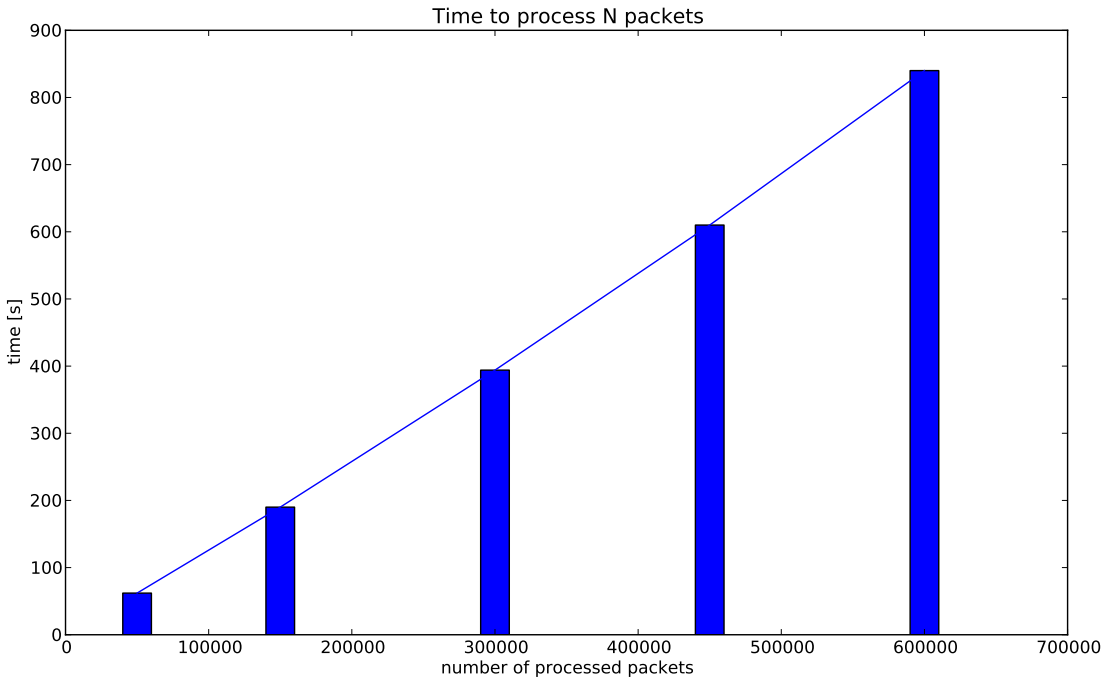


Figure 5.2: Time to process N packets

5.6 Summary

We have tested the `voipstream` system on laboratory, real and random data sets to verify its correct operation. The system performed well but its performance on large amount of data was poor due to slow `scapy` python library.

Chapter 6

Conclusions

6.1 Summary

We have implemented a system for VoIP stream detection and audio reconstruction in `python` programming language which takes an IP traffic packets in `pcap` format as input and outputs audio extracted from detected VoIP calls into files. For calls using G.711 (μ-law or A-law) encoding, system is able to fill gaps in audio caused by packet loss and save audio in popular WAVE format, including the ability to save both sides of a call in one dual channel audio file. System is able to decode RTP and IAX protocols including IAX trunk packets which are not supported by any similar software we are aware of.

In this thesis, we have briefly explained principles of VoIP communication including basics of IP networking, thoroughly described packet format and usage of RTP and IAX protocols and compared them, explained methods of VoIP stream detection and audio reconstruction and described the system implementation.

The system was tested on a real data provided by a VoIP operator composed of hundreds of thousands of IP packets on which it performed well.

Furthermore, we have modified Wireshark network protocol analyzer to properly analyze IAX trunk frames. These changes have been accepted upstream and are now part of Wireshark.

6.2 Future Work

First, several aspects of the system could be improved. The detection of IAX and especially RTP streams is a complicated task as the packets carry binary data. More advanced methods of detection could be employed, for example signaling protocol analysis or more sophisticated heuristic analysis. Decoding of various codecs used in VoIP including proprietary ones could be incorporated into the system. Rewriting the system in lower level programming language such as C or C++ could drastically improve its performance.

Second, output audio data could be analyzed. An VoIP quality evaluation system working on audio signal level could be built on top of our system as illustrated in Figure 6.1. Such system would compare received audio (reconstructed from IP traffic using our system) with original audio using POLQA or similar automated audio quality evaluation method. It might be used by VoIP providers and administrators to reveal problems and ensure service quality.

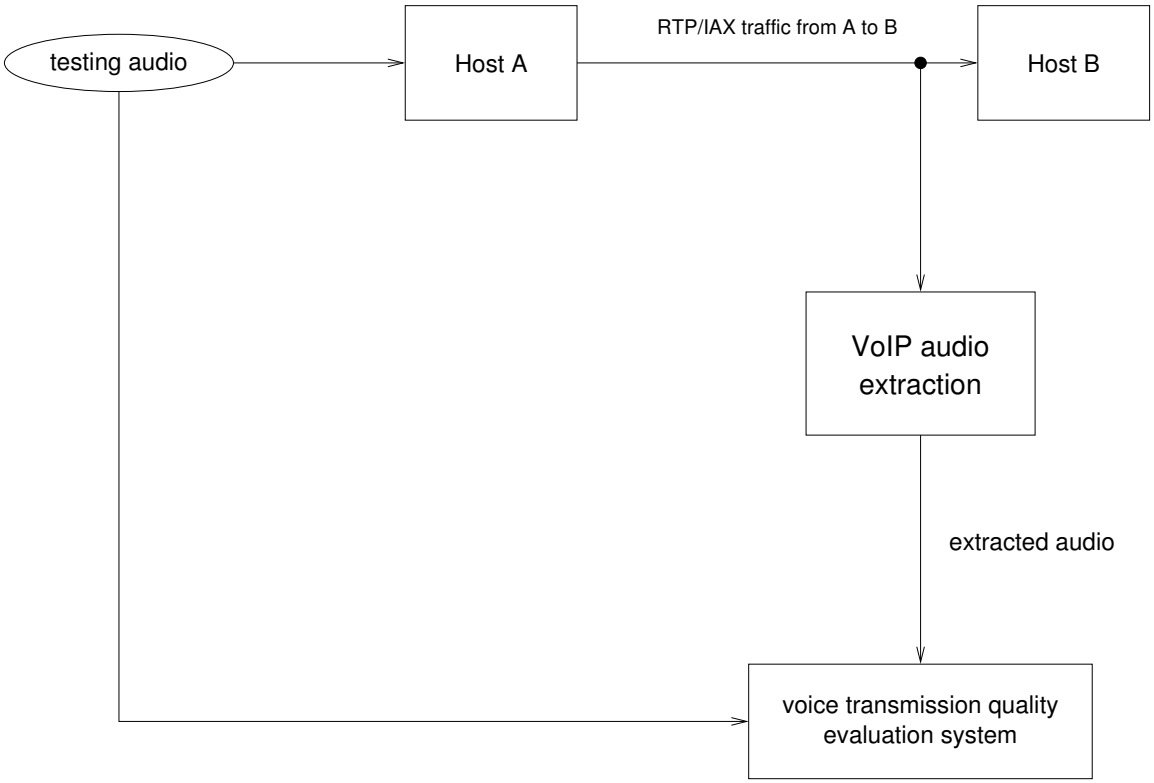


Figure 6.1: Voice transmission quality evaluation system using our VoIP extraction tool

Bibliography

[1] Braden, R.: Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), Říjen 1989.
URL `http://www.ietf.org/rfc/rfc1122.txt`

[2] Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; aj.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), Červen 2002.
URL `http://www.ietf.org/rfc/rfc3261.txt`

[3] Schulzrinne, H.; Casner, S.: RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard), Červenec 2003, updated by RFC 5761.
URL `http://www.ietf.org/rfc/rfc3551.txt`

[4] Schulzrinne, H.; Casner, S.; Frederick, R.; aj.: RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), Červenec 2003.
URL `http://www.ietf.org/rfc/rfc3550.txt`

[5] Sector, I. T. S.: Control protocol for multimedia communication. ITU-T Recommendation H.245, Září 2009.
URL `http://www.ietf.org/rfc/rfc3261.txt`

[6] Spencer, M.; Capouch, B.; Guy, E.; aj.: IAX: Inter-Asterisk eXchange Version 2. RFC 5456 (Informational), Únor 2010.
URL `http://www.ietf.org/rfc/rfc5456.txt`

Appendix A

Manual

A.1 Requirements

`python` and `scapy` python library version 2.2 are required. `scapy` v2.1 should work as well but it isn't thoroughly tested, v2.0 is known to not work.

`scapy` packages are available in most modern linux distributions and should be easy to install using package manager. On Debian based distributions,

```
aptitude install python-scapy
```

command should install `scapy` with all requirements. On Red Hat based distributions,

```
yum install scapy
```

should be equivalent. On systems where a native package is too old or not available, `scapy` can be installed using `pip` tool

```
pip install scapy
```

or its `easy_install` predecessor from python setuptools

```
easy_install scapy
```

Alternatively, you can manually install `scapy` from its homepage:
`http://www.secdev.org/projects/scapy/`

A.2 Usage

`vget.py` command line frontend is provided. Run it without parameters for help. Typical usage involves `-d` option to specify output directory.

To output all the recognized audio streams from `record.pcap` file into `out` directory:

```
./vget.py -d out record.pcap
```

To do the same with audio gaps in supported formats (G.711) filled with beep sound instead of silence:

```
./vget.py -b -d out record.pcap
```

A.3 Interface

`vget.py` is a minimal frontend and illustrates the usage of the `voipstream` interface. To extract audio from `pcap` file, create an instance of `StreamSeeker` class found in `voipstream.py` and call its `seek_and_dump_pcap()` method. Custom classes can be built on top of the `JitterBufer` class found in `jb.py`.

Appendix B

Wireshark IAX trunk packets patches

Author of this thesis has created and submitted patches for Wireshark network protocol analyzer described in subsection 1.3.2 to enable IAX trunk packet dissection. These patches were accepted into upstream and allow Wireshark users to inspect IAX trunk frames.

The patches can be found in Wireshark bugzilla:

- `https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=4783`
- `https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=6240`

Appendix C

CD Contents

The attached CD contains following folders:

- `voipstream`: The VoIP extraction system source code files:

`vget.py`	Fronted, VoIP audio extractor.
`voipstream.py`	High level voipstream public interface.
`jb.py`	voipstream jitter buffer - payload stream reconstruction.
`payload.py`	Payload specific data and functions.
`rtp.py`	scapy RTP layer (decoder).
`iax.py`	scapy IAX layer (decoder).
`gap.py`	Audio gap related functions.
`dumper.py`	Functions for writing audio to files.
`wave.py`	Classes for writing WAVE files.
`utils.py`	Utility functions.
`config.py`	A simple container for configurable options.
`snd/`	Samples for audio gap filling.

- `data`: Testing data in `pcap` format described in chapter 5:

`random.pcap`	Random data set.
`lab/`	Laboratory data set.

- `text`: LaTeX sources of this Bachelor's Thesis.