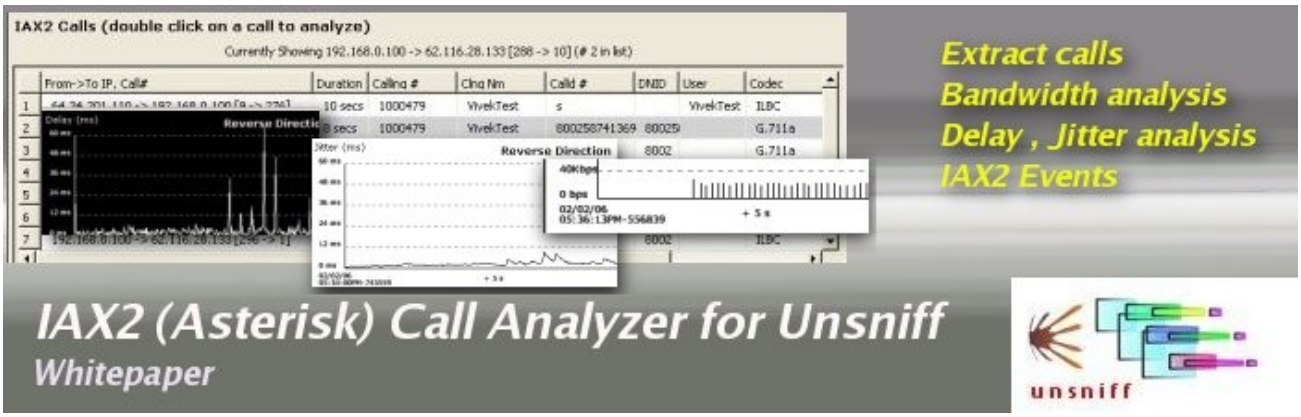


*This whitepaper presents a call analyzer for the IAX2 protocol. Learn how you can use this tool to draw bandwidth, delay, jitter charts for a VoIP network that uses this protocol.*

## Asterisk VoIP Call Analysis

Are you one of the growing number of people deploying the Asterisk VoIP platform ? We present a add-on tool for *Unsniff Network Analyzer* that performs complete VoIP Call Analysis for the Inter Asterisk Exchange (IAX2) protocol. Measure and plot call bandwidth, interarrival delay, jitter, packet loss, and IAX2 events for each direction of call. This tool like others in this series (such as TCP/IP analysis) is written in the excellent **Ruby** scripting language using the **Fox-Ruby** toolkit for its user interface. Full source code of the tool is provided for you to tweak it to your liking. If you are working with Asterisk in any capacity, this is a "must have" tool on your workbench.

## The IAX2 Protocol

Asterisk (the open source PBX server) is rapidly gaining in popularity as a powerful alternative to expensive PBX systems. The IAX2 (Inter Asterisk Exchange ver 2) protocol is the native language of Asterisk. The main strength of IAX2 when compared to competing protocols such as RTP/SIP/H.323 is its friendliness to NAT (Network Address Translation) and firewalls. IAX2 uses only a single UDP port 4569 to carry both media and control messages.

## Features

The IAX2 Call Analyzer can

- Extract all IAX call details
- Analyze the bandwidth usage
- Analyze the interarrival delay
- Analyze the jitter experienced
- Plot all IAX2 events

## Extract calls from a capture file

If the IAX2 Analyzer sees a “NEW” control message and a corresponding “ACCEPT” message, it assumes a call has been set up and creates a new call. The data associated with each call is extracted from these two messages and shown in a table. You can then double click any call in the table to analyze that call.

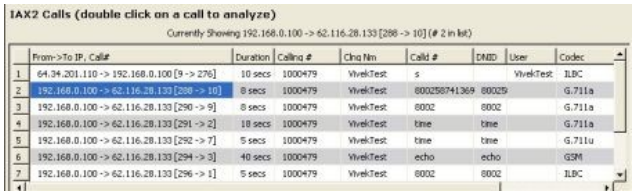


*Illustration 1: Call List*

For each call the following data is shown.

| Data Item | Description |
|---|---|
| From IP , To IP , Source Call Number, Dest Call Number | These four attributes are used to uniquely identify a call. |
| Start Time | When was the call started ? This corresponds to the time when the ACCEPT message was seen. |
| Duration Seconds | How long was the call in progress ? This is the time difference between the IAX Control HANGUP message and the call start time. |
| Codec | What codec was used for the call ? |

| Data Item | Description |
|---|---|
| Calling number | The number or extension of the calling phone |
| Calling name | The name of the calling phone |
| Called number | The number or extension of the called phone (maybe a SIP URI) |
| DNID | The dialled number ID |
| Username | The user who made the call |

*Table 1: Call properties*

## Call Bandwidth Chart

One of the most important factors that needs to be considered while deploying a VoIP system is the bandwidth requirements. The bandwidth at your disposal will help you plan tradeoffs on voice quality by selecting the right codec. Each codec has a specific voice stream bandwidth. For example G.711 generates a packetized voice stream of 64kbps bandwidth. However, the actual bandwidth required is larger due to the IP/UDP/IAX2 headers. The call bandwidth chart plots the actual bandwidth usage of the selected call.

The methodology used by the bandwidth analyzer is:

**Use entire packet size including Ethernet/ IP/ UDP/ IAX2 headers:** The entire packet size is used to calculate the bandwidth requirements. As an example, if we are analyzing a call which uses the GSM codec, the total payload is calculated as: Total payload = 33 (GSM) + 4 (IAX2) + 8 (UDP) + 20 (IP) + 14 (Ethernet) = 79 bytes
You can compare your bandwidth utilization with the specifications different codecs at http://www.openh323.org/docs/bandwidth.html

**Sample used bandwidth every 200 ms:** We use a 200ms sample rate of used bandwidth to plot. If you wish to use a more granular sample, then you can change the line @sliceus = 200000 line in the iax2ana.rb script.

In the picture below we can see that the call uses the iLBC codec and the used bandwidth is around 35kbps in each direction. You can play with this tool for other codecs such as G.711, GSM, Speex and verify for yourself the bandwidth usage claims of each of these codecs.

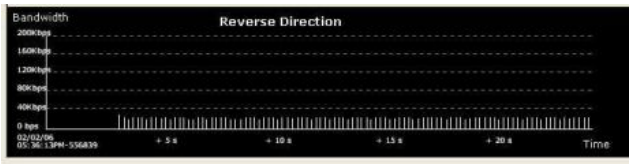


*Illustration 2: Call Bandwidth Chart*

## One-way delay (Latency)

When you talk into a microphone,the first thing that happens is the sampling of your voice (often at 8000 times per second). The one-way delay is the time between the sampling of a voice signal and the playout of the sample at the other end. This includes codec delays, network buffering, transmission delays, and playout buffering. While all these factors are important to determining the overall voice quality, the transmission delay is the largest and most unpredictable component. Even casual users of voice over IP services would have noticed that as latency gets longer, it becomes much harder to carry out a meaningful conversation. The well accepted benchmark for maximum one-way delay is 150ms (ITU-T G.411). Unfortunately, measuring one-way delay is not easy. For starters, we need timestamp synchronization between the two endpoints and a way to carry this information in each packet. This is too much to ask in a loosely controlled network such as the internet. So what we are really saying is, Unsniff cannot calculate the one-way delay. Luckily it turns out the inter-arrival delay (next section) is a pretty good indicator as well of voice quality. Unsniff can calculate the inter-arrival delay and plot them on a chart for the duration of the call. So lets accept that one-way delay calculation will remain an elusive goal for us and move on to the next section !

## Interarrival Delay

In the previous section, we saw that measuring one-way delay is not possible given our constraints. Let us now focus on the next best thing, the inter-arrival delay.

### What is Interarrival delay ?

Assume a VoIP transmitter is sending voice packets exactly 20ms apart, so they keep shooting out of the sender at 0ms, 20ms, 40ms, tick,tick,tick,tick, . In an ideal situation the receiver must also receive these packets exactly 20ms apart (at t+0ms, t+20ms, t+40ms) and so forth. Actually this is what happens in traditional phone systems. Unfortunately in an IP network, this not the case. While we can easily arrange the sender to transmit packets at exactly 20ms intervals, we cant "arrange" for them to be received at exactly the same rate. This is due to the routers and the packet switched network in between. So we may have a case where the receiver gets the packets at

(t+0, t+23, t+45, t+61, etc). This means that instead of arriving nicely at 20ms intervals, we get packets at 23ms, 22ms, and 16ms intervals. Since the actual numbers (23ms, 22ms, 16ms above) are dependent on the transmit rate (20ms), we define interarrival delay as the difference between the two. In this case we have interarrival delays of 3ms for the 1st packet, 2ms for the 2nd packet, -4ms for the 3rd packet. We also do not care about the sign. So we have delays of 3ms, 2ms, 4ms in the above example. This is how interarrival delay is calculated. This is also sometimes misleadingly referred to as simply "delay".

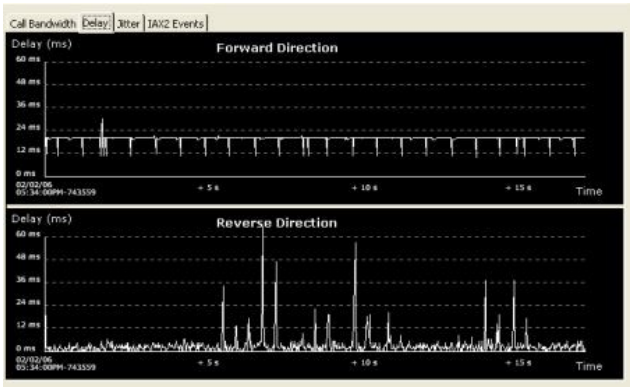


*Illustration 3: Interarrival delay*

**Calculating interarrival delay for IAX2**

The two measurement points required to calculate delay are the received time and the transmit time.

***Received Time***

The received time is when the packet was captured by Unsniff. Getting this is not a problem because Unsniff gets a microsecond resolution timestamp for each captured packet from its provider (Winpcap or Windows Raw Sockets). The only tricky part it to position Unsniff close to the receiver if we want a reasonably accurate measurement. Ok next..

***Transmit Time***

This is when a packet was transmitted. Unsniff does not have access to this information because it is only running near the receiver. You could probably run another instance of Unsniff near the sender and correlate two packets by writing simple scripts. This is not feasible in a number of situations because we may not have access to both ends of the call. So what do we do ? The timestamp carried in each IAX2 packet comes to our rescue. This timestamp is the number of milliseconds since the call began. We can use this timestamp because we are only interested in the time difference not the actual time. All we have to do is convert the microsecond resolution timer at the receive side to milliseconds and we have our measurements.

## Jitter (statistical variation of Interarrival Delay)

Within bounds, interarrival delay can be easily controlled by adding a buffer. Sometimes interarrival delays are not problem at all because they are uniform. Let us consider an example : If at a receiver packets arrive at (t+0, t+22, t+40, t+62, t+80), we can see that the delays are +2ms, -2ms, +2ms, -2ms and so on. So you can just add a buffer for a single packet and still play it out with no problems, except that you have now introduced some additional delay due to the buffer. The real problem happens when these interarrival delays vary. This interarrival delay variation is known as the jitter (a term borrowed from electronics).

The jitter is quite a good indicator of voice quality. However, it does not mean much at a single point of time. So, if someone walks up to you and says "*My call experienced a jitter of 8.3ms at 10:42:22 AM*". You will not know what to say about the voice quality at that instant. You can use jitter to compare different calls or compare time periods of the same call. If the same person walks up to you and says, "*My call experienced high jitter (between 7ms and 8ms) for a few minutes at 10:42:22AM compared to the beginning of the call (between 0ms and 2ms)*". You can conclude that the voice quality during the phase of high jitter was lower than during the period of low jitter.

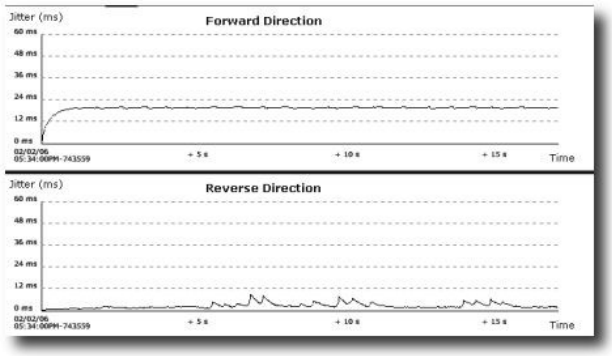


*Illustration 4: Jitter (both directions of call)*

**Calculating jitter for IAX2Jitter Chart for IAX2**

The IAX2 informational draft does not include a formula for calculating jitter from interarrival delay samples. So we stole one from RFC3550 (RTP - A transport protocol for real time applications) - the predominant VoIP protocol. We think that the formula is valid for both RTP and IAX2.

The formula is :

Jitter (at time T) = Jitter (at time T-1) +
(Interarrival Delay(at time T) - Jitter (at time T-1) ) x 1/16

So the jitter is calculated continuously over the duration of the call. You may ask , What is the mutliplier "1/16" doing up there in the formula ? It is the "gain parameter". Here is what RFC3550 has to say about the 1/16 factor. "*This algorithm is the optimal first-order estimator and the gain parameter 1/16 gives a good noise reduction ratio while maintaining a reasonable rate of convergence.*"

## IAX2 Events

We have looked at call bandwidth, interarrival delay, and jitter charts for voice over the IAX2 protocol. While all these give you an idea of the quality of the call. You still want to see how the IAX2 protocol itself works over the time period of a single call:

- what messages are sent at what points in time ?
- when was the call setup / call hung up
- were there any DTMF digits transmitted
- was the call transferred
- IAX2 control messages (PING/PONG/LAG/etc)
- how many mini frames vs full voice frames

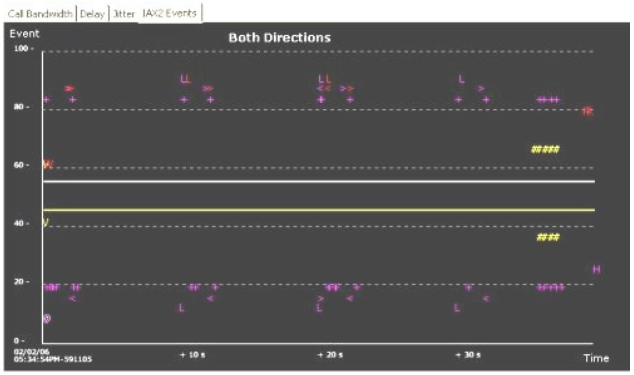


*Illustration 5: IAX2 Events*

| Notation | What it means |
|---|---|
| White Dots along top around the 55 mark | IAX2 Mini Frames containing voice samples from initiator of call |
| Yellow Dots along top around the 45 mark | IAX2 Mini Frames containing voice samples from receiver of call |
| N | NEW |
| > | PING |
| R | Registration messages |
| < | PONG |
| V - yellow | Full frame containing voice |
| + | ACK |
| # | DTMF digit pressed |
| any RED color | Retransmitted packet |
| A | Accept |
| L | Lag messages (LAGRQ and LAGRP) |
| O | Other IAX Control message |
| * | RINGING |
| @ | ANSWER |
| U | Other Control message |

*Table 2: IAX2 Events Symbols*

## *How to Download and run this tool*

**Availability**

Free download from the Unleash Networks Website at http://www.unleashnetworks.com/articles/asterisk-call-analyzer-for-iax2.html

1. Install **Unsniff Network Analyzer** and **Ruby**
2. Download iax2ana.rb (the IAX2 Call Analyzer Ruby Script) and UnleashCharts.rb

**How to run ?**

- Capture some IAX2 packets from the network under test
- Run iax2ana.rb on the captured file
- Double click on a call to analyze

**Usage:**

```
iax2ana <capture-file-name>
```

*Where:*

*capture-file-name : Capture file in Unsniff (*.usnf) format*

**Example:**

```
c:\RubyTest>iax2ana.rb  AstrCap.usnf
```



UNLEASH NETWORKS

http://www.unleashnetworks.com
Author: Vivek Rajagopalan (Unleash Networks)
vivek@unleashnetworks.com