SIP Troubleshooting #ONE

WORKSHOP

SIPCAPTURE

Written and Presented by:

Alexandr Dubovikov & Lorenzo Mangani

HOMER Development Team

http://sipcapture.org

HOMER

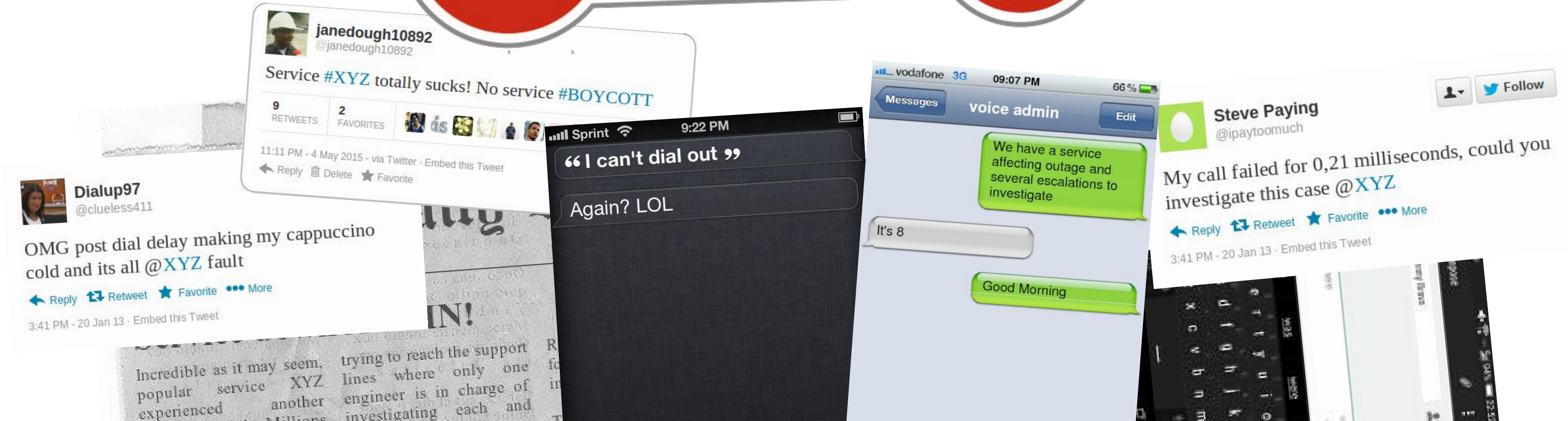
qxip ntop

Workshop Sponsored by QXIP / NTOP - http://qxip.net

SIP Troubleshooting #ONE

WORKSHOP

SIPCAPTURE



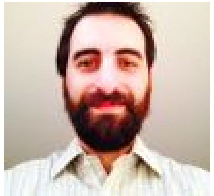
Who are we?

meet the **SIPCAPTURE** Development Team!

proud makers of HOMER

Alexandr Dubovikov

Senior Voice Expert for QSC AG, one of the major German voice and data providers. Alexandr holds a diploma in physics of Odessa State University and brings 20 years of experience in telecommunication techniques, contributing to many Open Source projects like FreeSwitch, SER, Kamailio, SEMS, Asterisk, SIPp, Wireshark. Alexandr is the main developer of Homer SIP Capture project. Also founder of IRC RusNet Network, one of the biggest national IRC networks in the world.



Lorenzo Mangani

Sr. Voice Engineer and Designer for the largest international cable operator worldwide, founder of Amsterdam based QXIP BV, Co-Founder and Developer of Homer SIP Capture project and voice specialist of the NTOP Team. Formerly a Sound Engineer, Lorenzo has been deeply involved with telecommunications and VoIP for well over a decade and has contributed ideas, design concepts and code to many voice-related Open-Source and commercial projects specializing in active and passive monitoring solutions.

Joseph Jackson

Sr. Network Engineer for VoIP Long Distance wholesale provider. Specializing in high performance and redundant network design with a special interest in high speed packet capturing and analysis. Focusing on providing real time VoIP metrics.

Who are you?

In order to adapt the speed and phasing of this workshop to a fair median we would like to quickly scope our audience

(please raise your hand when a matching group is mentioned)

Voice Pal 	Works with SIP occasionally and/or deals with other aspects of the network/business
Voice OP 	Works with SIP daily, dealing with real cases/solutions practicing deep commandline-fu
Voice Dev 	Works with SIP all day, leads or contributes to several Voice related projects all night

SIP Troubleshooting #1: Toolset in 30 minutes

with Team SIPCapture

We all know it - SIP is an *ASCII/UTF-8* application-layer control protocol defined by *RFC3261* that can initiate, modify and terminate sessions, presenting a wide variety of header fields, often carrying additional body data such as *SDP* used to allow *RFC3550* endpoint RTP communication.

If you work with SIP & RTP you know they can bring both tears of joy and pain - on the other hand, we would be jobless if it all was *perfect* ;)

This brief workshop will *attempt* to cover:

- tools of the trade to get the job done from the "one-off" session to permanent capture setups
- technical approaches and quick recipes for capturing SIP/RTP network packets in all weather conditions
- relevant community references, useful resources, ideas and links
- tools we ourselves developed to make your voice life a little easier

This workshop will unfortunately *not* cover:

- how to master SIP Protocol and its every RFC in less than 30 minutes w/ free drinks
- how to read packet captures blindfolded and complete SIP investigations using sniffer dogs
- techniques for capturing and decoding audio streams using the power of your mind and arduino

NOTE: Several Tools and Tool Suites will be referenced during this workshop, while most of them are freely available and/or fully Open-Source we decided to also mention and compare the features of some relevant commercial solutions suited as companions or extension to Open-Source components for completeness of analysis of the options in the higher end of the scope and for those in need of them. The choice is yours!

INTRODUCTION

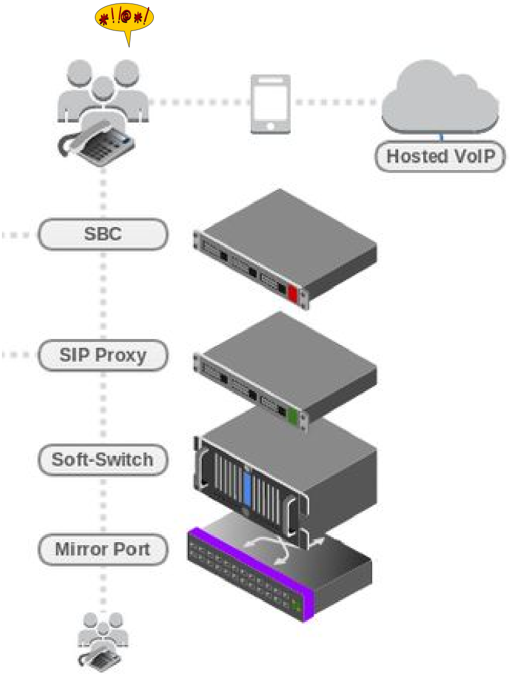
Battlefield Hardline: VoIP

Introduction

VoIP Breakdown of Typical Areas of Investigation

Although issues with SIP setups can manifest themselves in many forms and shapes, the vast majority of them can be covered by investigating the following critical areas:

- INTEROPERABILITY ISSUES
  - Different vendors of semi compatible "standard" solutions
  - Different Interpretations and Implementations of RFCs and Standards in UAs
  - Misconfiguration of remote party/interconnect *(the hardest to prove and argue)*
- NEGOTIATION ISSUES
  - No common codecs or rates (ptime), DTMF transport/tone mismatch
  - No network path, NAT Detection and resolve Issues. (Vendor: A)
  - SDP from hell (Vendor: C) multiple Via: 127.0.0.1
- SYSTEM PERFORMANCE ISSUES
  - Stressed or Misconfigured Hardware/Software on either side of the call
  - Overloaded Transcoders, Gateways, etc.
  - Attacks/Scanning/DDOS attacks overloading voice sub-systems
- NETWORK & NETWORK PERFORMANCE
  - Routing Issues, NAT Issues, SIP ALG Issues
  - Latency, Jitter and UDP Packet Loss in transit
- OSI-8 ERROR
  - Dial Errors, Broken Handsets, Broken B-Party Handset, Broken Ears



<u>NEXT: How do we get to the juicy protocols out?</u>

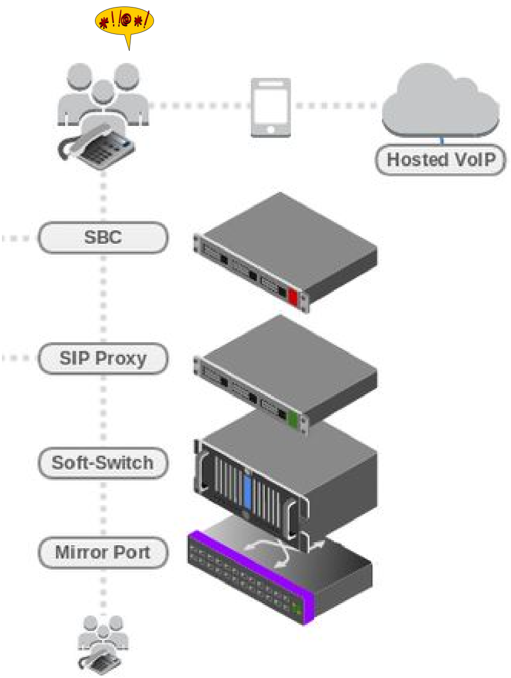
Introduction

VoIP Ecosystem and Elements

This workshop assumes basic familiarity with the standard elements and protocols typically involved with SIP Services and their roles. Beyond their implicit functional differences each system can produce plenty of valuable information and useful details:

Some Examples:

- SIP USER-AGENT, SBC / B2BUA, SSW
    - SIP, RTP, RTCP, CDRs, QoS Metrics, Application Logs
- SIP PROXY, REGISTRAR, ROUTER
    - SIP, Database & Application Logs
- MIRRORED ROUTER/SWITCH Ports
    - SIP, RTP, RTCP protocol traffic to/from peering networks
- OSI-8 / END-USER
    - Usage Logs, Issue Timestamps, Ultra Mean Opinion Score



CAPTURING THE PACKETS
PHYSICAL METHODS

SPAN / MIRROR PORT

Traditional method of capturing data

PRO:

Widely support on almost any “managed switch” even small work group swi

Easy to use.

Bidirectional traffic flows.

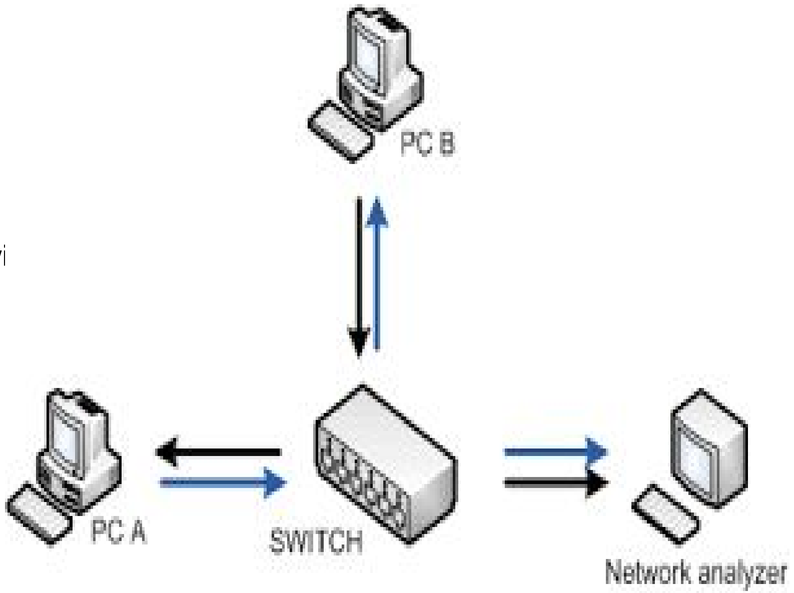
Many to one (depending on hardware).

CON:

Dumb capturing - no control of traffic selection.

Packet loss on over subscribed destination port.

Bad or corrupted packets are not transmitted.



CAPTURING THE PACKETS
PHYSICAL METHODS

Granular Packet Capturing.

Vendor dependant and platform dependant.

Cisco:

Vlan Access Control List (VACL)

Mark interesting traffic and only have that sent to capture port, rest of traffic is forwarded as normal

VLAN based - if all your devices are in the same vlan you won’t see intra vlan traffic.

Platform dependant - Catalyst 6500 and the Nexxus

Flow-based SPAN alt to VACL on Catalyst X Series.

Juniper:

Much more robust capturing.

Using Firewall filters can mark interesting traffic and forward to capture port.

Depending on model (SRX) can drop to linux OS and run tcpdump.

CAPTURING THE PACKETS
PHYSICAL METHODS

Network Taps

Active Taps:

Intelligent - able to identify traffic based on layers 3-7 and send to capture device.

Pros: Intelligent and programmable.

Cons: Epensive

Passive Taps:

Dumb tap. All traffic is replicated to the capture ports.

Pros: Cheap!

Cons: Super dumb

Remember never plug a TX into a capture port on an optical tap

REAL-TIME CAPTURE TOOLS

Terminal Heroes

Standard Tools
The ABC of packet capturing

“Everybody lies, but not SIP “ *Doctor House* 

Let's face it - If the packets we need are not available for us to investigate when we need them, we're in trouble.

Regardless of the title or experience, a good voice engineer should be prepared to do whatever the conditions dictate to capture voice packets needed to get the job done. Sometimes we own the systems and can pick our fancy weapons, other times we are bound to strict limitations - *you simply never know* - this is why the *ABC* really never gets *too* old.

Amongst the *"evergreen"* packet capture tools every voice op should know and use, we will briefly mention:

tcpdump, wireshark, tshark, ngrep, sipgrep, sngrep, pcapsipdump, captagent

Several of the above will offer overlapping features and/or equivalents to perform similar actions - this is great news for any voice generalist, as you never know which default tools will be found waiting for you on an impaired alien system.

NEXT: Let's see a few everyone should be familiar with...

Standard Tools

The old school ways: 8 bits games, tcpdump

Let's assume everyone knows **tcpdump**, the grandfather of packet capture tools and highlander of any unix system. **tcpdump** familiarity is definitely not an optional - when everything else fails, this good old friend won't let you down.

Capturing SIP Packets with tcpdump:

Display SIP packets with verbose details:

```
# tcpdump -nqt -s 0 -A -vvv -i eth0 port 5060
```

Capture SIP packets to disk in PCAP format:

```
# tcpdump -nq -s 0 -i eth0 -w /tmp/dump.pcap port 5060
```

Capture SIP packets to disk in PCAP format, rotate file every 15mb w/ file timestamp:

```
# tcpdump -s 0 -w /tmp/capture-dep`date +%Y%m%d-%H%M%Z`.pcap -C15 udp and port 5060
```

NOTES:

-s 0
Setting *snaplen* to 0 sets it to the default of 65535

-n
Do not convert addresses to names.

-i
Input capture interface

-w
Output PCAP filename

Standard Tools

The old school ways: 16 bits games, tshark

TShark is a network protocol analyzer part of the wireshark family. It lets you capture packet data from a live network, or read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file delivering the power of wireshark filtering alongside many advanced functions including RTP heuristics.

Capturing Packets with Tshark:

Capture all SIP on specified port and switch files every hour:

```
# tshark -nq -i eth0 -b duration:3600 -w /tmp/trace/sip.pcap port 5080
```

Extract SIP Server/Client details from INVITEs:

```
# tshark -r myFile -R "sip.CSeq.method eq INVITE"
```

Capture SIP, RTP, ICMP, DNS, RTCP, and T38 traffic in a ring buffer capturing 100 50MB files continuously:

```
# tshark -i eth0 -o "rtp.heuristic_rtp: TRUE" -w /tmp/capture.pcap -b filesize:51200 -b files:100 -R 'sip or
  rtp or icmp or dns or rtcp or t38'
```

Filter on RTCP packets reporting any packet loss or jitter over 30ms:

```
# tshark -i eth0 -o "rtp.heuristic_rtp: TRUE" -R 'rtcp.ssrc.fraction >= 1 or rtcp.ssrc.jitter >= 240' -V
```

Analyze RTP Network Stream Quality by portrange:

```
# tshark -q -f 'udp portrange 20000-30000' -o rtp.heuristic_rtp:TRUE -z rtp,streams
 Src IP addr  Port    Dest IP addr  Port    SSRC        Payload     Pkts Lost      Max Delta(ms)  Max Jitter(ms) Mean Jitter(ms)
 10.1.3.143   5000    10.1.6.18     2006    0xDEE0EE8F  G.711 PCMA  236  0 (0.0%) 34.83          0.83           0.37
```

Standard Tools

The old school ways: Remote Captures

There are occasions where you might need to capture key packets on a remote system and analyze them locally. To avoid the trouble of saving and transferring pcap files, native linux options might come handy and apply fine to severals of our available tools:

Capturing Packets Remotely:

Capture remote traffic to local pcap with **tcpdump**:

```
# ssh root@host 'tcpdump -w - -p -n -s 0 port 5060 and host 1.2.3.4' > remote_capture.cap
```

Analyze a remote real-time capture stream using a local **wireshark** over ssh:

```
# wireshark -k -i <(ssh -l root 192.168.10.20 tshark -w - not tcp port 22)
```

Capture from remote system via named pipe, display using **sipgrep** and forward to HEP Collector:

```
# mkfifo /tmp/pcap
# ssh root@192.168.10.20 "tcpdump -s 0 -U -n -w - -i any portrange 5060-5090" > /tmp/pcap
# sipgrep -I /tmp/pcap -H udp:192.168.50.60:9060
```

Standard Tools

~~Decoding and Analyzing SIP TLS packet captures with Wireshark~~

The world is finally catching up with Encryption - this is great news for end users but can result in complications for voice ops. Unless you are capturing traffic from within your VoIP platform *(using an internal capture agent)* you might have to deal with troubleshooting TLS sessions.

Wireshark can decode SSL/TLS sessions when the following conditions are fulfilled:

- the private key of the TLS server is known *(both keys might be needed if mutual TLS (=client certificate) is used)*
- the TLS connections does not use a Diffie-Hellman cipher
- Wireshark captures the TLS session from the beginning *(including handshake)*

Configure **Wireshark** to decode SSL/TLS:

- Copy the server's private key to the PC running Wireshark. Configure Wireshark to use the key:
- Edit → Preferences → Protocols → SSL → RSA Keys List: *i.e.: ip.address.of.server,5061,sip,/opt/ssl/agent.pem*
- If the server uses Diffie-Hellman (DH) Ciphers by default you should configure the server to use other ciphers.

WIRESHARK EXAMPLE:

```
wireshark -o "ssl.desegment_ssl_records: TRUE" \
  -o "ssl.desegment_ssl_application_data: TRUE" \
  -o "ssl.keys_list: 4.2.2.2,5061,sip,/opt/ssl/agent.pem" \
  -o "ssl.debug_file: /tmp/wireshark.log" \
  -i eth0 -f "tcp port 5061"
```

TSHARK EXAMPLE:

```
tshark -o "ssl.desegment_ssl_records: TRUE" \
  -o "ssl.desegment_ssl_application_data: TRUE" \
  -o "ssl.keys_list: 4.2.2.2,5061,sip,/opt/ssl/agent.pem" \
  -o "ssl.debug_file:/tmp/tshark.log" \
  -i eth0 \
  -f "tcp port 5061"
```

Information courtesy of FreeSwitch and Kamailio/SER communities

REAL-TIME CAPTURE TOOLS

Terminal Heroes pt II

PCAPSIPDUMP

The old school ways: Dumping SIP Sessions to PCAP files

pcapsipdump is a console tool for dumping SIP sessions and RTP packets *(only when available)* to disk in a fashion similar to "tcpdump -w" by creating a single PCAP per each detected SIP session with optional number filters, for later analysis.

This old-school tool can still be useful for *"one-off"* activities and to temporarily monitor/intercept traffic, but clearly introduces a growing level of complexity when analyzing numerous results over long time ranges or when dealing with busy networks alone.

Capture from eth0 and store all SIP sessions in /tmp/

```
# pcapsipdump -i eth0 -d /tmp/
```

```
pcapsipdump version 0.1.4-trunk
Usage: pcapsipdump [-fpU] [-i <interface>] [-r <file>] [-d <working directory>] [-v level]
 -f   Do not fork or detach from controlling terminal.
 -p   Do not put the interface into promiscuous mode.
 -U   Make .pcap files writing 'packet-buffered' - slower method,
      but you can use partitially written file anytime, it will be consistent.
 -v   Set verbosity level (higher is more verbose).
 -n   Number-filter. Only calls to/from specified number will be recorded
 -t   T.38-filter. Only calls, containing T.38 payload indicated in SDP will be recorded
```

PCAPSIPDUMP: http://sourceforge.net/projects/pcapsipdump/

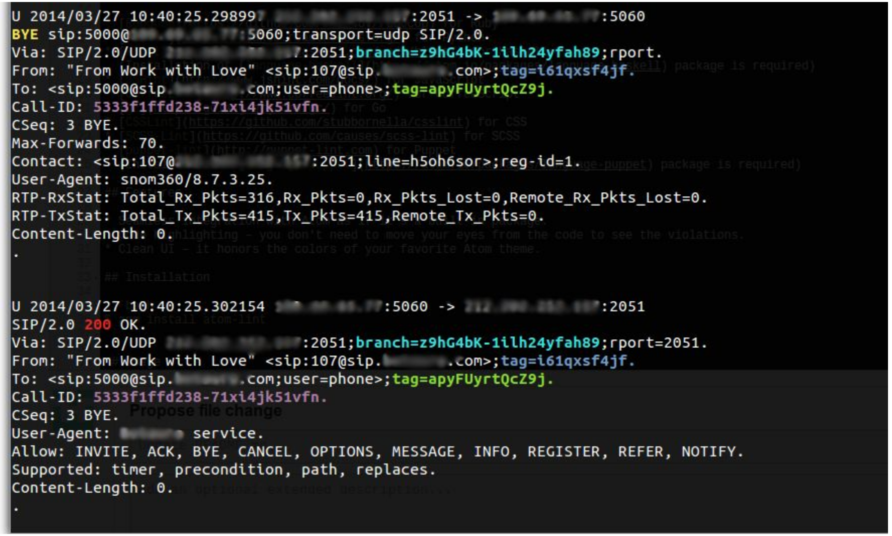
SIPGREP2

CLI Usage and Features (add images)

Sipgrep2 is a modern pcap-aware tool command line tool to capture, filter, display and help troubleshoot SIP signaling over IP networks, allowing the user to specify extended regular expressions matching against SIP headers and with nifty extra features.

Some Handy Examples:

```
# Find a dialog there From user contains '2323232'
sipgrep -f 2323232

# Find a dialog there To user contains '1111' and print dialog
report
sipgrep -f 1111 -G

# Display only 603 replies without dialog match
sipgrep '^SIP/2.0 603' -m

# Display only OPTIONS and NOTIFY requests
sipgrep '^(OPTIONS|NOTIFY)'

# Display only SUBSCRIBE dialog
sipgrep 'CSeq:\s?\d* (SUBSCRIBE|PUBLISH|NOTIFY)' -M

# Collect all messages while pcap_dump smaller than 20kb
sipgrep -q 'filesize:20' -O sipgrep.pcap
```

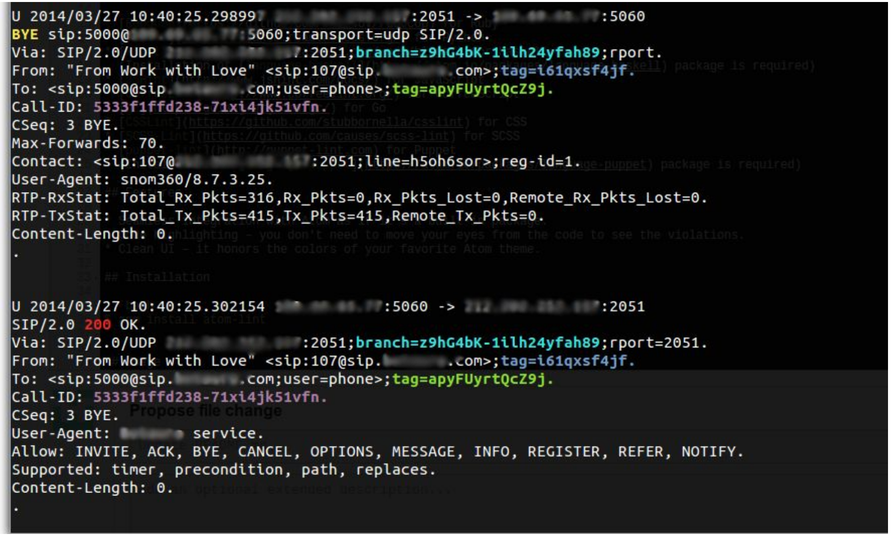


SIPGREP: https://github.com/sipcapture/sipgrep

SIPGREP2
CLI Usage and Features

More Handy Examples:

```
# Kill friendly-scanner automatically
sipgrep -J

# Kill friendly-scanner with custom UAC name
sipgrep -j sipvicious

# Collect all Calls/Registrations dialogs during 120
seconds, print reports and exit:
sipgrep -g -G -q 'duration:120'

# Split pcap_dump to 20 KB files in sipgrep_INDEX_YYYYMMDDHHMM.
pcap
sipgrep -Q 'filesize:20' -O sipgrep.pcap


# Split pcap_dump in sipgrep_INDEX_YYYYMMDDHHMM.pcap each 120
seconds
sipgrep -Q 'duration:120' -O sipgrep.pcap
```



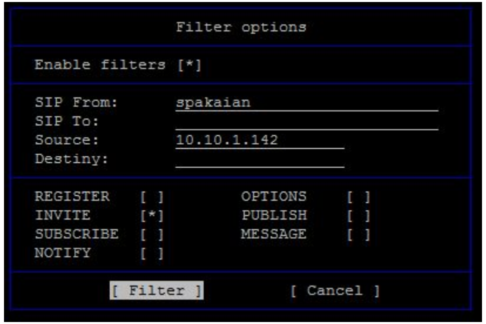
Sipgrep packages are available natively on Debian SID:

`https://packages.debian.org/sid/sipgrep`

SNGREP by Kaian/irontec

Troubleshooting SIP sessions in the terminal… *HEP included!*

sngrep is a great tool for displaying SIP calls message flows from a terminal, exporting HEP3 packets to a HOMER instance and great for watching traffic over multiple local views:



- Call List Window: Allows to select the calls to be displayed
- Call Flow Window: Shows a diagram of source and destiny of messages
- Call Raw Window: Display SIP messages texts
- Message Diff Window: Displays differences between two SIP messages

Display SIP packets from a PCAP file using filters

```
# sngrep -I file.pcap host 192.168.1.1 and port 5060
```

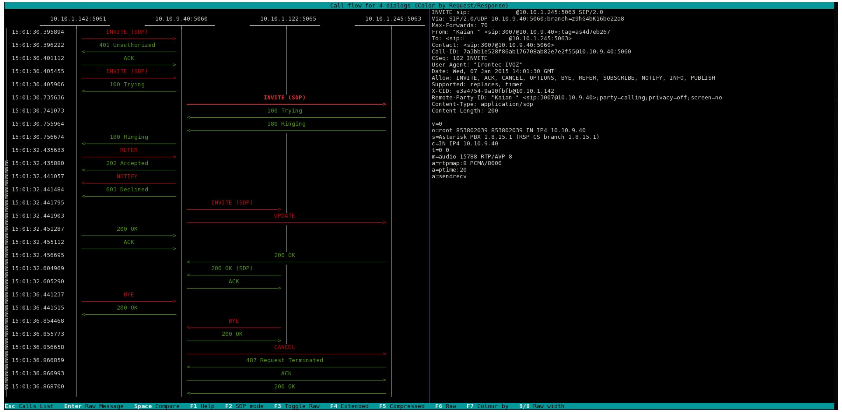
Display Live packets, save to a new PCAP file

```
# sngrep -d eth0 -O save.pcap port 5060 and udp
```

Export HEP3 Encapsulated packets to HOMER *(eep.send)*

```
# sngrep -H -d eth0 port 5060 and udp
```

SNGREP: https://github.com/irontec/sngrep



CENTRALIZED SOLUTIONS

Capture Servers & Long-Term Storage

Centralized Capture Systems

Voice Packets echoing from the Past!

Centralized Capture Systems are generally designed for voice network operators, providers and ITSPs in need of permanent monitoring and troubleshooting resources for their Voice and Customer support and engineering teams on a daily basis and provides key features to protect and maximize voice products and accurately measure infrastructure or peering investments.

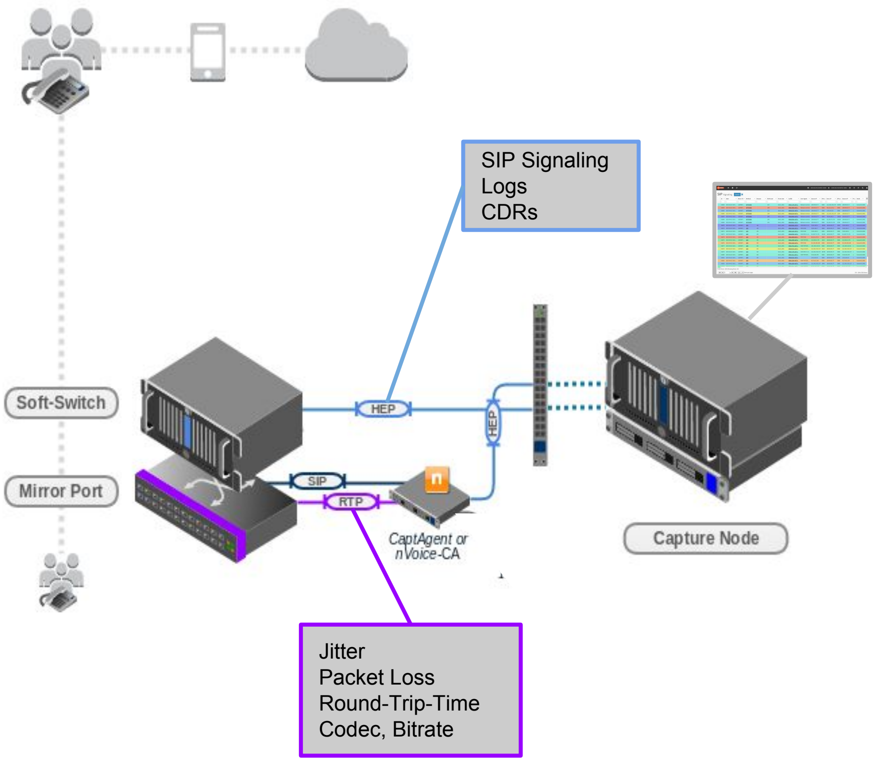
Several commercial and a few free options are available on the market covering this key role, each focusing on different areas but sharing some common advantages:

Key Benefits:

- system/platform agnostic capture viewpoint
- permanent monitoring of service resources
- instant troubleshooting present and past events
- long-term storage of signaling and usage metrics

User Benefits:

- accelerate access to aggregated information
- reduce initial investigation complexity
- reduce unsecured user access to key resources
- empower teamwork in case handling



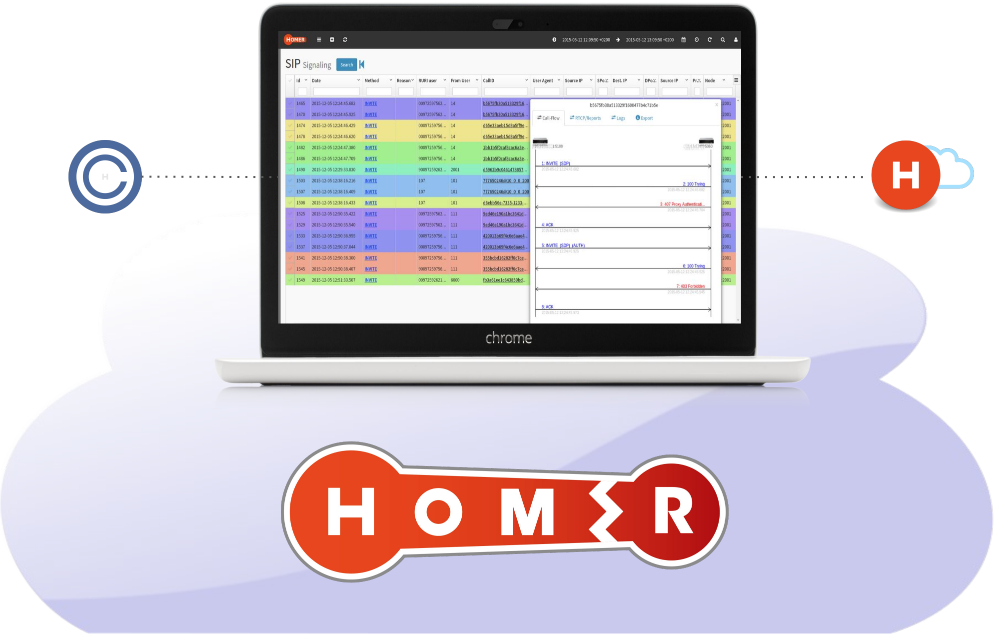
Homer/HEP Example Diagram

CENTRALIZED SOLUTIONS

HOMER + SIPCAPTURE

Proudly Introducing

HOMER 5



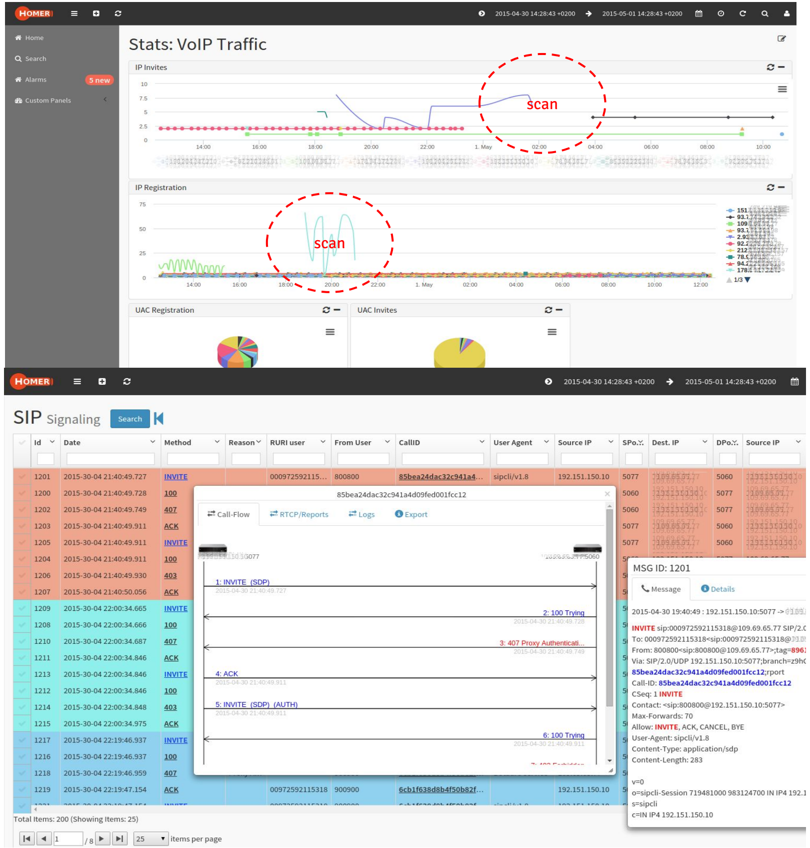
SIPCAPTURE

New to our Projects?

SIPCAPTURE is a powerful suite of tools enabling Voice Engineers to focus on their actual job without having to spend hours figuring how to get the data they need to work with on each instance. Our flagship product **HOMER** is a self-contained SIP Analysis and Troubleshooting environment fully customizable based on the preferences of its users:

HOMER is a turnkey solution providing many advantages:

- Instant centralized access to present and past signaling & stats
- Full SIP/SDP payload with precise timestamping
- Automatic correlation of sessions and reports
- Visual representation of multi session call-flows
- Fast detection of usage and system anomalies
- System agnostic view of VoIP traffic flows
- Unlimited plug & play capture agents and HEP data feeds
- Easy data integration with other systems via API
- No Desktop/Mobile client software required
- Ease of installation *(no more 1st setup headaches!)*



HOMER: http://github.com/sipcapture/homer

SIPCAPTURE

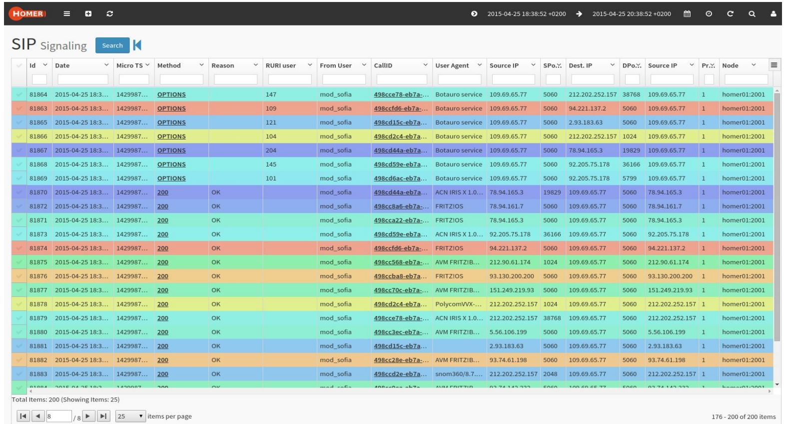
HOMER 5

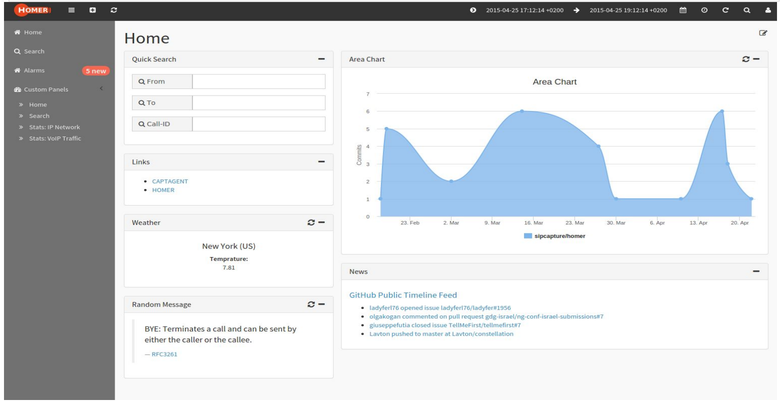
What's New in Homer 5 UI?

HOMER 5 brings many core improvements and module extensions to handle so much more than just signaling, and delivers a complete overhaul of the web User-Interface component migrating to modern JS framework while retaining the simplicity and style many users worldwide rely upon daily.

HOMER's Main Features:

- 100% HTML5 & API Based User Interface
- No Defaults! All Pages and Dashboards fully customizable
- Multiple DB options *(MySQL/MariaDB, PSQL, ElasticS, InfluxDB …)*
- Modern & Extensible Angular Drag & Drop UI
- User Customizable Widgets for Charts & Analytics
- Powerful SIP Search and Filtering functionality
- Native Canvas Call-Flow display with multi-session correlation
- Native support for PCAP and Text file export of all results
- Supports token Authentication for API and User Interface
- Multi-User support with Local, LDAP, Radius options
- Production Ready, supports high volumes and PPS rates
- Supported by a strong and growing community





HOMER: http://github.com/sipcapture/homer

HOMER 5: How does it work?

Build your own HOMER Capture Server using SIPCAPTURE modules

HOMER setups requires two basic building blocks:

CAPTURE SERVER

A *Capture Server Collects, Indexes and Stores* SIP packets received from *Capture Agents* using *HEP v1/2/3*, SBCs using *IPIP* or Raw *SIP* from Ethernet interfaces and mirrored switch ports, using flexible rules defined in the powerful, extensible and fully customizable capture plan

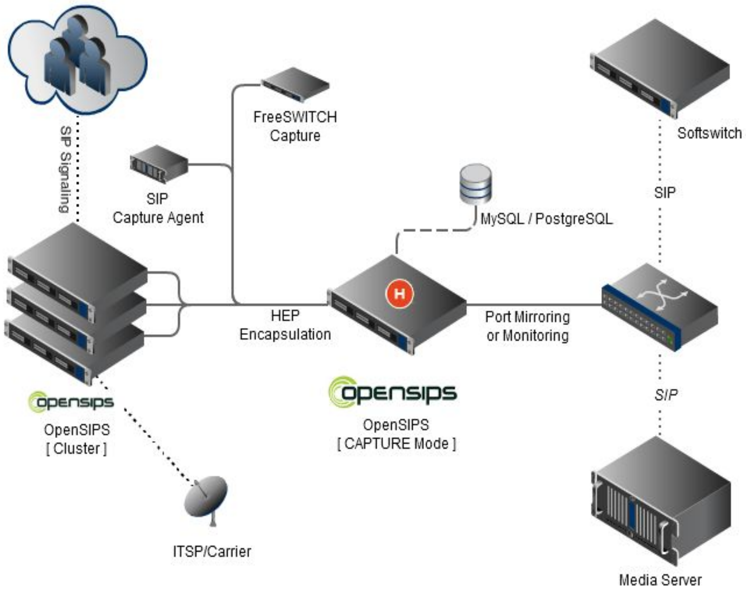
- Requires:OpenSIPS + sipcapture module

CAPTURE AGENT(s)

A Capture Agent sends encapsulated packets to a Capture Server using the *HEP* Encapsulation protocol designed for **HOMER**

The Capture Agent role can be covered by different elements running on different platforms or architectures and distributed in a completely modular fashion, allowing it to support any network topology and complexity and to easily scale with the monitored architectures, as displayed in the illustration on the right.

- *Next Slide: what is HEP?*



SIPCAPTURE

HOMER Capture Server: Capture Plan, Alarms and Statistics configuration

SIPCAPTURE module logic for packet capture, alarms and statistics is completely customizable and extensible with no limits:

```
####### Packet Capture Logic ########


        if(is_method("INVITE|BYE|CANCEL|UPDATE|ACK|PRACK|REFER"))
        {
                $var(table) = "sip_capture_call";
        }
        else if(is_method("REGISTER"))
        {
                $var(table) = "sip_capture_registration";
        }
        else if(is_method("INFO"))
        {
                $var(table) = "sip_capture_call";
        }
        else if(is_method("OPTIONS"))
        {
                $var(table) = "sip_capture_rest";
        }
        else {
                $var(table) = "sip_capture_rest";
        }

        $var(a) = $var(table) + "_%Y%m%d";

        sip_capture("$var(a)");
```

More Examples: https://github.com/sipcapture

```
####### Alarms & Statistic Parameters #########

if (is_method("INVITE|REGISTER")) {

    if($ua =~ "(friendly-scanner|sipvicious)") {
          sql_query("cb", "INSERT INTO alarm_data_mem (create_date, type,
total, source_ip, description) VALUES(NOW(), 'scanner', 1, '$si', 'Friendly
scanner alarm!') ON DUPLICATE KEY UPDATE total=total+1");
          route(KILL_VICIOUS);
    }

    #IP Method
          sql_query("cb", "INSERT INTO stats_ip_mem ( method, source_ip, total)
VALUES('$rm', '$si', 1) ON DUPLICATE KEY UPDATE total=total+1");

     if($au != $null)  $var(anumber) = $au;
     else $var(anumber) = $fU;

     #hostname in contact
     if($sel(contact.uri.host) =~ "^(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})$") {
            if($sht(a=>alarm::dns) == $null) $sht(a=>alarm::dns) = 0;
            $sht(a=>alarm::dns) = $sht(a=>alarm::dns) + 1;
     }
}
```

HOMER 5: What the HEP is HEP?

HEP = Homer Encapsulation Protocol

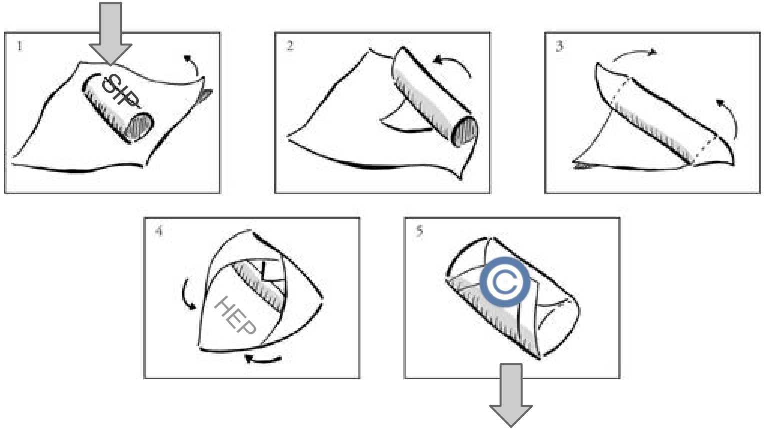
HOMER's own Encapsulation protocol *(HEP/EEP)* is used to wrap and transfer captured packets between a capture Agent and Server.

The HEP Extensible Encapsulation protocol was designed to provide an efficient, modular and low -level framework to accurately duplicate passively obtained IP datagrams for remote collection over *UDP/TCP/SCTP* connections, where full retention of original datagram headers and payload MUST be provided to the collector without alterations or data loss.

The HEP3/EEP definition includes both generic *(internal)* and vendor-specific custom defined chunk types providing ground for implementors to extend the spectrum of the deliverable data within the HEP protocol alongside the encapsulated IP datagram.

HOMER currently supports HEP decoding for *SIP, XMPP, RTCP, RTCP-XR* and *Custom Logs* or *CDRs* in plain text or JSON format.

Find the full specs at: http://github.com/sipcapture/hep

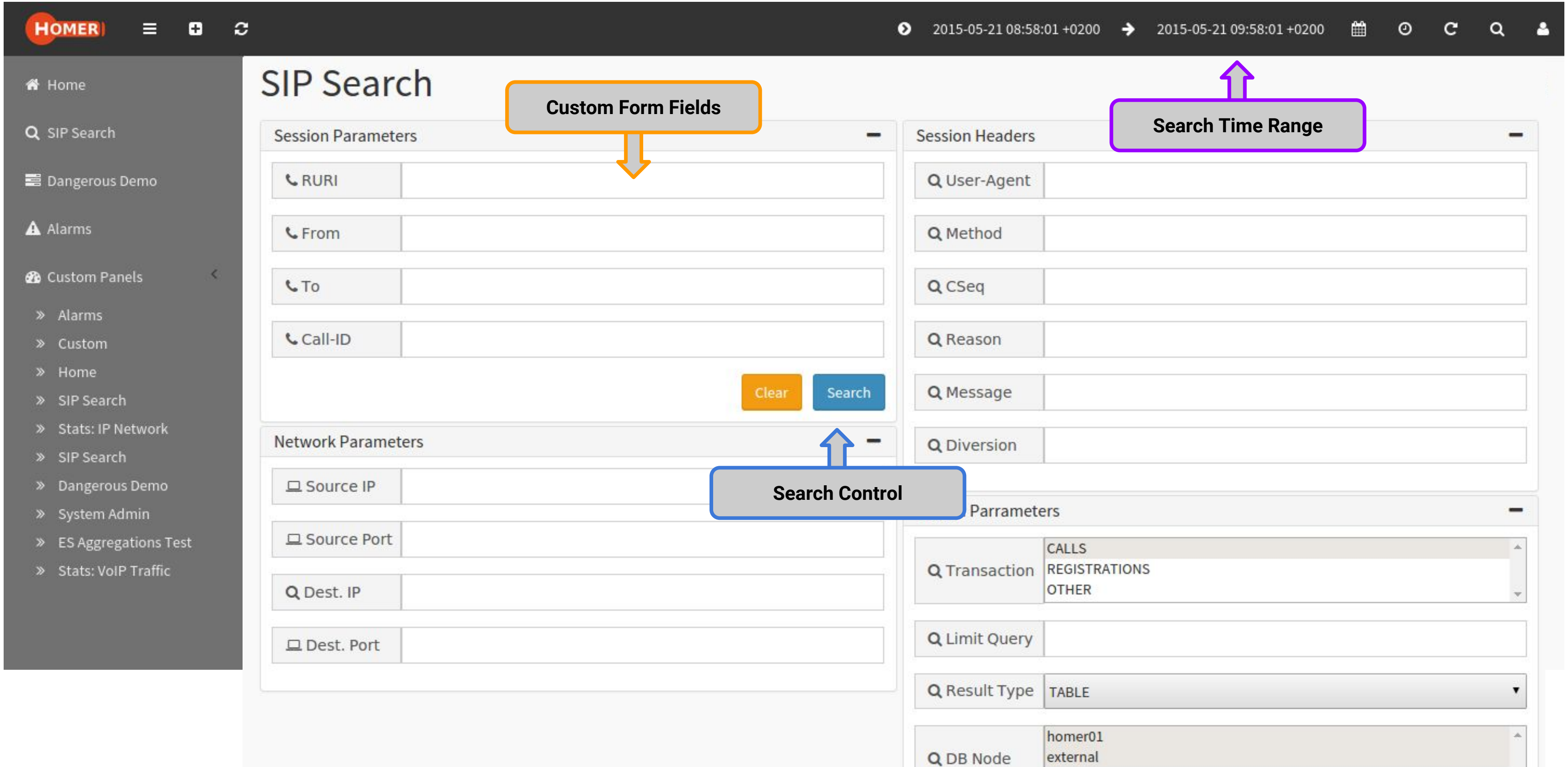


HOMER 5

SIP Search Application

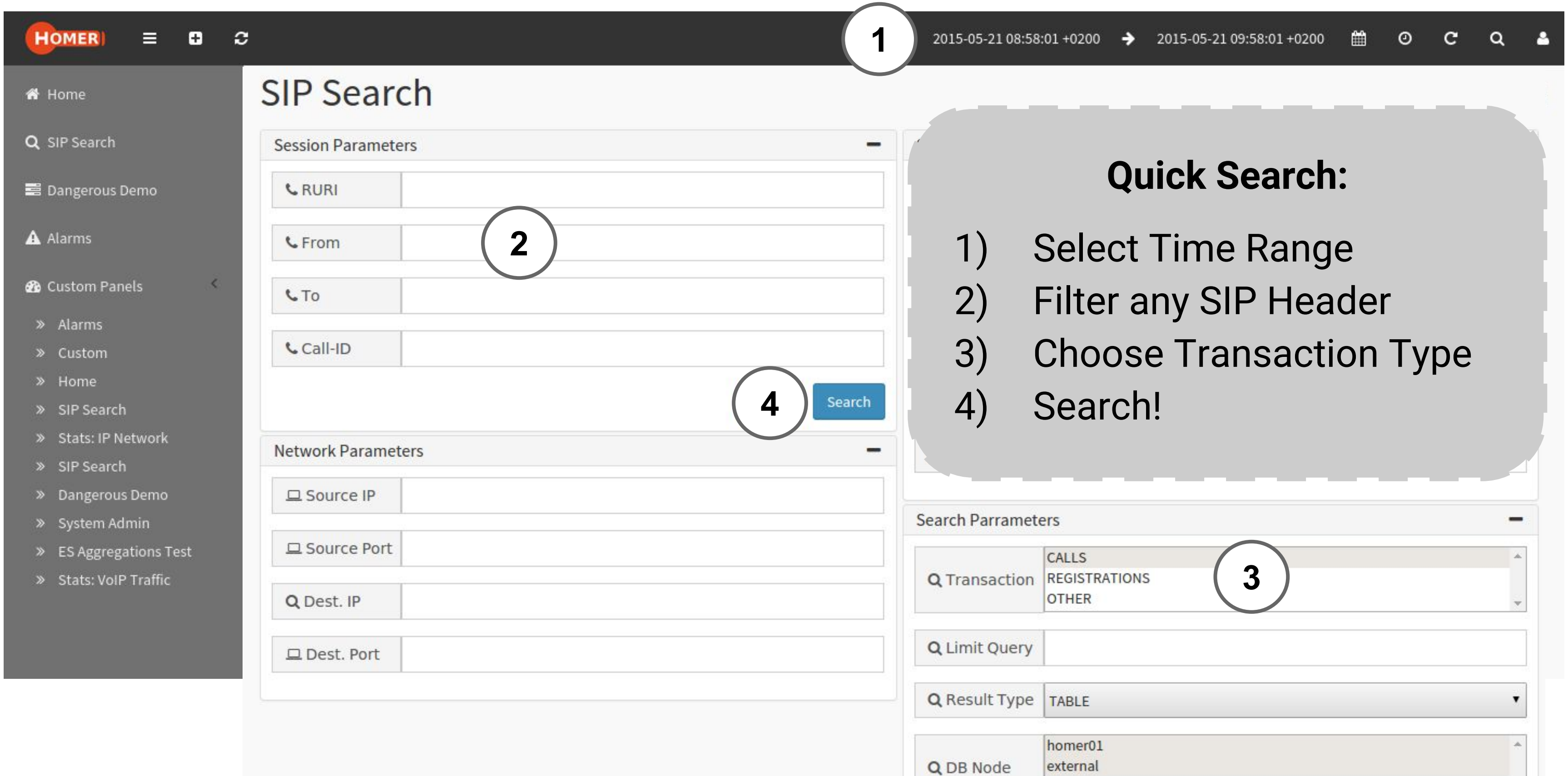
HOMER 5

Your new SIP Search Dashboard is ready to use!



HOMER 5

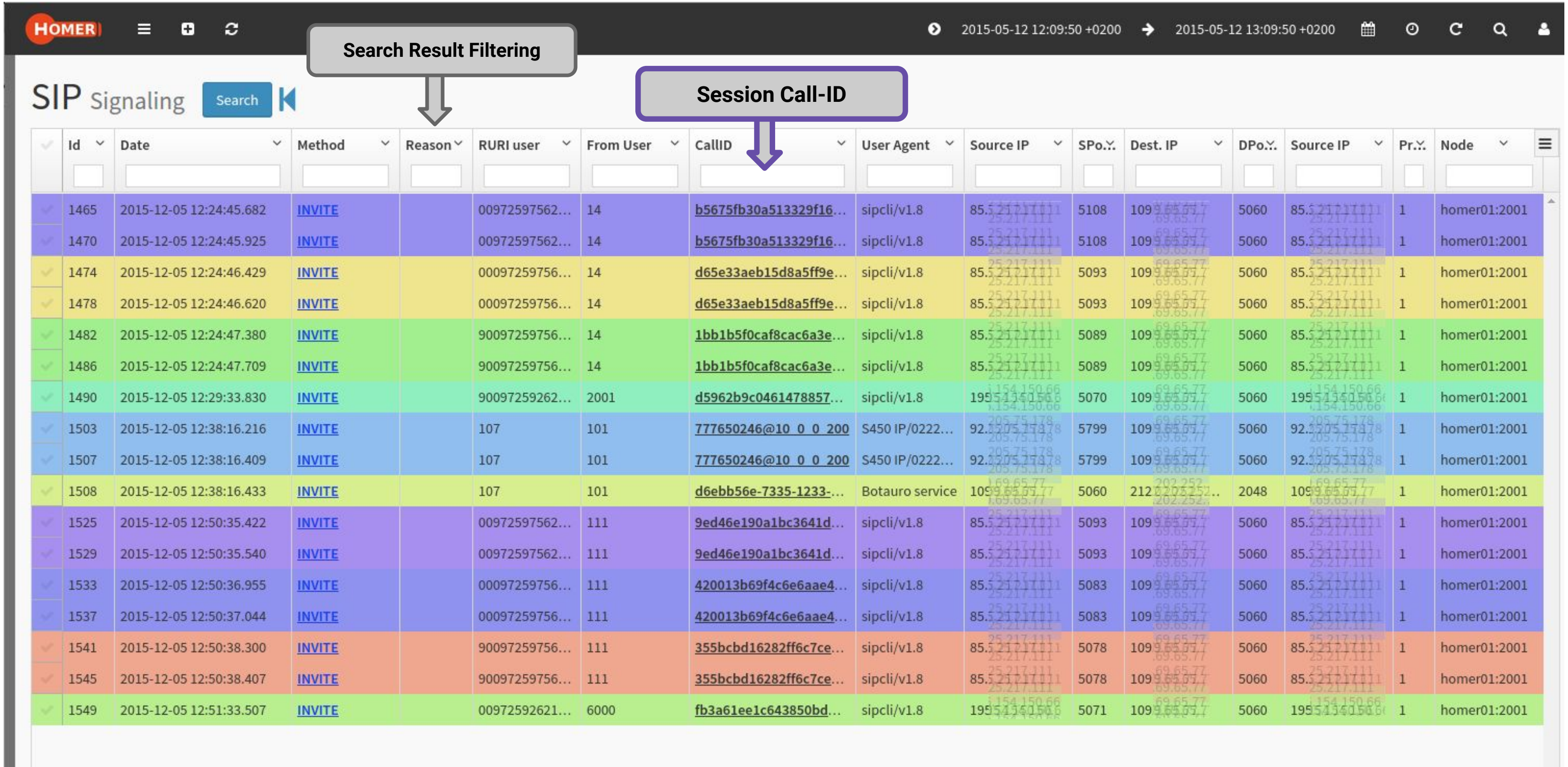
Let's find some SIP traffic next!



HOMER 5

Example: Search Results

1 Find the session of interest

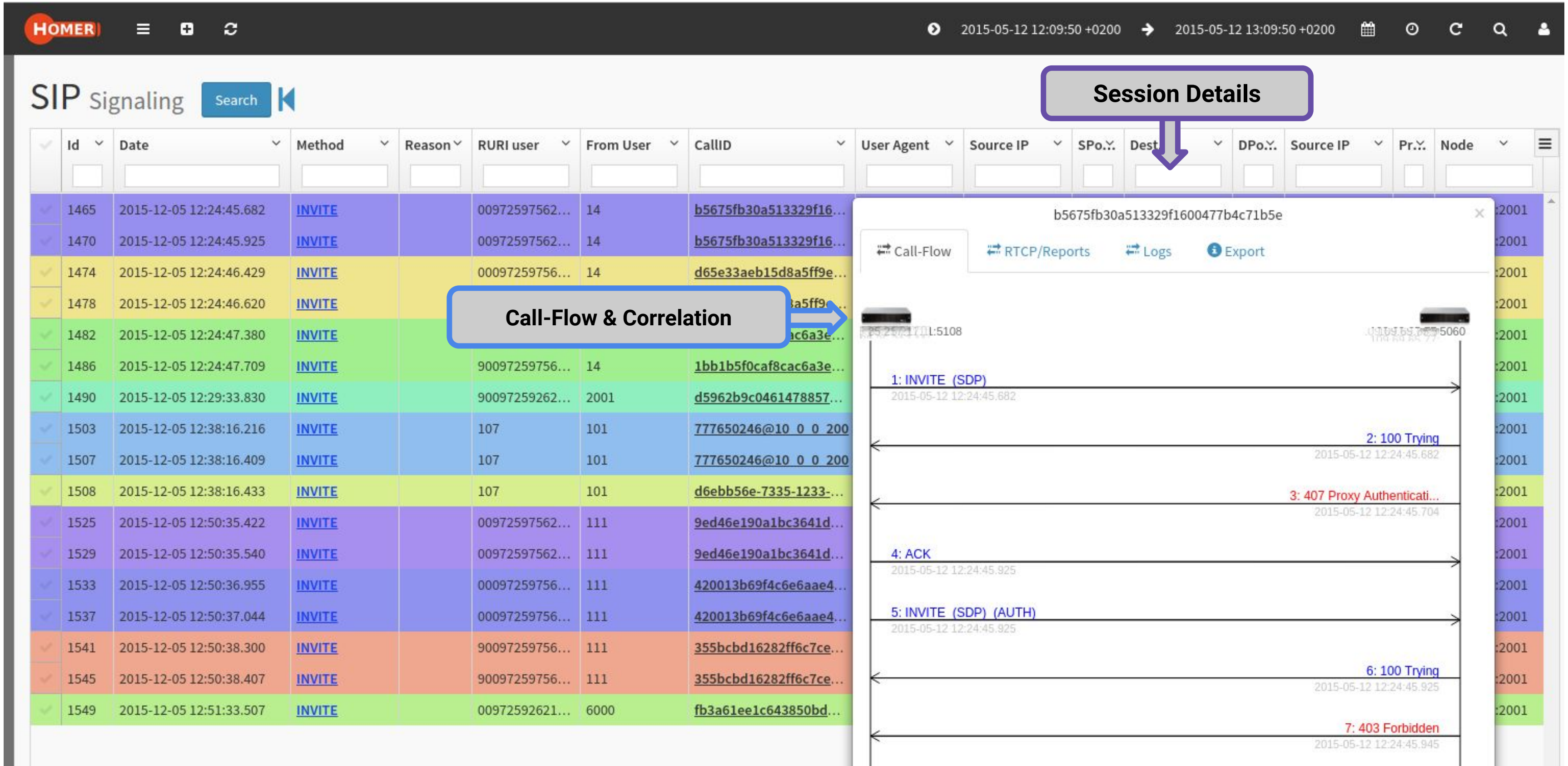


Id	Date	Method	Reason	RURI user	From User	CallID	User Agent	Source IP	SPo...	Dest. IP	DPo...	Source IP	Pr...	Node
1465	2015-12-05 12:24:45.682	INVITE		00972597562...	14	b5675fb30a513329f16...	sipcli/v1.8	85.25.217.111	5108	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1470	2015-12-05 12:24:45.925	INVITE		00972597562...	14	b5675fb30a513329f16...	sipcli/v1.8	85.25.217.111	5108	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1474	2015-12-05 12:24:46.429	INVITE		00097259756...	14	d65e33aeb15d8a5ff9e...	sipcli/v1.8	85.25.217.111	5093	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1478	2015-12-05 12:24:46.620	INVITE		00097259756...	14	d65e33aeb15d8a5ff9e...	sipcli/v1.8	85.25.217.111	5093	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1482	2015-12-05 12:24:47.380	INVITE		90097259756...	14	1bb1b5f0caf8cac6a3e...	sipcli/v1.8	85.25.217.111	5089	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1486	2015-12-05 12:24:47.709	INVITE		90097259756...	14	1bb1b5f0caf8cac6a3e...	sipcli/v1.8	85.25.217.111	5089	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1490	2015-12-05 12:29:33.830	INVITE		90097259262...	2001	d5962b9c0461478857...	sipcli/v1.8	195.154.150.66	5070	109.69.65.77	5060	195.154.150.66	1	homer01:2001
1503	2015-12-05 12:38:16.216	INVITE		107	101	777650246@10_0_0_200	S450 IP/0222...	92.205.75.178	5799	109.69.65.77	5060	92.205.75.178	1	homer01:2001
1507	2015-12-05 12:38:16.409	INVITE		107	101	777650246@10_0_0_200	S450 IP/0222...	92.205.75.178	5799	109.69.65.77	5060	92.205.75.178	1	homer01:2001
1508	2015-12-05 12:38:16.433	INVITE		107	101	d6ebb56e-7335-1233-...	Botauro service	109.69.65.77	5060	212.202.252...	2048	109.69.65.77	1	homer01:2001
1525	2015-12-05 12:50:35.422	INVITE		00972597562...	111	9ed46e190a1bc3641d...	sipcli/v1.8	85.25.217.111	5093	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1529	2015-12-05 12:50:35.540	INVITE		00972597562...	111	9ed46e190a1bc3641d...	sipcli/v1.8	85.25.217.111	5093	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1533	2015-12-05 12:50:36.955	INVITE		00097259756...	111	420013b69f4c6e6aae4...	sipcli/v1.8	85.25.217.111	5083	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1537	2015-12-05 12:50:37.044	INVITE		00097259756...	111	420013b69f4c6e6aae4...	sipcli/v1.8	85.25.217.111	5083	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1541	2015-12-05 12:50:38.300	INVITE		90097259756...	111	355bcbd16282ff6c7ce...	sipcli/v1.8	85.25.217.111	5078	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1545	2015-12-05 12:50:38.407	INVITE		90097259756...	111	355bcbd16282ff6c7ce...	sipcli/v1.8	85.25.217.111	5078	109.69.65.77	5060	85.25.217.111	1	homer01:2001
1549	2015-12-05 12:51:33.507	INVITE		00972592621...	6000	fb3a61ee1c643850bd...	sipcli/v1.8	195.154.150.66	5071	109.69.65.77	5060	195.154.150.66	1	homer01:2001

HOMER 5

Example: Session Details

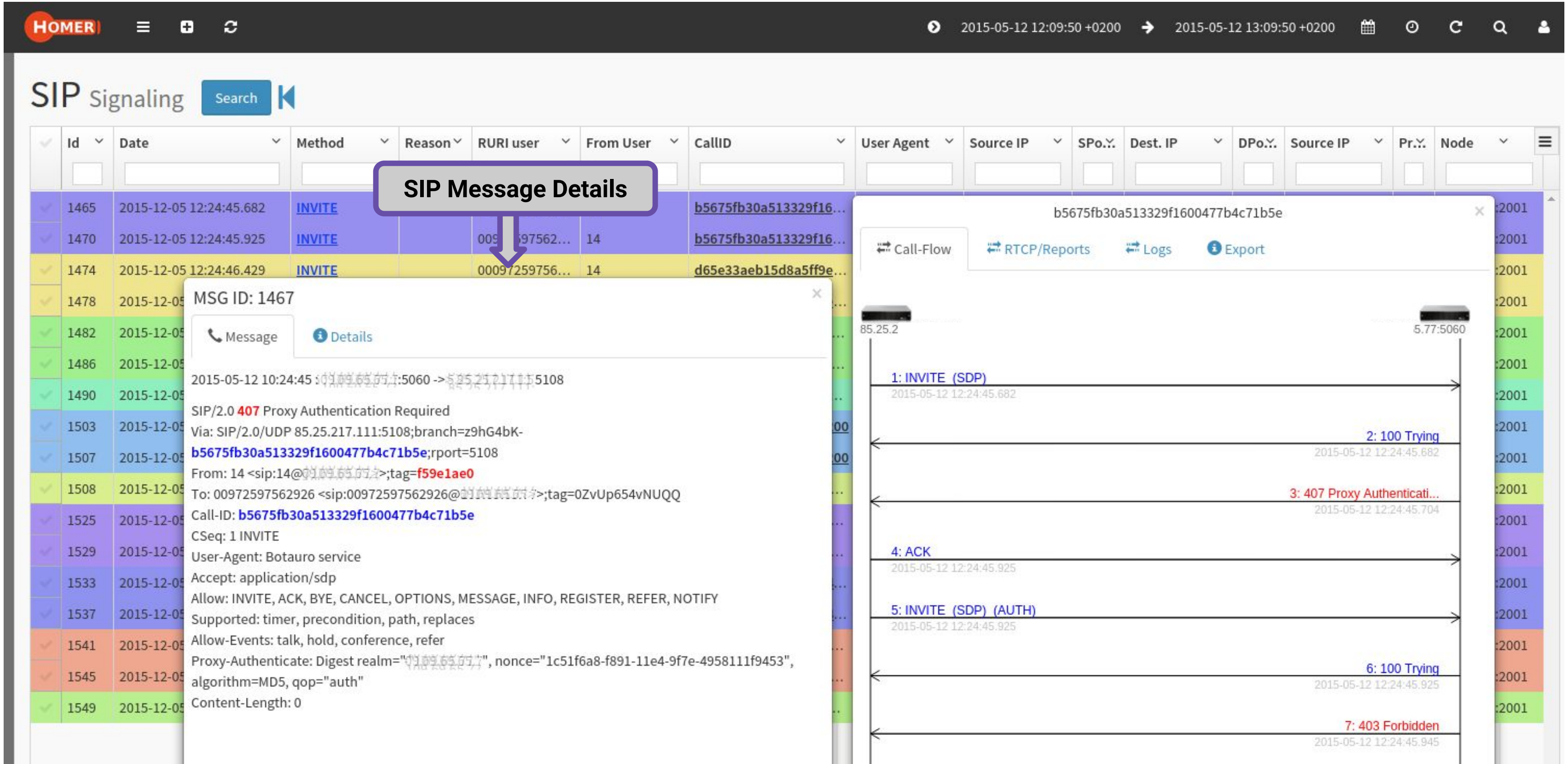
2 Click a Call-ID to correlate a Session



HOMER 5

Example: Session and Packet Details

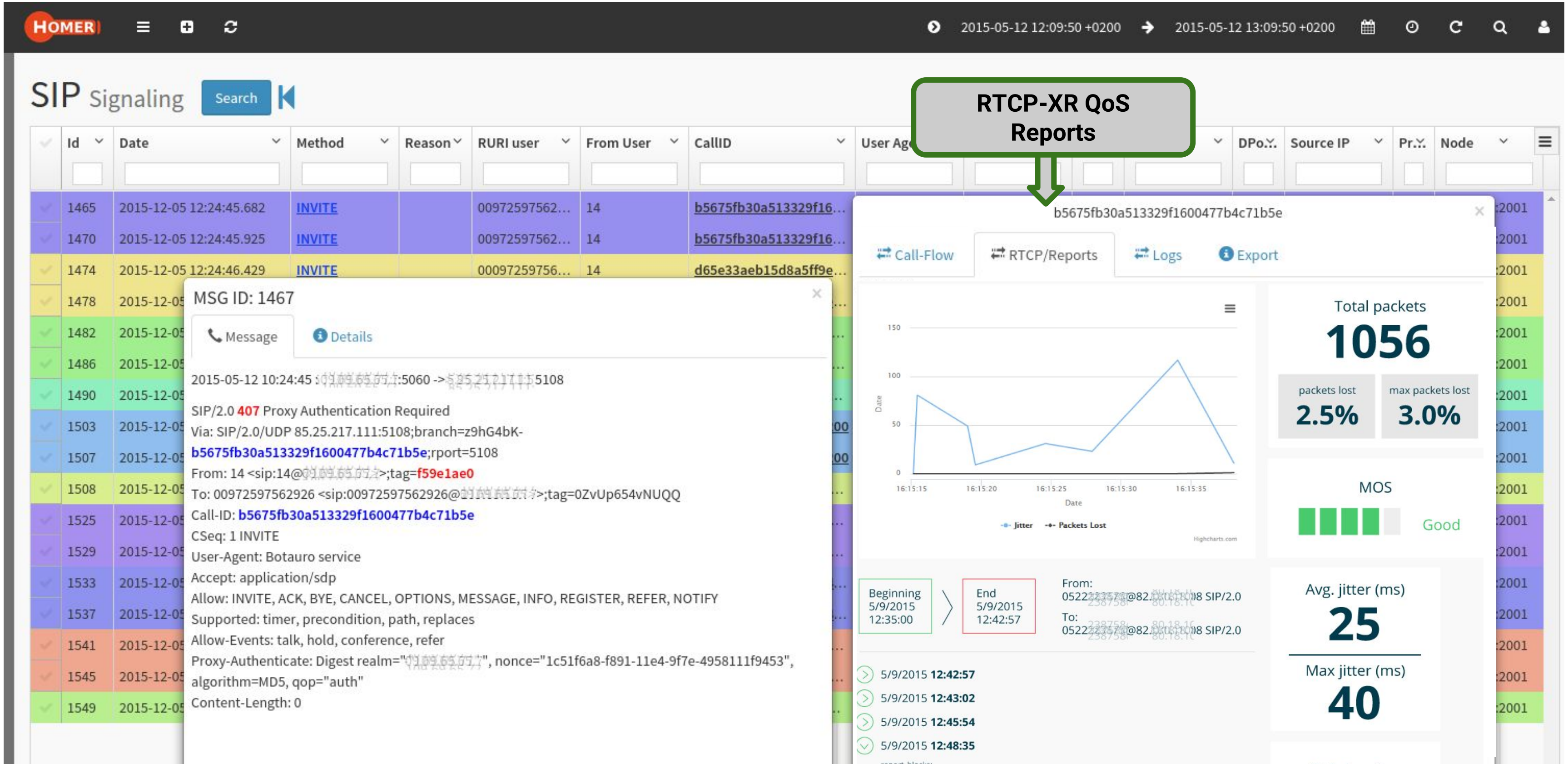
3 Click & Inspect any SIP Message



HOMER 5

Example: Session and Packet Details

4 Click & Inspect RTCP-XR Reports



SIPCAPTURE

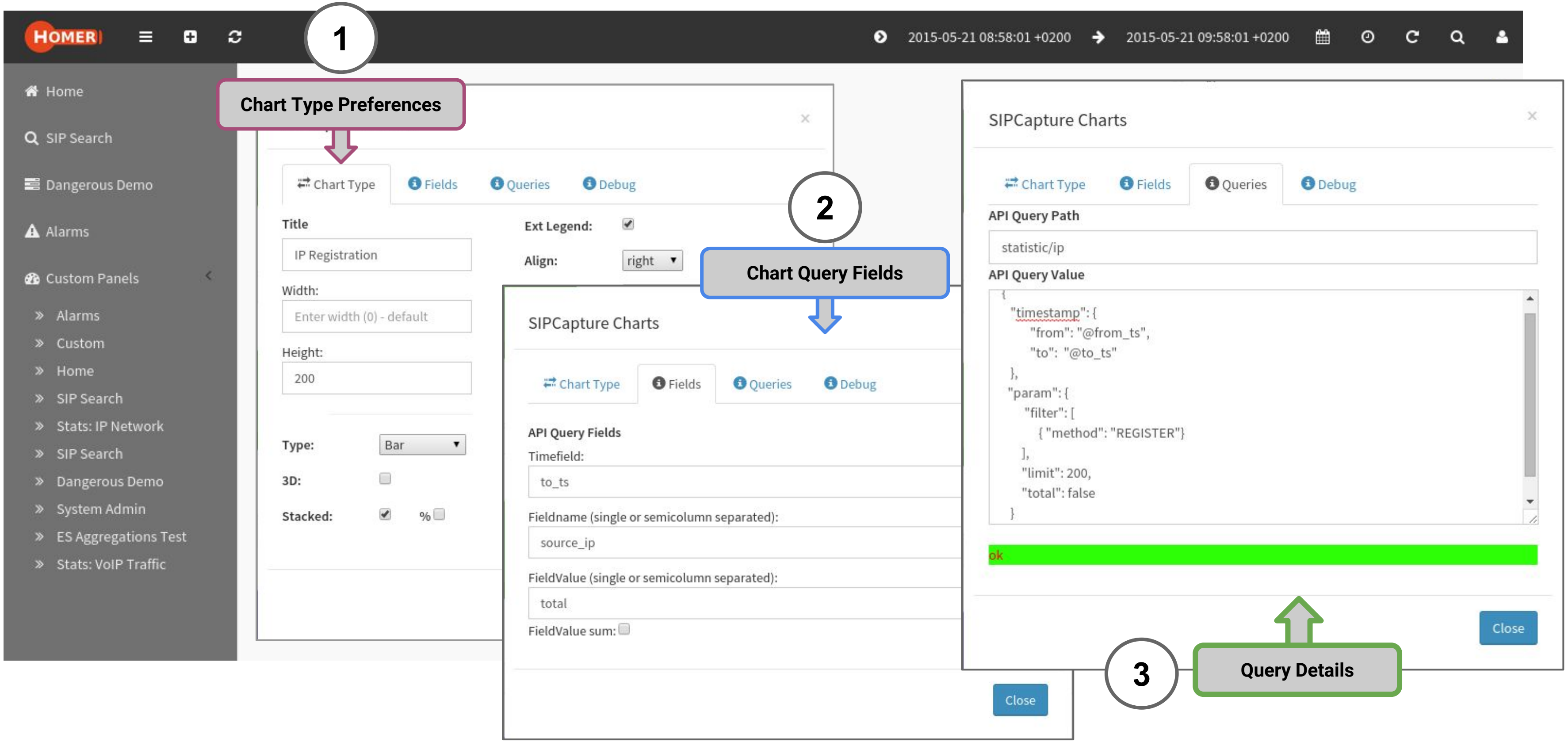
HOMER 5

Got Charts?

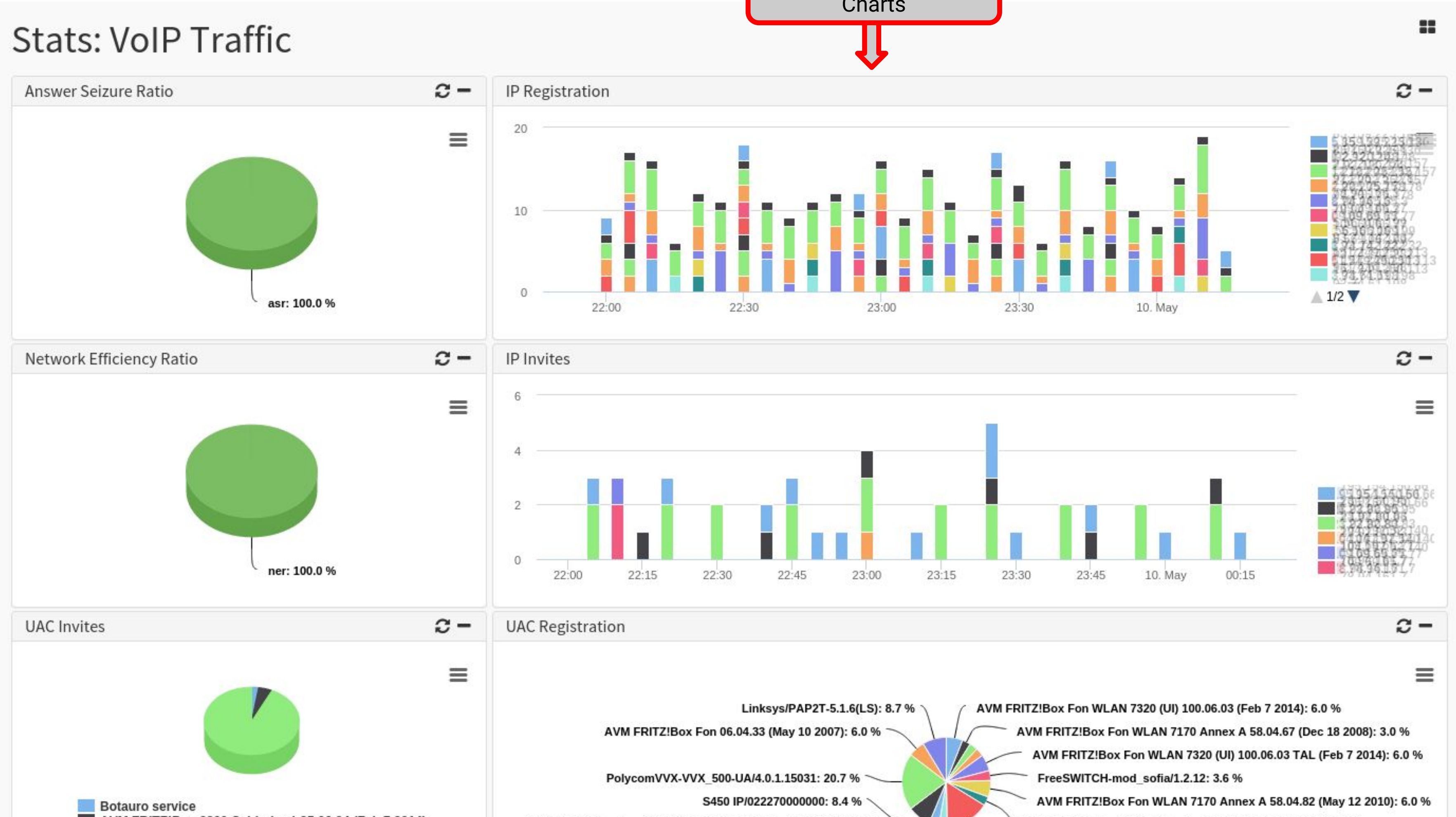
H

HOMER 5

Create a Stats Dashboard in seconds



SIPCapture API Charts



Stats: IP Network

HOMER Capture Server using OpenSIPS: QoS Reports and Logging

```
# PUBLISH REPORT

if(is_method("PUBLISH") && has_body("application/vq-rtcpxr"))
{
     $var(table) = "report_capture";
     $var(callid) = $(rb{re.subst,/(.*)CallID:([0-9A-Za-z@-]{5,120})(.*)$/\2/s});

     $var(temp) = $(rb{re.subst,/^(.*)JitterBuffer:(.*)JBN=([0-9]{1,5})(.*)$/\3/s});
     if(float2int("$var(temp)", 1)) $var(jbn) = $rc;

      #Mos
      $var(temp) = $(rb{re.subst,/^(.*)QualityEst:(.*)MOSCQ=([0-9.]{1,4})(.*)$/\3/s});
      if(float2int("$var(temp)", 10)) $var(mos) = $rc;

      statsd_set($var(customer)+"Mos", $var(mos));
      statsd_set($var(customer)+"JBN", $var(jbn));

      #save to db
      report_capture("$var(table)", "$var(callid)");

      drop;
}
```

More Examples: https://github.com/sipcapture

RTCP-XR provides a range of VoIP call and network quality metrics generated by user agents and devices supporting the protocol. The reports can be very useful to debug the user quality of a given session and are supported by HOMER. RTCP-XR packets can be handled in two different ways by a capture agent:

- ***STORE*** Mode
  Using HEP proto_id 99 QoS reports are sent to DB

- ***FORWARD*** Mode
  Using HEP SIP proto_id, QoS reports are forwarded to kamailio.cfg where users can parse and extract relevant information for statistical purposes and store to internal hashmap, Homer DB, or statsd module

HINT: Don't miss our QoS Dangerous Demo!

References:

- RFC 3611 *(RTP Control Protocol Extended Reports)*
- RFC 6035 *(SIP Package for Voice Quality Reporting)*

OpenSIPS + flatstore recipe

On-demand, long-term archiving of SIP signaling

This configuration option instructs the sipcapture module to use the flatstore db module which is configured to create all of its files in the "*/db/homer_dat*" directory - *note such directory must exist and have write permissions for the process user!*

```
modparam("sipcapture", "db_url", "flatstore:/db/homer_data")
```

Define sip_capture table as:

```
$var(table) = “sip_capture_%Y%m%d%H%M.flat”
```

and each hour we start bzip2 inside this table and move to special directory:

```
find /db/homer_data -type f -name “*.flat” -exec “movefile.sh” {} \;
```

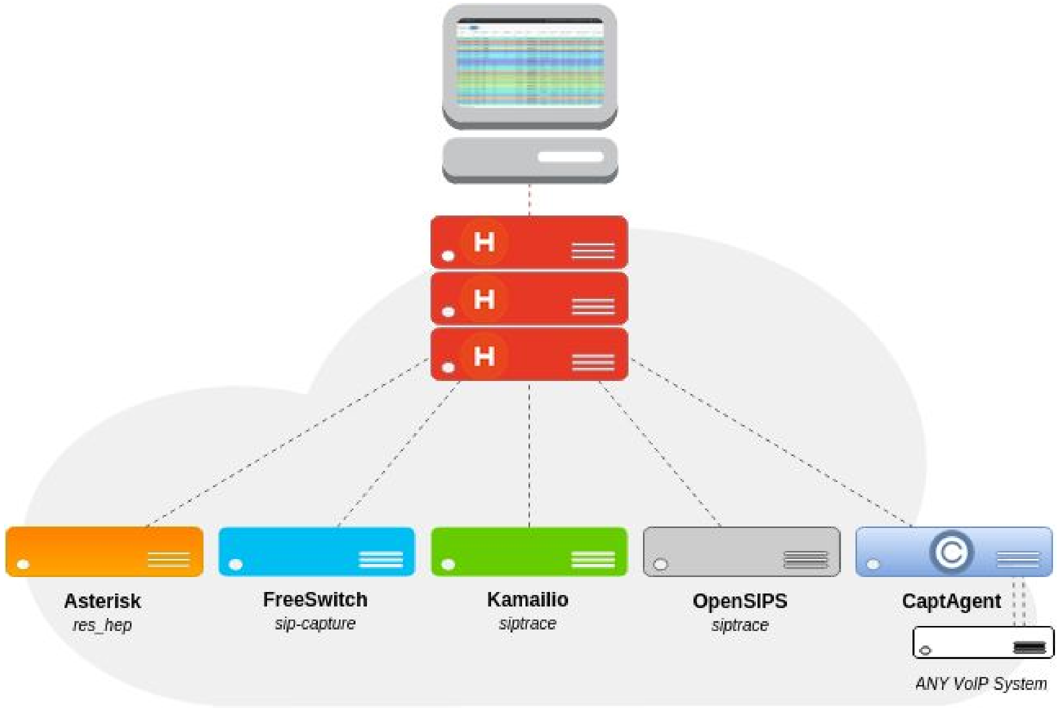
movefile.sh

```
#!/bin/sh
FILE=$1
bzip2 -kv9 $FILE
mv $(FILE).bz2
/db/homer_bzip/
```

Flatstore files can be restored to a local mysql DB if and when necessary.
A dedicated node connector can also be defined from Homer's UI and used for searches on demand.

HEP

Capture Agents



HEP - Homer Encapsulation Protocol

Integrated Capture Agents in OSS Platforms

HOMER's own encapsulation protocol *(HEP/EEP)* is used to transfer your packets unmodified and carries several key information in its headers designed for perfect capturing.

HEP agents have been consistently integrated across leading OSS solutions - chances are you have one in your fleet already!

The following projects provide integrated HEP support:

- *Kamailio*
- *OpenSIPS*
- *FreeSWITCH*
- *Asterisk*
- *sipXecs*

Examples are also provided for the following languages:

- *C/C++*
- *Java*
- *Javascript / Node.JS*
- *Erlang*
- *Go*

The *HEP/EEP* Protocol is defined in a mature Draft pending submission and is freely available for developers to integrate.

Find more about HEP: http://hep.sipcapture.org/

Other HEP Agents

```
OpenSIPS Example:
```

https://github.com/sipcapture/homer/wiki/Examples%3A-OpenSIPS

```
Kamailio Example:
```

https://github.com/sipcapture/homer/wiki/Examples%3A-Kamailio

```
FreeSWITCH Example:
```

https://github.com/sipcapture/homer/wiki/Examples%3A-FreeSwitch

```
CaptAgent Example:
```

https://github.com/sipcapture/homer/wiki/Examples%3A-Captagent4

```
nProbe VoIP Example:
```

https://github.com/sipcapture/homer/wiki/Examples%3A-nProbe

```
ACME SBC Example:
```

https://github.com/sipcapture/homer/wiki/Examples%3A-ACME-Packet

CAPTAGENT 6

Modular Capture Agent w/ HEP3 Support

Captagent started as a SIP-only capture agent for HOMER. The codebase over time has been completely redesigned from the ground up to follow the evolution of the **HEP** protocol and **Captagent** grew to become a powerful, flexible, completely modular capture agent *framework* ready for virtually any kind of protocol and encapsulation method, past, present - *and future*.

Currently available modules:

- UNI Proto Module
  - SIP, XMPP and other text signaling Protocols
- RTCP Module
  - RTCP and RTCP-XR Parser and Collector
- CLI Module
  - CLI Shell Access and control of Captagent
- HEP Module
  - HEP Encapsulation output (v1/2/3)
- SSL/TLS Module
  - Encryption and Compression Module for HEP3

Upcoming modules:

- Remote API Module
  - Configure and Control a feet of Captagents from a Central server

CAPTAGENT: https://github.com/sipcapture/captagent

```
<!-- CORE MODULES -->

   <configuration name="core_hep.conf" description="HEP Socket">
      <settings>
        <param name="version" value="3"/>
        <param name="capture-host" value="capture.server.org"/>
        <param name="capture-port" value="9060"/>
        <param name="capture-proto" value="udp"/>
        <param name="capture-id" value="2001"/>
        <param name="capture-password" value="myHep"/>
        <param name="payload-compression" value="false" />
       </settings>
   </configuration>


<!-- PROTOCOLS -->

        <configuration name="proto_uni.conf" description="UNI Proto Basic
capture">
            <settings>
              <param name="port" value="5060"/>
              <!-- <param name="portrange" value="5060-5090"/> -->
              <!--
                  use -D flag for pcap import
                  use "any" for all interfaces in your system
              -->
              <param name="dev"  value="eth0"/>
              <param name="promisc" value="true"/>
              <!-- comment it if you want to see all IPProto (tcp/udp)  -->
              <param name="ip-proto" value="udp"/>
              <param name="proto-type"  value="sip"/>
              <!-- <param name="filter" value="not src port 5099"/> -->
            </settings>
        </configuration>
```

Example: Captagent 6 programming

```
<module name="socket_pcap" description="HEP Socket" serial="2014010402">
 <profile name="socketspcap_sip" description="HEP Socket" enable="true"
serial="2014010402">
        <settings>
           <param name="dev" value="any"/>
           <param name="promisc" value="true"/>
           <param name="reasm" value="false"/>
           <param name="capture-plan" value="sip_capture_plan.cfg"/>
           <param name="filter">
                      <value>portrange 5060-5091</value>
           </param>
       </settings>
</profile>
<profile name="socketspcap_rtcp" description="RTCP Socket" enable="true"
serial="2014010402">
        <settings>
           <param name="dev" value="any"/>
           <param name="promisc" value="true"/>
           <param name="reasm" value="false"/>
           <param name="capture-plan" value="rtcp_capture_plan.cfg"/>
           <param name="filter">
                      <value>portrange 30000-50000</value>
           </param>
       </settings>
</profile>
</module>
```

More Examples: https://github.com/sipcapture

```
#sip_capture_plan.cfg
capture[pcap] {

   # here we can check source/destination IP/port, message size
   if(msg_check("size", "100")) {

       #Do parsing
       while(parse_sip()) {

           /* many packets */
           clog("NOTICE", "parsing SIP message ");

           if(source_ip("10.0.0.1")) {
                   #Can be defined many profiles in transport_hep.xml
                    if(!send_hep("hepsocket_homer01")) {
                         clog("ERROR", "Error sending HEP!!!!");
                    }
          }
          else {
                    #Can be defined many profiles in transport_hep.xml
                   if(!send_hep("hepsocket_homer02")) {
                         clog("ERROR", "Error sending HEP!!!!");
                   }
          }

           #Duplicate all INVITEs to JSON transport
           if(sip_is_method() && sip_check("method","INVITE")) {

                #Can be defined many profiles in transport_json.xml
                if(!send_json("jsonsocket")) {
                     clog("ERROR", "Error sending JSON");
                }
           }
       }
  }
  drop;
}
```

Information courtesy of Homer and SER communities

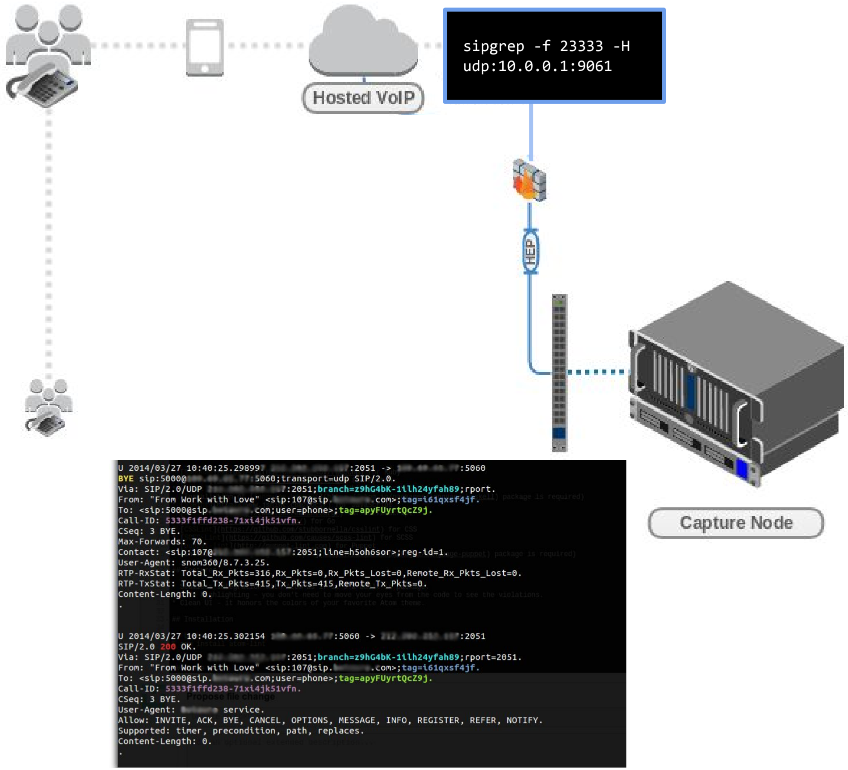
SIPCAPTURE

SIPGREP2

Sipgrep as disposable HEP3 Agent

Sipgrep is able to act as a quick on-demand HEP3 capture agent and forward packets to a collector very easily when a simple terminal check does not suffice.

In the following example, Sipgrep is used to display the traffic of interest as well as log it to a remote location, useful for instance when troubleshooting issues on hosted platforms or disposable instances on the cloud.



HEP3 Example:

Display dialogs and duplicate all traffic to HOMER sipcapture in HEPv3:

```
sipgrep -f 23333 -H udp:10.0.0.1:9061
```

NPROBE SIP Mirroring

Capture & Mirror SIP Signaling using nProbe/nVoice SIP Plugin

NTOP **nProbe** *(w/ VoiP PRO Plugin)* can act as *HEP3* capture agent for SIP Protocol mirroring to a centralized collector such as Homer and can perform this task at high packet rates. The HEP3 features are simply controlled by the following switches:

--hep <host>:<port> | Send JSON flows via HEPv3 protocol
--hep-auth <capture id>:<password> | Specify the HEP authentication parameters.

Example HEP3 SIP Syntax:

```
# nprobe -T "%SIP_CALL_ID" --drop-flow-no-plugin -i eth0    --hep 10.0.10.20:9063 --hep-auth 10:myhep123 -b 0 -G
```

NTOP nProbe SIP Plugin can also send out its SIP detections via JSON, NetFlow, or dump logs locally for server-less, ad-hoc implementations or simple batch processing:

--sip-dump-dir <dump dir> | Directory where SIP logs will be dumped
--sip-exec-cmd <cmd> | Command executed whenever a directory has been dumped

NPROBE VoIP: http://ntop.org

%SIP_CALL_ID	SIP call-id
%SIP_CALLING_PARTY	SIP Call initiator
%SIP_CALLED_PARTY	SIP Called party
%SIP_RTP_CODECS	SIP RTP codecs
%SIP_INVITE_TIME	SIP time (epoch) of INVITE
%SIP_TRYING_TIME	SIP time (epoch) of Trying
%SIP_RINGING_TIME	SIP time (epoch) of RINGING
%SIP_INVITE_OK_TIME	SIP time (epoch) of INVITE OK
%SIP_INVITE_FAILURE_TIME	SIP time (epoch) of INVITE FAILURE
%SIP_BYE_TIME	SIP time (epoch) of BYE
%SIP_BYE_OK_TIME	SIP time (epoch) of BYE OK
%SIP_CANCEL_TIME	SIP time (epoch) of CANCEL
%SIP_CANCEL_OK_TIME	SIP time (epoch) of CANCEL OK
%SIP_RTP_IPV4_SRC_ADDR	SIP RTP stream source IP
%SIP_RTP_L4_SRC_PORT	SIP RTP stream source port
%SIP_RTP_IPV4_DST_ADDR	SIP RTP stream dest IP
%SIP_RTP_L4_DST_PORT	SIP RTP stream dest port
%SIP_RESPONSE_CODE	SIP failure response code
%SIP_REASON_CAUSE	SIP Cancel/Bye/Failure reason cause
%SIP_C_IP	SIP C IP adresses
%SIP_CALL_STATE	SIP Call State

MEDIA QUALITY STATISTICS

RTP & RTCP Analysis

RTP Statistics

Network and Media quality probing using RTP, RTCP, RTCP-XR, RTP Reports...

In order to capture, investigate and analyze media stream quality and network issue, we need to interact with the protocols involved with transmission, controlling and reporting of media streams - We should all all be familiar with the following:

RTP (Real-time Transport Protocol)

The aim of RTP is to provide a uniform means of transmitting data subject to real time constraints over IP (audio, video, etc.). The principal role of RTP is to implement the sequence numbers of IP packets to reform voice or video information even if the underlying network changes the order of the packets. More generally, RTP makes it possible to: identify the type of information carried, add temporary markers and sequence numbers to the information carried, monitor the packets' arrival at the destination. RTP works over UDP and its header carries synchronization and numbering information such as sequence number, timestamp and unique identifier for the source.

RTCP (Real-time Transport Control Protocol)

RTCP is a protocol associated with RTP based on periodic transmissions of control packets by all participants in the session and used provide different types of information and a return regarding the quality of reception.

RTCP-XR (Real-time Transport Control Protocol Extended Reports)

Extended Report (XR) packet type for the RTP Control Protocol (RTCP) are used to convey information beyond what is already contained in the reception report blocks of RTCP sender report (SR) or Receiver Report (RR) packets, such as internal statistics about the stream quality and network conditions encapsulated in various types of SIP PUBLISH/OPTIONS reports sent by enabled endpoints during and after a call session.

X-RTP-Stats, P-RTP-Stat (User Agent generated End of Call Statistics)

The Reporting of End-of-Call QoS Statistics in Session Initiation Protocol (SIP) BYE Message feature enabled user-agents to send quality statistics to a remote end when a call terminates. The call statistics are sent as a new header included in the BYE message or in the 200 OK response, and include Real-time Transport Protocol (RTP) packets sent or received, total bytes sent or received, total number of packets that are lost, delay jitter, round-trip delay, call duration and more, providing the endpoint view over the call performance.

RTCP-XR Statistics

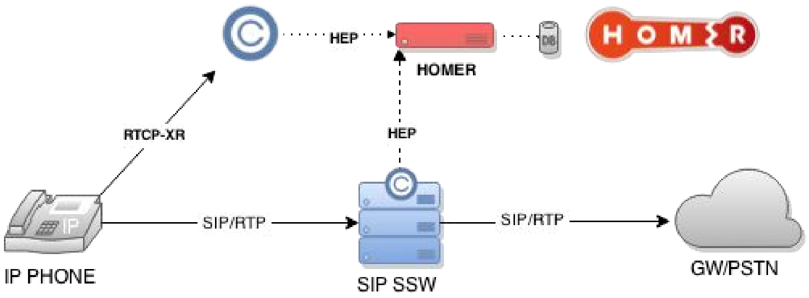
CaptAgent as RTCP-XR Collector or Reporter

<u>How can we use RTCP-XR to troubleshoot call quality?</u>

CaptAgent 6 features a powerful *RTCP-XR* collector module.

RTCP-XR enabled User-Agents *(Snom, Cisco, Polycom, etc)* can directly use **captagent** as a quality report collector. The dedicated module will forward an HEP encapsulated RTCP-XR report to your capture server (such as Homer or PCapture) for later analysis and correlation with the call sessions they belong with and indexed for general statistical purposes.

Captagent can also collect raw RTCP packets and send them as HEP3 or JSON/RAW format to a capture server and can also optionally generate and transmit final RTCP-XR reports *(CallTerm)* including RTP statistics generated for the call duration including Jitter, Delay, Packet Loss and so on, performing an RTCP-> RTCP-XR format adaption/conversion

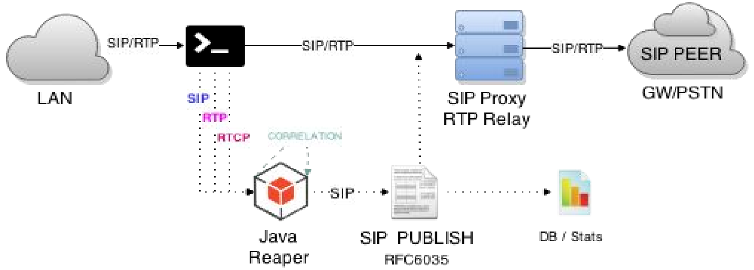



```
PUBLISH SIP/2.0

From: <sip:446@intern.snom.de>;tag=45hkui59ns
To: <intern.snom.de>;tag=nohhk4xu21
Call-ID: 3c26a8de500f-12ct7zov3kjs
CSeq: 3 PUBLISH
Max-Forwards: 70
Contact: <sip:446@192.168.5.251:2060;transport=tls;line=w2wuvhk9>;reg-id=1
Event: vq-rtcpxr
Accept: application/sdp, message/sipfrag
Content-Type: application/vq-rtcpxr
Content-Length: 832

VQSessionReport
LocalMetrics:
Timestamps:START=2010-02-17T13:59:42Z STOP=2010-02-17T13:59:46Z
SessionDesc:PT=0 PD=G.711U PPS=50 SSUP=off
CallID:3c26a8de500f-12ct7zov3kjs
x-UserAgent:snom360/8.2.sf
FromID:<sip:446@intern.snom.de>
ToID:<sip:447@intern.snom.de;user=phone>
LocalAddr:IP=192.168.5.251 PORT=62754 SSRC=0xCBE3450E
RemoteAddr:IP=192.168.0.233 PORT=54018 SSRC=0xB80B52F3
DialogID:3c26a8de500f-12ct7zov3kjs;to-tag=866ed0cf03;from-tag=45hkui59ns
x-SIPmetrics:SVA=RG SRD=310 SFC=0
x-SIPterm:SDC=OK
JitterBuffer:JBA=0 JBR=0 JBN=0 JBM=0 JBX=65535
PacketLoss:NLR=0.0 JDR=0.0
BurstGapLoss:BLD=0.0 BD=0 GLD=0.0 GD=6569 GMIN=16
Delay:RTD=0 ESD=0 IAJ=4
RemoteMetrics:
JitterBuffer:JBA=0 JBR=0 JBN=0 JBM=0 JBX=0
PacketLoss:NLR=0.0 JDR=0.0
BurstGapLoss:BLD=0.0 BD=0 GLD=0.0 GD=4677 GMIN=16
Delay:RTD=0 ESD=0 IAJ=2
```

RTP Statistics

SIP Voice Quality Report **Reaper** (java)

The **Reaper** is a java tool is designed to sniff **SIP/RTP/RTCP** packets *(using a modified tcpdump agent pipe)* and generate correlated voice quality reports in accordance with *RFC6035* forwarding the media stream statistics into the SIP signaling flow for post-processing.



RTCP Reports are processed as forwarded as received:

★ RTCP → VQIntervalReport → SIP PUBLISH

RTP Final Statistics are released once the call is Terminated:

★ RTP → VQSessionReport → SIP PUBLISH

In order to work the Reaper depends on a modified tcpdump binary forwarding packets to special queues feeding the Java process. This makes this solution only suitable for small, custom setups.

REAPER Github: https://github.com/TerryHowe/SIP-Voice-Quality-Report-Reaper
RFC6035: https://tools.ietf.org/html/rfc6035

```
PUBLISH sip:collector@127.0.0.1:5999;transport=udp SIP/2.0.
Call-ID: f1f90855d85e9c874a0dd8e3b14bc607@127.0.0.2.
CSeq: 1 PUBLISH.
From: "reaper" <sip:reaper@127.0.0.2:5070>;tag=ReaperV1.0.
To: "collector" <sip:collector@127.0.0.1:5999>.
Via: SIP/2.0/UDP 127.0.0.2:5070;branch=reaperv1.0-
f1f90855d85e9c874a0dd8e3b14bc607-127.0.0.2-1-publish-127.0.0.2-5070333031.
Max-Forwards: 70.
Contact: "reaper" <sip:reaper@127.0.0.2:5070>.
Content-Type: application/vq-rtcpxr.
Content-Length: 451.
.
VQSessionReport : CallTerm.
LocalMetrics:.
SessionDesc:PT=8 PD=PCMA SR=8000.
Timestamps:START=2015-02-28T21:04:31.000582Z STOP=2015-02-28T21:04:36.000638Z.
CallID:1233727184.
FromID:<sip:caller@domain.net>.
ToID:<sip:callee@domain.net>.
OrigID:<sip:caller@domain.net>.
LocalAddr:IP:192.168.1.23 PORT:7079.
LocalMAC:99:72:b9:28:c2:82.
RemoteAddr:IP:192.168.1.55 PORT:30539.
RemoteMAC:99:e6:ba:df:7b:dd.
PacketLoss:NLR=4.6.
Delay:IAJ=166.
```

RTCP Statistics

Asterisk RTCP Statistics

The latest **Asterisk** patch developed by Alexandr Dubovikov and Matt Jordan implements module *res_hep_rtcp*

The module performs RTCP packet capturing for the internal RTP engine in Asterisk and transmits HEP3 encapsulated call quality metrics & statistics in HEP encapsulated JSON format.

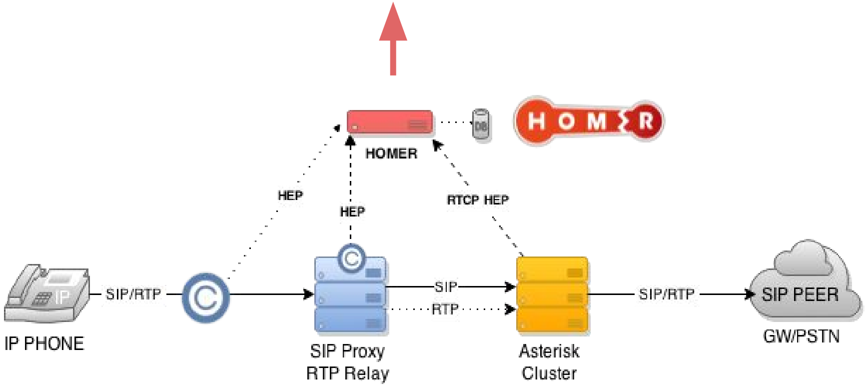
The module can be coupled with *res_hep* to build a full HEP capture node and send SIP signaling as well as call QoS.

With the above setup, statistics can be observed historically and in real time as they reach the server when observing a call including pseudo-MOS score calculated on the client-side.

Example HOMER integration is presented on the side slide:

For more information and patch details:

https://github.com/sipcapture/homer/tree/master/asterisk_rtcp_patch



RTP Statistics

SIP User Agent: End-of-Call Reports

The Reporting End-of-Call Statistics in Session Initiation Protocol (SIP) BYE Message feature enables user-agents to send call statistics to a remote end when a call itself terminates. The call statistics are sent as a new header in the BYE message or in the 200 OK message *(response to BYE message)*.

The statistics include Real-time Transport Protocol (RTP) packets sent or received, total bytes sent or received, total number of packets that are lost, delay jitter, round-trip delay, and call duration.

Commonly implemented SIP headers are **X-RTP-Stat** and **P-RTP-Stats** and the less complex **RTP-RxStat** / **RTP-TxStat**

```
X-RTP-Stat:
PS=207;OS=49680;;PR=314;OR=50240;PL=0;JI=600;LA=40;
```

```
The X-RTP-Stat header contains the following fields:

PS=<voice packets sent>
OS=<voice octets sent>
PR=<voice packets received>
OR=<voice octets received>
PL=<receive packet loss>
JI=<jitter in ms>
LA=<latency in ms>
```

Specs: https://www.avm.de/de/Extern/files/x-rtp/xrtpv32.pdf

```
P-RTP-Stat: PS=326,OS=52160,PR=318,OR=50880,PL=0,JI=0,LA=0,DU=7,
EN=G711a,DE=G711a
```

```
The P-RTP-Stat header contains the following fields:

PS=<Packets Sent>
OS=<Octets Sent>
PR=<Packets Recd>
OR=<Octets Recd>
PL=<Packets Lost>
JI=<Jitter>
LA=<Round Trip Delay in ms>
DU=<Call Duration in seconds>
EN=<Audio Encoder>
DE=<Audio Decoder>
```

RTP Statistics

RTPProxy Statistics injection into P-RTP-Stat Header

Although RTP Statistics are to be generated by the UA/client in order to be fully meaningful, **RTPProxy** can still provide back its own *internal rtp statistics* *(as seen by the relay)* to be included in *BYE / 200 OK* messages using the data sent back to the SIP Proxy core by RTPProxy module, and formatted in a **P-RTP-Stat** compatible header.

Additional information can be injected into the header from database queries or other local or external sources.

A pseudo basic example script extension could look as follows:

```
## Pseudo P-RTP-Stats snippet for RTPProxy

                 if (is_method("BYE")) {
                                setflag(FLT_ACC); # do accounting ...
                                setflag(FLT_ACCFAILED); # ... even if the transaction fails

                                $var(xrtpstat) = $(rtpstat{s.striptail,1});

                                # Work the new stats
                                $var(rtp0) = $(var(xrtpstat){s.select,1, });
                                $var(rtp1) = $(var(xrtpstat){s.select,2, });
                                $var(rtp2) = $(var(xrtpstat){s.select,3, });
                                $var(rtp3) = $(var(xrtpstat){s.select,4, });
                                $var(rtp4) = $(var(xrtpstat){s.select,5, });
                                if ($var(rtp0) != "" || $var(rtp1) != "")
                                {
                                 append_hf("P-RTP-Stat:  EX=RTPProxy,PS=$var(rtp0),PR=$var(rtp1),PL=$var(rtp3)\r\n");
                                }
                 }
```

RTP Statistics at Wire-Speed

nProbe RTP Plugin w/ Pseudo-MOS Estimation

NTOP **nProbe** *(w/ VoIP RTP Plugin)* can produce granular RTP Statistics for network streams detected via nDPI and is able perform full SIP session report bi-directional correlation and codec aware Pseudo-MOS/R-Factor estimations, all exportable at user defined sample rates via JSON over TCP or HTTP/S to a centralized collector.

Example RTP Plugin Syntax:

```
# nprobe -T "%IPV4_SRC_ADDR %L4_SRC_PORT %IPV4_DST_ADDR %L4_DST_PORT %PROTOCOL %
RTP_IN_JITTER %RTP_OUT_JITTER %RTP_IN_PKT_LOST %RTP_OUT_PKT_LOST %
RTP_IN_PAYLOAD_TYPE %RTP_OUT_PAYLOAD_TYPE %SIP_CALL_STATE %RTP_SIP_CALL_ID %
SIP_CALL_ID %RTP_RTT %RTP_MOS %RTP_R_FACTOR %IN_PKTS %OUT_PKTS %RTP_IN_TRANSIT %
RTP_OUT_TRANSIT %RTP_RTT" --redis 127.0.0.1 --drop-flow-no-plugin -i eth1 -b 3 --
json-labels  -t 30 --hep 10.0.10.20:9063--hep-auth 10:myhep123 -b 0 -G
```

Example RTP Statistics:


```
{"FIRST_SWITCHED":1411119211,"IPV4_SRC_ADDR":"1.2.2.222","L4_SRC_PORT":11034,"
IPV4_DST_ADDR":"1.1.1.233","L4_DST_PORT":37308,"PROTOCOL":17,"RTP_IN_JITTER":2391,"
RTP_OUT_JITTER":475,"RTP_IN_PKT_LOST":1,"RTP_OUT_PKT_LOST":0,"RTP_IN_PAYLOAD_TYPE":
18,"RTP_OUT_PAYLOAD_TYPE":18,"RTP_SIP_CALL_ID":"h8A02kd73jdc","IN_PKTS":729,"
OUT_PKTS":240,"IN_BYTES":43740,"OUT_BYTES":24000,"RTP_IN_TRANSIT":1135,"
RTP_OUT_TRANSIT":11,"RTP_RTT":0,"L7_PROTO_NAME":"RTP","RTP_MOS":435,"RTP_R_FACTOR":
9033,"TOTAL_FLOWS_EXP":19731}
```


NPROBE VoIP: http://ntop.org

%RTP_FIRST_SSRC	First flow RTP Sync Source ID
%RTP_FIRST_TS	First flow RTP timestamp
%RTP_LAST_SSRC	Last flow RTP Sync Source ID
%RTP_LAST_TS	Last flow RTP timestamp
%RTP_IN_JITTER	RTP jitter (ms * 1000)
%RTP_OUT_JITTER	RTP jitter (ms * 1000)
%RTP_IN_PKT_LOST	Packet lost in stream (src->dst)
%RTP_OUT_PKT_LOST	Packet lost in stream (dst->src)
%RTP_IN_PAYLOAD_TYPE	RTP payload type
%RTP_OUT_PAYLOAD_TYPE	RTP payload type
%RTP_IN_MAX_DELTA	Max delta (ms*100) between pkts (src->dst)
%RTP_OUT_MAX_DELTA	Max delta (ms*100) between pkts (dst->src)
%RTP_SIP_CALL_ID	SIP call-id corresponding to this RTP stream
%RTP_MOS	RTP pseudo-MOS (value * 100)
%RTP_R_FACTOR	RTP pseudo-R_FACTOR (value * 100)
%RTP_IN_TRANSIT	RTP Transit (value * 100) (src->dst)
%RTP_OUT_TRANSIT	RTP Transit (value * 100) (dst->src)
%RTP_RTT	RTP Round Trip Time (ms)
%RTP_DTMF_TONES	DTMF tones sent (if any) during the call
%SIP_RTP_CODECS	SIP RTP codecs
%SIP_RTP_IPV4_SRC_ADDR	SIP RTP stream source IP
%SIP_RTP_L4_SRC_PORT	SIP RTP stream source port
%SIP_RTP_IPV4_DST_ADDR	SIP RTP stream dest IP
%SIP_RTP_L4_DST_PORT	SIP RTP stream dest port

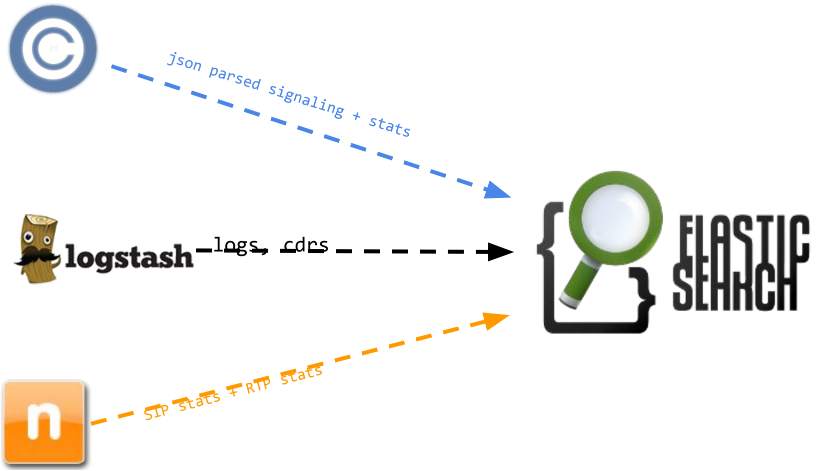
Voice CDRs & LOGS

Elasticsearch + CaptAgent / nVoice

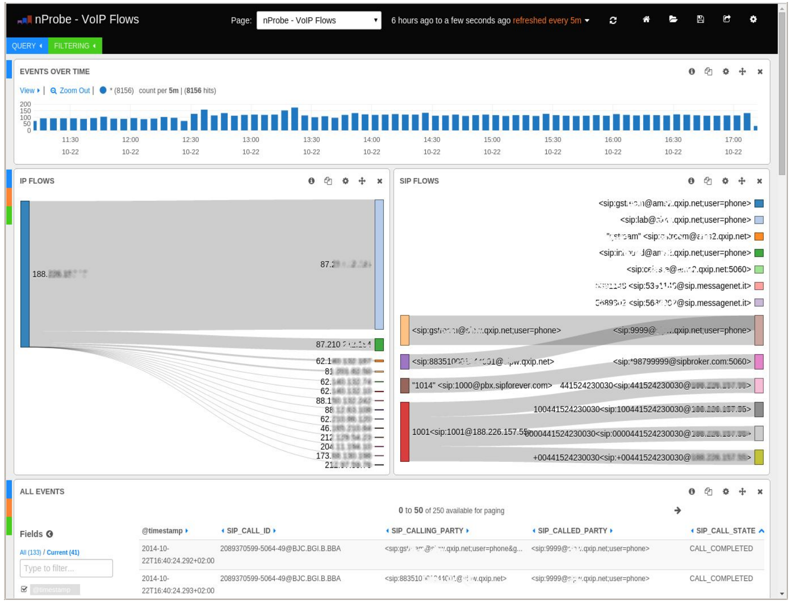
Already collecting metrics in Elasticsearch or any other JSON-centric backend? *Good News!* You can integrate your voice statistics to your existing data infrastructure with very little work with minimal technical efforts and investment.

CDRs and System logs can now be aggregated with their network counterpart adding a further dimension to your data.

CAPTAGENT + JSON Module



NPROBE + ES/JSON + VoIP Plugin



Voice CDRs & LOGS

Experiment with **HEPipe**

Troubleshooting is not all about network packets - many times system logs will hold valuable pointers at internal issues not expressed at the protocol level. There are many tools able to forward syslog/rsyslog to notorious collectors but for those looking to build their own voice data collection, we have developed a HEP3 playground utility called **HEPipe**

HEPipe *(pronounced HEP-pipe)* is an application for logging arbitrary data *(ie: logs, cdrs, debug lines)*to a *HEP/EEP* capture server such as HOMER or PCAPTURE via command pipe.

The utility can be used to prototype HEP3 implementations as well as to feed real data into a HEP Collector for real life usage, for instance by using the session Call-ID as correlation parameter.

```
INPUT FORMAT:

timestamp_sec; timestamp_usec; correlation_id; source_ip; source_port; destination_ip; destinaton_port; payload in json


USAGE EXAMPLE:

echo '1396362930;1003;18731b65be;127.0.0.1;5060;10.0.0.1;5060;{"pl": 10, "jt": 10}'|./hepipe -s hepserver -p 9061 -t 100
```

AUTOMATED TESTS

Friendly Probes

SIP Testing with Scripted Agents
PJSUA and SIPSAK

pjsua can be used as a simple call generator to test SIP Trunk or equipment availability:

```
# pjsua < (echo "sleep 2000;M;20;sip:192.168.1.10;sleep 10000;ha;sleep 5000;quit;")
```

pjsua can be launched in daemon mode and configured to act as a playback auto-responder:

```
# pjsua –null-audio –play-file=data3.wav –auto-play –auto-answer=200 –config-file=pj-config
```

```
--id sip:1234@domain.com
--registrar sip:domain.com
--username 1234
--password password
--realm asterisk
--null-audio
--auto-play
--play-file /tmp/audio.wav
```

sipsak is perfectly suitable for simple tests such as sending a single OPTION probe:

```
# sipsak -vv -s sip:192.168.1.10:5060
```

sipsak can also send customer methods (NOTIFY Event: check-sync;reboot=true causing yealink phone to reboot):

```
# sipsak -f reboot_yealink.sipfile -s sip:1234@192.168.1.10
```

sipsak is ideal for Nagios usage: http://exchange.nagios.org/directory/Plugins/Network-Protocols/*-VoIP/SIP/check_sip-sipsak/details

(we use this ourselves since 2002 and still up)

SIP Testing with quality-aware Agents

BARESIP User-Agent w/ X-RTP-Stats

Baresip is a modular open-source (BSD) user agent built on top of LibRE/LibREM by Alfred E. Heggstad

One of our contributions to the project was the ability to export the valuable internal stream/codec details and statistics *(Jitter, Packet Loss, Payload details, etc)* by implementing *X-RTP-Stat* header export in BYE/200 OK SIP Messages.

This enables Baresip being used as a *"quality probing" SIP* user-agent (or echo-test agent) with call-quality results efficiently distributed alongside the session closure methods, featured in many existing brand Hardware SIP Phones.

Test Calls can be automatically scheduled (or triggered via HTTP Command API) and results collected by existing systems.

Header Example:

```
X-RTP-Stat: EX=BareSip;CS=0;CD=152;PR=7383;PS=7635;PL=0,0;PD=0,0;JI=0.8,0.1;EN=PCMU/8000;DE=PCMU/8000;IP=A.B.C.D:4202,E.F.G.H:29778;*
```

BARESIP Git: https://github.com/alfredh/baresip

BARESIP Wiki: https://github.com/alfredh/baresip/wiki

XRTP Specs: https://www.avm.de/de/Extern/files/x-rtp/xrtpv32.pdf

SIP Testing with quality-aware Agents

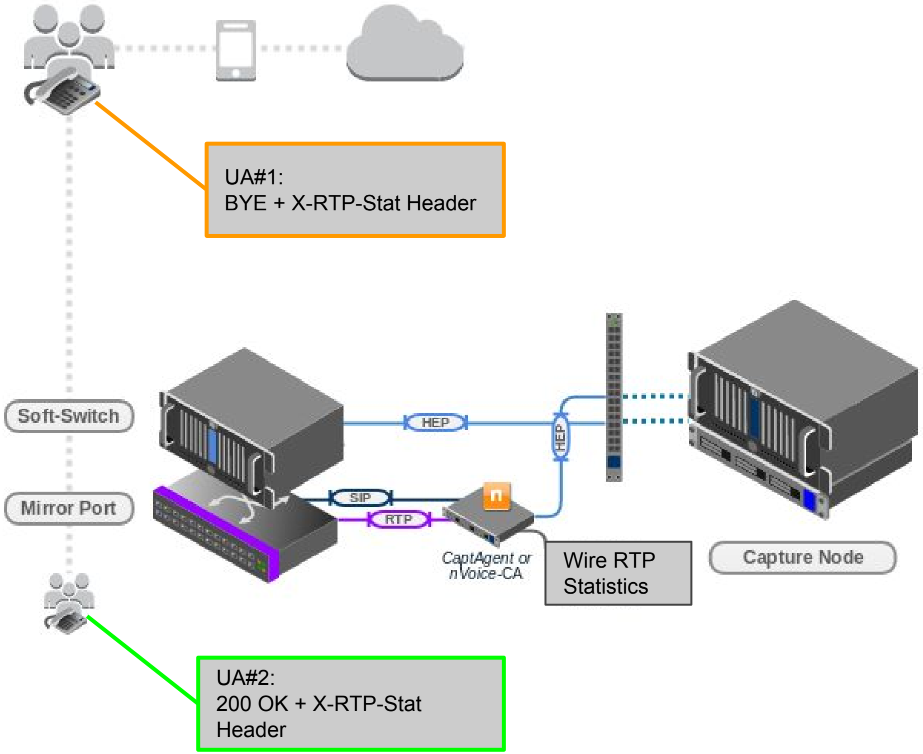
BARESIP User-Agent w/ X-RTP-Stats (continued)

Baresip agents can be deployed in tandem to validate call quality across specific SIP Paths.

In the following illustration:

- UA#1 Originates a session and steams pre-recorded audio to UA#2
- UA#2 acting as an Echo-Test streaming all packets back to the UA#1 (auto-answer)
- Both Agents will publish Stream quality statistics on session termination as X-RTP-Stats

A Capture Server monitoring SIP Signaling (such as HOMER) will be able to extract and process the quality reports from SIP Headers and provide this additional insight for troubleshooting issues in investigations or for alarming on automated tests.



SIPCAPTURE

SIP Testing with Scripted Agents

SIPP Scenarios for Service Validation

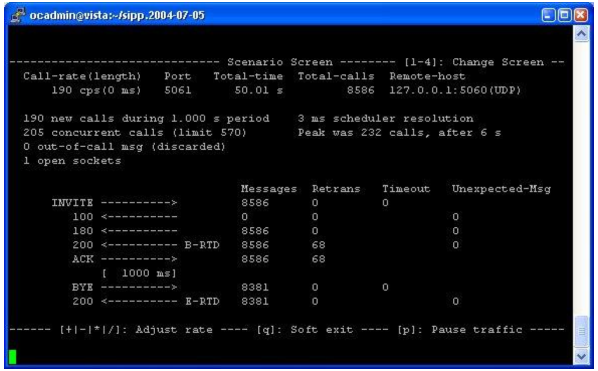
SIPp is a free Open-Source test tool and traffic generator for the SIP protocol, able to read custom XML scenario files describing from very simple to complex call flows simulating both User-Agent Servers and Clients supporting optional media traffic through *RTP echo* and *RTP / PCAP replay*. While optimized for stress and performance testing, **SIPp** can be used to run one single call and exit, providing a passed/failed verdict *(Exit code 0: Test Successful, Exit code 1: Test with Failures)* and export its details and results to CSV files making it perfectly suitable for ad-hoc testing and able to be paired with other platforms/scripts.

SIPp scenarios are easy and fun to write and customize with many community collections ready to be used and extended for just about any purpose - Our favourite is kindly provided by Saghul on Github:

https://github.com/saghul/sipp-scenarios

Several old-school tools are available to convert PCAP traces to SIPp Scenarios:

- http://sourceforge.net/projects/pcap2sipp/
- http://frox25.no-ip.org/~mtve/wiki/Pcap2Sipp.html
- http://svn.digium.com/svn/sniff2sipp/trunk/sniff2sipp



SIPp also runs great on the *Raspberry-PI* and makes a fantastic pocket tool. A good custom Pi-Tailored installer is maintained by Paul Miller on bitbucket:

```
# wget “http://bitbucket.org/idkpmiller/installation-scripts/raw/master/install_sipp.sh”
# chmod +x install_sipp.sh
# ./install_sipp.sh
```

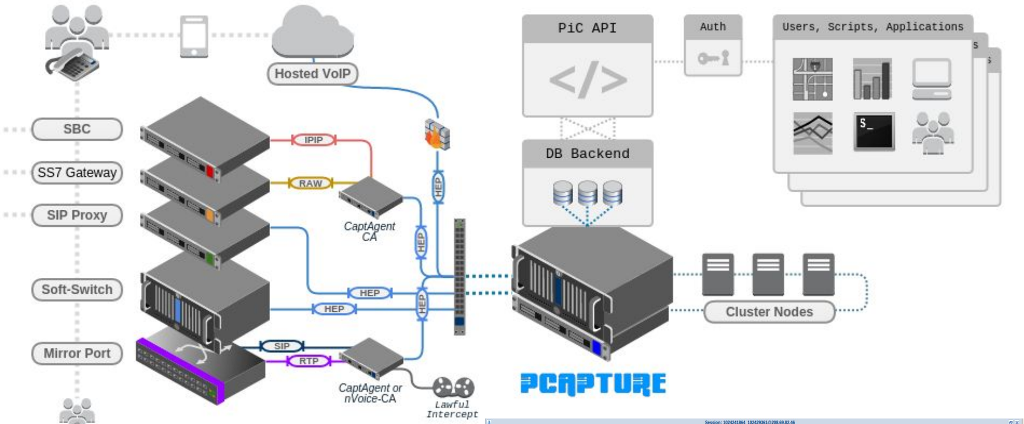
Running too BIG for Homer and MySQL?

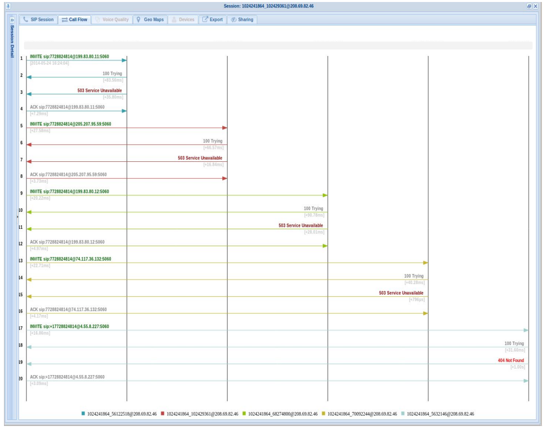
Meet **PCAPTURE**: The Extensible Capture Server and API

PCAPTURE is the commercial big-brother of *HOMER* and *SIPCAPTURE*, designed and crafted to provide a virtually infinite voice monitoring solution, leveraging the vast experience gathered assisting and developing solutions for some of the largest and busiest ITSPs, Telecommunication Networks and Vendors of Voice Services and Equipment in the industry.

PCAPTURE provides many additional features:

- Real-Time Tracking and Monitoring of Sessions
- RTP/RTCP/PUBLISH QoS Reports, MOS/RFactor
- CDRs & Log Collectors with integrated parsing
- Automatic Correlation of Sessions legs, QoS, Logs
- Scalable, Multiple Distributed-Database layers
- Rich Multi-User User-Interface (HTML5/ExtJS)
- 1-Click Complete Session Details, Real-Time Usage
- Fully customizable Dashboard and Widgets
- Cross-Platform Capture Agents & Analyzers
- 100% REST API based & Integration Ready
- One Click-Troubleshooting for Tech and Non-Tech





Find out more: http://www.pcapture.com

BONUS

SIPCAPTURE

Install & Run a **HOMER** Capture Server & Capture Agent in a snap!

All *SIPCAPTURE* Projects are now available as packages supporting CentOS 6/7, Debian 7/8 and Ubuntu Server 14+

Pick an install script for your platform - it will detect your platform and architecture, install the gpg key that we use for repo signing and setup the repo accordingly. Once set, proceed to install the Homer bundle meta-packet for your OS distribution.

DEB:

Debian 7/8, Ubuntu Server 14.04

```
curl -s https://packagecloud.io/install/repositories/qxip/homer/script.deb.sh | sudo bash
```

RPM:

CentOS 6/7, RHEL 6/7

```
curl -s https://packagecloud.io/install/repositories/qxip/homer/script.rpm.sh | sudo bash
```

Any Questions?

Q&A

SIP Troubleshooting

SIPCAPTURE

"That's all Folks!"

Time's UP! Want to go further?

Contact us to learn more about our advanced Capture and Troubleshooting Workshops

<training@qxip.net>