



**Performance Benchmark Test for
OpenSER with RTPproxy
July 22, 2008**

Revision History

Revision	Date of Issue	Changes
0.1	March 10, 2008	Initial Draft.
0.2	March 10, 2008	Minor edits.
0.3	March 31, 2008	Add test raw data.
0.4	May 16, 2008	Add charts.
0.5	July 21, 2008	Add test cases.
0.6	July 21, 2008	Minor edits and formatting.
0.7	July 22, 2008	Add details about data collection.
1.0	September 19, 2008	Final edits for publishing

Contents

Revision History	2
Contents	3
1 Introduction.....	5
2 Summary of Results.....	5
2.1 Call Completion.....	5
2.2 CPU Utilization.....	5
2.3 Network Traffic	6
2.4 Shared Memory Utilization.....	7
2.5 Lost RTP Packets.....	7
2.6 Round Trip Delay	8
2.7 Jitter Buffer	8
3 Test Bed Diagram	9
3.1 Call Scenarios	9
3.1.1 Performance Test Call Scenario.....	9
3.1.2 Quality Test Call Scenario	10
3.2 Summary of Test Bed Devices	10
4 Test Bed Configuration.....	11
4.1 OSP Server.....	11
4.2 Host of SIPp Clients and Servers.....	11
4.3 Host of OpenSER with RTPproxy	12
4.4 Routing Info on OSP Server	12
4.5 OpenSER.....	13
4.5.1 Openser.cfg	13
4.5.2 OpenSER Ver1.3 compile time options.....	22
4.6 SIPp.....	22
4.6.1 SIPp Client XML Scenario	22
4.6.2 Media Source XML Scenario	24
4.6.3 SIPp Server / Media Destination XML scenario	26
5 SIPp Client Parameters	28
5.1 Transport Mode.....	28
5.2 Call Limit	28
5.3 Timer Resolution	28
5.4 Frequency.....	29
5.5 Call Numbers	29
5.6 Screen Flush Frequency	29
6 Scripts for Running the Test	29
6.1 Data Collection Scripts	29
6.1.1 Sar Write Script.....	29
6.1.2 Sar Read Script	29
6.2 RTPproxy Script	29
6.3 SIPp Scripts.....	29
6.3.1 SIPp Client.....	30
6.3.2 Media Source	30
6.3.3 SIPp Server / Media Destination.....	30

7	Test Procedure	30
7.1	Start Test	30
7.2	After Test	30
8	Test Results.....	31
8.1	Actual Results	36

1 Introduction

This document describes a benchmark test and performance results for OpenSER with RTPproxy. The purpose of the stress test is to understand the performance boundary of OpenSER with RTPproxy in a production environment. To simulate a production environment, the OpenSER queries an external OSP peering server for routing information on each call and then reports call detail records to the external OSP server. Five destinations are returned to the OpenSER for every call in random order. Four of the five destinations simulate call failure scenarios so the OpenSER must retry the call an average of two times before the call is completed. These tests were performed on a Dell Precision 490 server with two Intel Xeon 5140 dual core, 2.33 GHz, CPUs and 4 GB RAM. Only one CPU core is used for this test.

Section 8.1 provides the raw data collected from the test.

2 Summary of Results

For production operations (with call retries and CDR reporting), TransNexus recommends the following guideline for sizing server hardware for OpenSER Ver1.3 and RTPproxy Ver1.0:

For a server hosting both OpenSER and RTPproxy, each One GHz of CPU processing capacity can manage a maximum of 325 simultaneous calls.

For example, a server with two, dual core, 3.0 GHz CPUs would effectively have twelve GHz of CPU processing capacity (2 CPUs * 2 cores * 3 GHz per CPU). This server, hosting OpenSER Ver1.3 and RTPproxy Ver1.0, would be able to manage 2580 simultaneous calls at approximately 95% CPU utilization.

Notes:

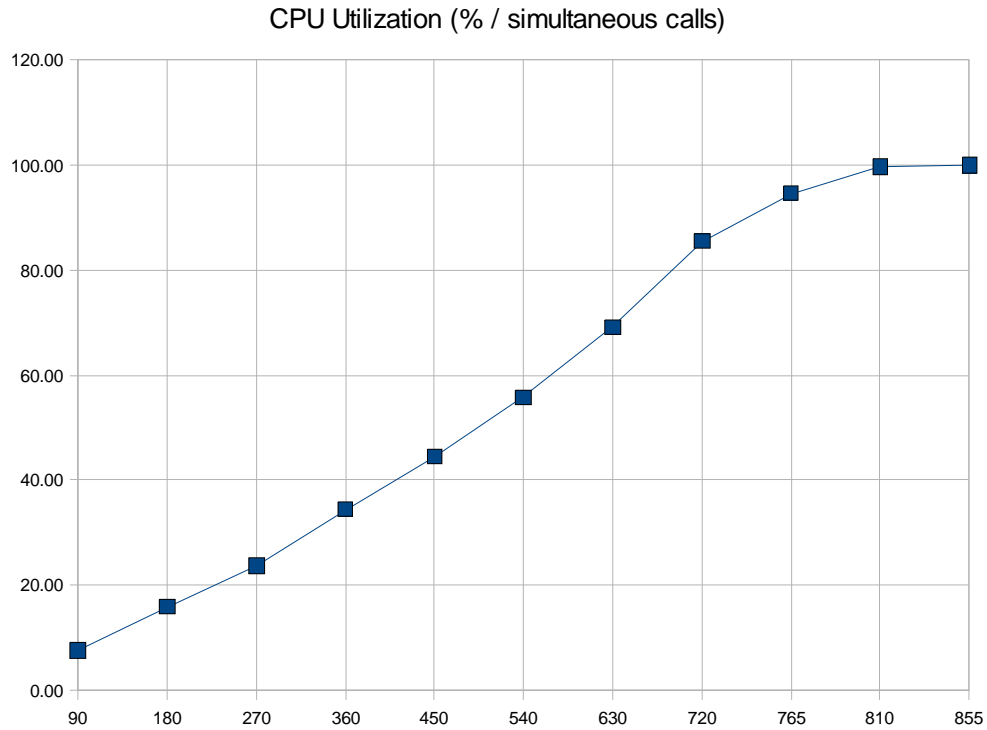
1. This estimation does not include network IO and other factors.
2. RTPproxy can only utilize a single CPU core. Multiple instances of RTPproxy must be run to utilize multiple CPU cores.

2.1 Call Completion

OpenSER Ver1.3 with RTPproxy Ver1.0 completed all test calls successfully in all test cases.

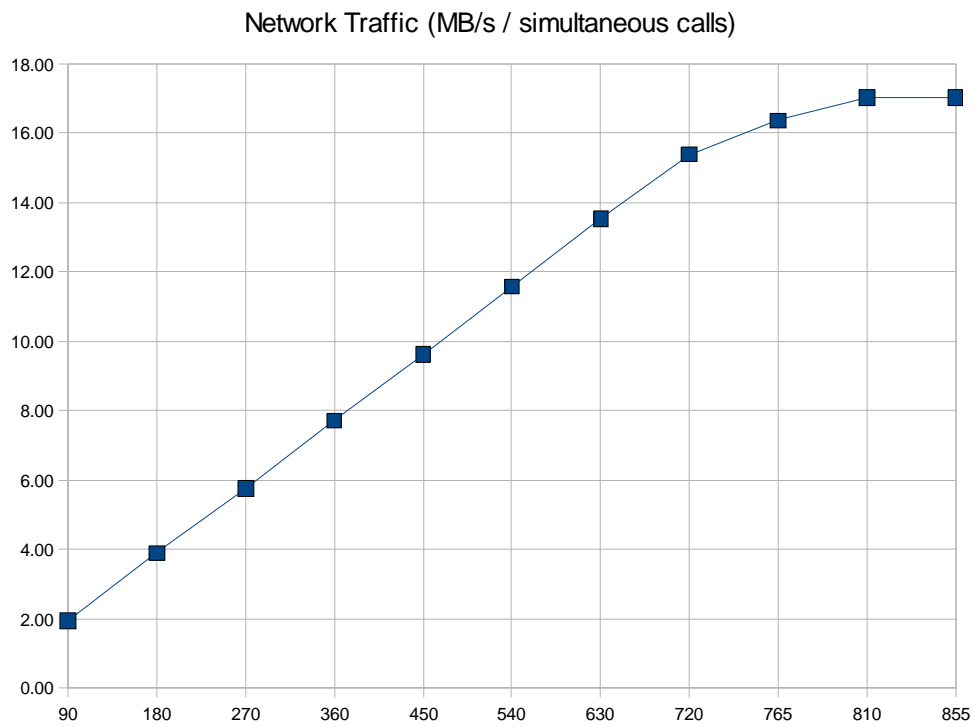
2.2 CPU Utilization

The following chart plots CPU utilization (y-axis) as a function of simultaneous calls (x-axis).



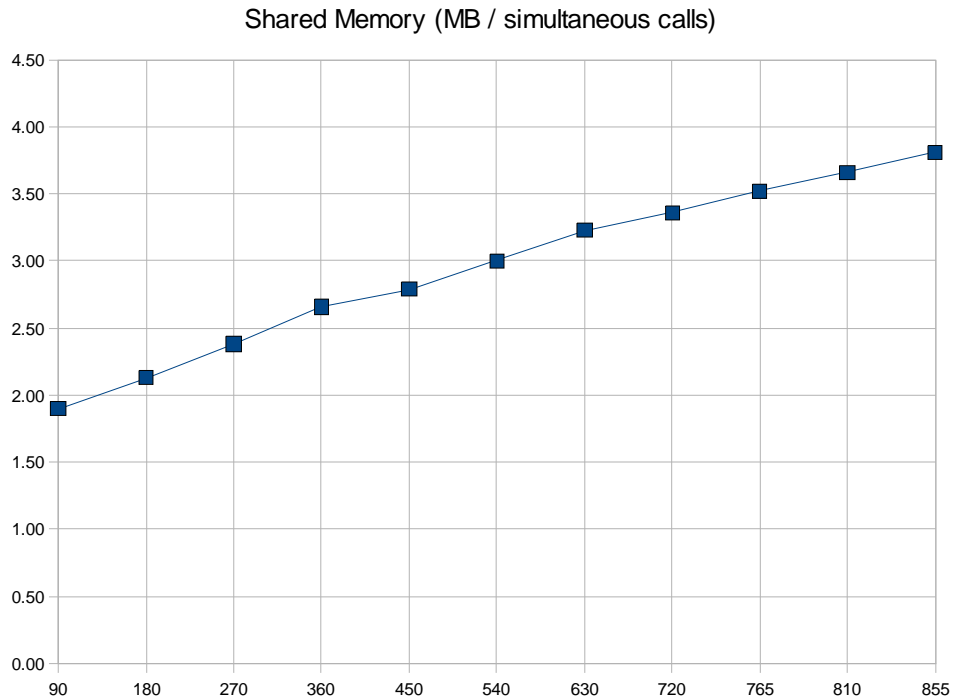
2.3 Network Traffic

The following chart plots network usage as a function of simultaneous calls. The data was for one direction network traffic.



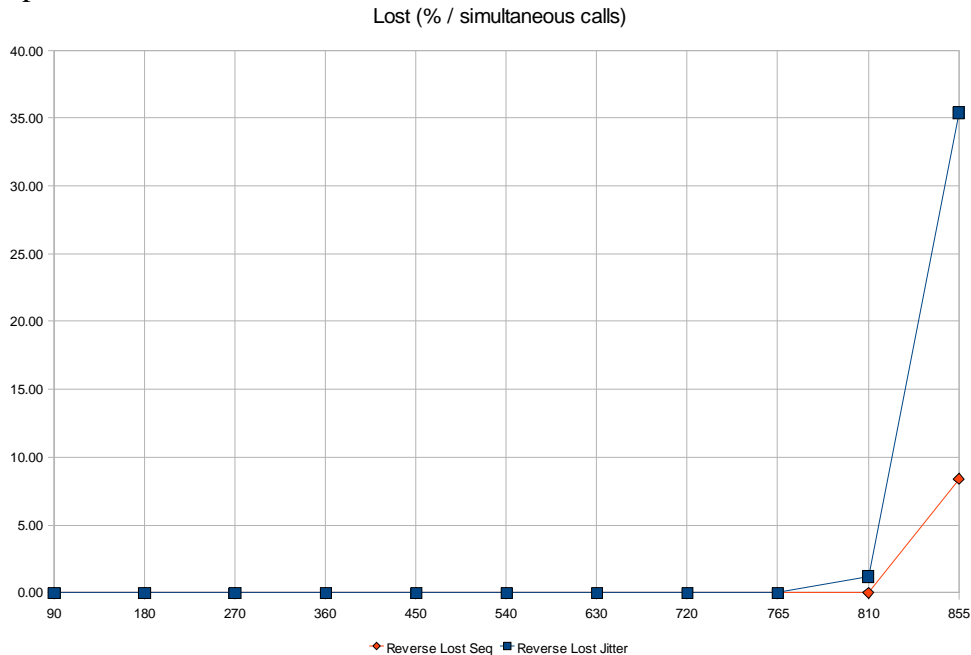
2.4 Shared Memory Utilization

The following chart plots OpenSER shared memory utilization (y-axis) as a function of simultaneous calls (x-axis).



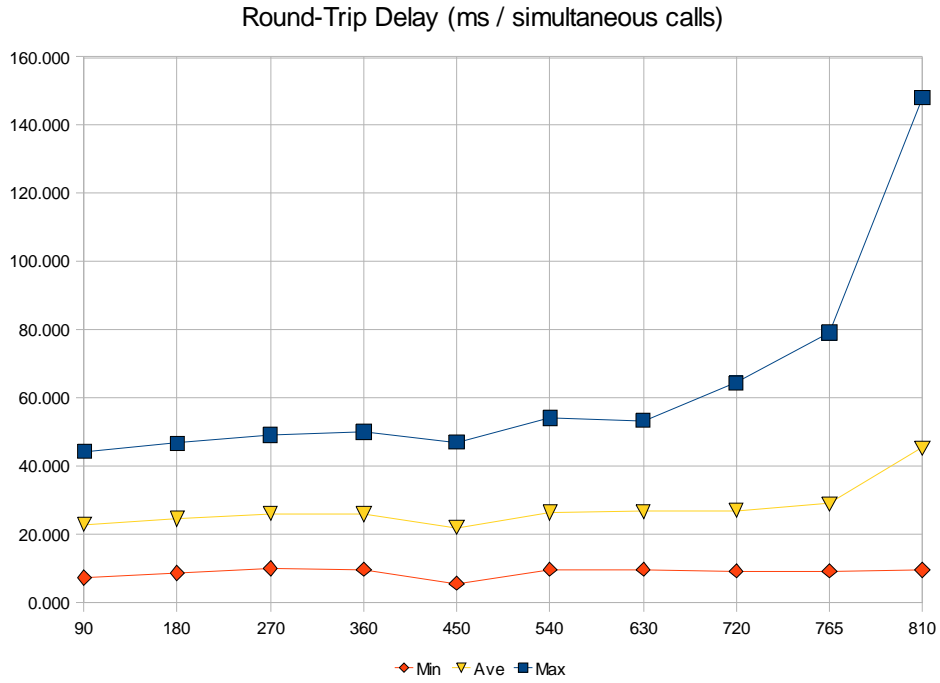
2.5 Lost RTP Packets

The following chart plots reverse packets lost as a function of simultaneous calls. **Note:** The lost RTP packets may include packets dropped by jitter at 50 ms jitter buffer, and out of sequence packets.



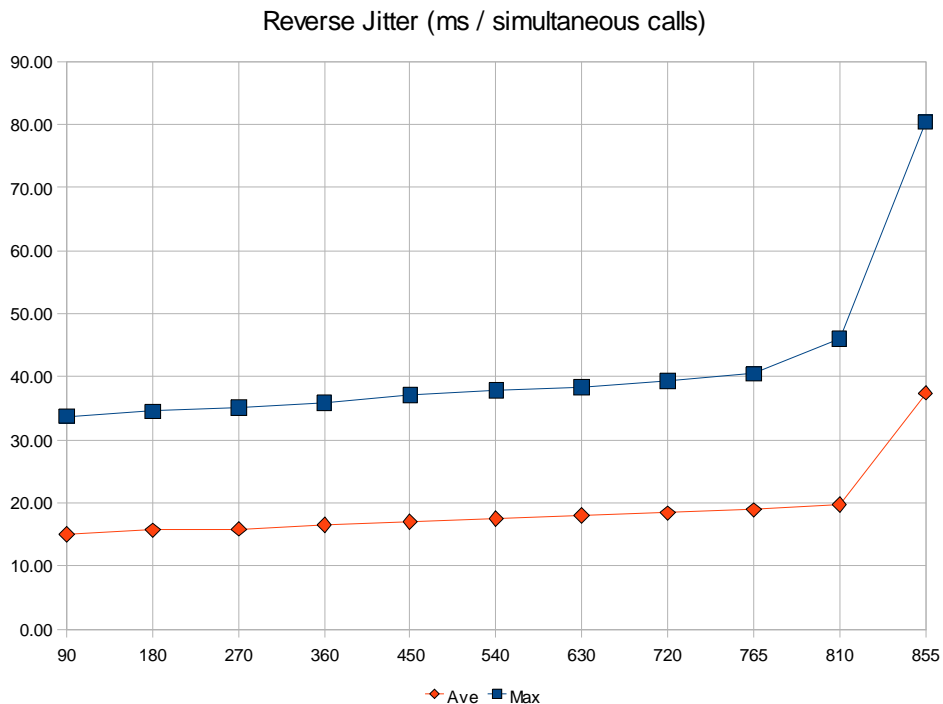
2.6 Round Trip Delay

The following chart plots round trip delay in milliseconds (y-axis) as a function of simultaneous calls (x-axis). **Note:** The round-trip delays at 855 simultaneous calls are max 14s, average 9s. These data points are not displayed in the following chart.

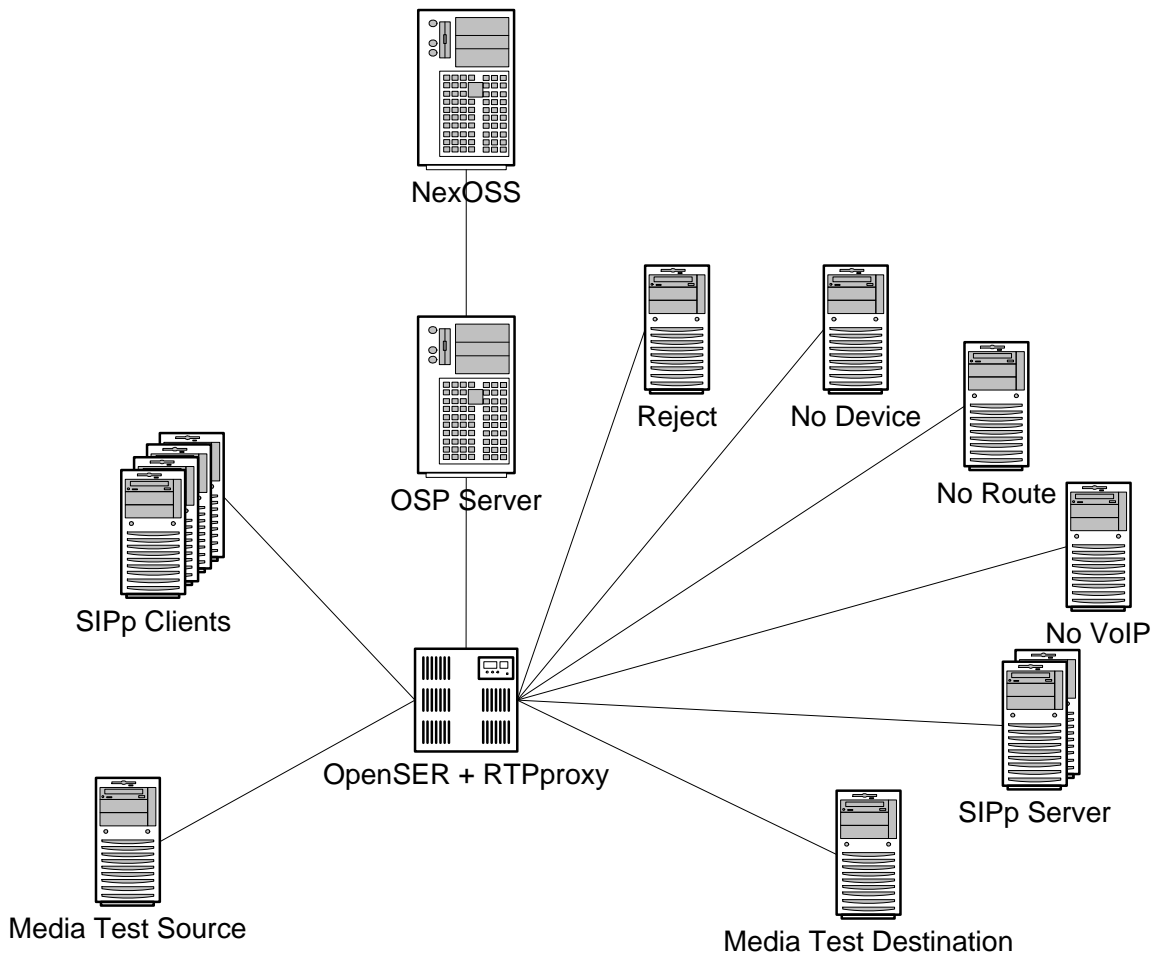


2.7 Jitter Buffer

The following chart plots jitter buffer (y-axis) as a function of simultaneous calls (x-axis). The jitter buffer was calculated according to RFC 3550.



3 Test Bed Diagram



OpenSER + RTPproxy Performance Test Bed Diagram

3.1 Call Scenarios

The following two test call scenarios should be performed at the same time. The performance test call scenario is used to measure the performance of OpenSER with RTPProxy. It also provides the background traffic for the call quality test. The quality test call scenario is used to measure the call quality during the performance test.

3.1.1 Performance Test Call Scenario

1. A SIPp client sends a SIP INVITE message to the OpenSER.
2. The OpenSER sends an OSP AuthorizationRequest message to the OSP peering server to query routing information.
3. The OSP server returns five destinations in random order (Reject, No Device, No Route, No VoIP, and one of the SIPp Servers). Four destinations test specific failover/retry scenarios. Only the returned SIPp server destination can complete the call and the other four destinations are configured to fail. Therefore, around

- 20% of the calls are completed on the first try.
 - 20% of the calls fail on the first attempt and complete on the second attempt.
 - 20% of the calls fail on the 1st two attempts and complete on the 3rd attempt.
 - 20% of the calls fail on the first three attempts and complete on the 4th attempt.
 - 20% of the calls fail on the first four attempts and complete on the 5th attempt.
4. The OpenSER requests the media path from the RTPproxy working as a media server in the call scenario.
 5. The SIPp client sends g711 ulaw voice stream for approximately three minutes (moh_ulaw.pcap) to the RTPproxy.
 6. The RTPproxy forwards the RTP messages to the SIPp server destination.
 7. The SIPp server destination echoes the RTP messages back to the RTPproxy.
 8. The RTPproxy forwards the voice stream to the SIPp client.
 9. The SIPp client sends a BYE to the OpenSER.
 10. When the call is finished, the OpenSER sends OSP UsageIndication messages (Call Detail Records) to the OSP server.

3.1.2 Quality Test Call Scenario

1. The media test source sends a SIP INVITE message to the OpenSER.
2. The OpenSER sends an OSP AuthorizationRequest message to the OSP server.
3. The OSP server returns the media test destination as the only destination back to complete the test call.
4. The OpenSER requests the media path from the RTPproxy working as a media server in the call scenario.
5. The media test source sends g711 ulaw voice stream for approximately three minutes (stress_ulaw.cap) to the RTPproxy.
6. The RTPproxy forwards the RTP messages to the media test destination.
7. The media test destination echoes the RTP messages back to the RTPproxy.
8. The RTPproxy forwards the voice stream to the media test source.
9. The media test source sends a BYE to the OpenSER.
10. When the call is finished, the OpenSER sends OSP UsageIndication messages to the OSP server.
11. During the test call, all SIP and RTP messages from/to the media test source and destination should be captured on both the media test source and destination boxes.

3.2 Summary of Test Bed Devices

- **NexOSS:** 172.16.4.32 NexOSS Ver3.4.1. It will run offline for the test.
- **OSP Servers:** 172.16.4.75 NexSRS Ver3.2.1.
- **OpenSER:** Ver1.3 is used in this test. The OpenSER has been compiled with OSP Toolkit to support the OSP protocol.
- **RTPproxy:** Ver1.0 is used in this test.
- **Server hosting OpenSER and RTPproxy:** Dell Precision 490 server with two Intel Xeon 5140 dual core, 2.33 GHz, CPUs and 4 GB RAM. Only one core is used in this test. The operating system is Fedora Core 7. The IP address is 172.16.4.32. The NIC used must be a Gigabyte NIC.
- **Reject:** 172.16.4.56. An Asterisk is running on 172.16.4.56. It will reject the called numbers we use in this test. The TCCode for this destination should be 404.

- **No Device:** An IP address in the 172.16.4.x network segment but no device uses this address, such as 172.16.4.100. This is a connection timeout destination. The TCCode for this destination should be 408.
- **No Route:** An IP address out of the 172.16.4.x network segment, such as 1.1.1.1. This is a connection timeout destination. The TCCode for this destination should be 408.
- **No VoIP:** A PC in the 172.16.4.x network segment without running SIP proxy or gateway, such as 172.16.4.1. This is a response timeout destination. The TCCode for this destination should be 408.
- **SIPp Clients:** 172.16.4.34, 172.16.4.35, 172.16.4.22 and 172.16.4.36. The port will be automatically selected by the SIPp application. A pcap file, moh_ulaw.pcap, is used to generate the 3 minute call duration. The SIPp client script includes the instruction to use this voice file. The NICs on the SIPp clients must be Gigabyte NICs.
- **SIPp Server:** 172.16.4.60 and 172.16.4.25. The ports must be 5060. OpenSER does not support other port in the recent release. The NICs on the SIPp servers must be Gigabyte NICs.
- **Media Test Source:** 172.16.4.59. This box runs the same SIPp application as the SIPp clients run. The pcap file is stress_ulaw.cap for the call quality test.
- **Media Test Destination:** 172.16.4.58. This box runs the same SIPp application as the SIPp servers run.

Note:

1. In order to simulate a production environment:
 - The OpenSER and RTPproxy should be hosted on the same box.
 - The OSP server and OpenSER should be hosted on the different boxes.
 - The SIPp devices and OpenSER should be hosted on the different boxes.
 - The OSP server should return 5 destinations for every OSP AuthReq.
 - Multiple SIPp clients/servers are used.
2. Since the limitation of OpenSER and OSP server applications, the port of the destinations must be 5060, the default SIP port.

4 Test Bed Configuration

4.1 OSP Server

In order to use multiple OSP Server instances on one host, the ports of the OSP Server instances must be configured.

There are 3 places where the configurable ports must agree.

1. PORT_BASE=N000 (in start_osp_server.sh)
 2. portbase=N000 (in xitami/defaults.cfg)
 3. port = N443 (in xitami/sslhtt.cfs)
- N should be 1, 2, 3, etc.

Note: Only one OSP server instance is used in this test.

4.2 Host of SIPp Clients and Servers

There is a configurable soft limit should be changed. The stack size should be 3072. Use *ulimit -s 3072* by root to set this parameter.

4.3 Host of OpenSER with RTPproxy

There is a configurable soft limit should be changed. The max open file should be at least 20480. Use *ulimit -n 20480* by root to set this parameter.

Change syn syslog to asyn will reduce CPU usage for I/O wait.

4.4 Routing Info on OSP Server

1. Two Customers:
 - STRESS
 - QUALITY
2. Fifteen Devices:
 - Devices of STRESS customer
 - Stress_Dev_SRS 172.16.4.75
 - Stress_Dev_Proxy 172.16.4.32
 - Stress_Dev_Reject 172.16.4.56
 - Stress_Dev_Nodevice 172.16.4.100
 - Stress_Dev_Noroute 1.1.1.1
 - Stress_Dev_Novoip 172.15.4.1
 - Stress_Dev_Src1 172.16.4.22
 - Stress_Dev_Src2 172.16.4.34
 - Stress_Dev_Src3 172.16.4.35
 - Stress_Dev_Src4 172.16.4.37
 - Stress_Dev_Src5 172.16.4.38
 - Stress_Dev_Dst1 172.16.4.60
 - Stress_Dev_Dst2 172.16.4.25
 - Devices of QUALITY customer
 - Quality_Dev_Src 172.16.4.59
 - Quality_Dev_Dst 172.16.4.58

Device Properties: All devices, except the source devices (Stress_Dev_Src1/2/3/4/5 and Quality_Dev_Src), should be configured with the following properties: OSP version 2.1.1, may terminate, is online, and has enrolled. The source devices should have the following properties: OSP version 2.1.1, may NOT terminate, is online, and has enrolled.

3. Six Destinations:
 - Stress_Dst_SIPp: with increment 1. 2 devices, Stress_Dev_Dst1 with weight 2 and Stress_Dev_Dst2 with weight 1.
 - Stress_Dst_Reject: with increment 1, device Stress_Dev_Reject.
 - Stress_Dst_Nodevice: with increment 1, device Stress_Dev_Nodevice.
 - Stress_Dst_Noroute: with increment 1, device Stress_Dev_Noroute.
 - Stress_Dst_Novoip: with increment 1, device Stress_Dev_Novoip.
 - Quality_Dst_SIPp: with increment 1, device Quality_Dev_Dst.
4. Two Route Plans:
 - Stress_Route: for all numbers return five destinations (Reject, No Device, No Route, No VoIP, and one of the SIPp Servers) in random order. This route plan is used for the performance test.

- Quality_Route: for all numbers return one destination, Quality_Dst_SIPp. This route plan is used for quality test.
5. Two Products:
 - STRESS: for all numbers, use route plan Stress_Route.
 - QUALITY: for all numbers, use route plan Quality_Route.
 6. Dial plan 1 is used for both customer STRESS and QUALITY for all numbers.

4.5 OpenSER

OpenSER should be compiled using default compile time options. The following parameters should be configured:

- OSP server IP addresses.
- OSP server weights. According to our study, for the stress test, the weights should be small to generate the best load balancing results. For example, 90/10 is better than 900/100. Large weights are better for backup scenarios.
- OpenSER local IP address, 172.16.4.32
- OSP key files
- *rtpproxy_sock* of nathelper module should be set to *unix:/var/run/rtpproxy.sock*.
- Debug level. It is set to 0 to reduce log output.
- Children. For the stress test, we use 64.
- TCP/TLS should be disabled by add *disable_tcp=yes* and *listen=udp:172.16.4.32:5060*. It reduces the total number of processes.
- *disable_dns_blacklist* should be set to yes to disable the DNS blacklist feature.
- *fr_timer* of tm module should be set to 2 to generate a 2 sec timeout for no connection destinations.

4.5.1 Openser.cfg

```
debug=0          # debug level (cmd line: -ddddddddd)
fork=yes
log_stderr=no   # (cmd line: -E)
sip_warning=no

/* Uncomment these lines to enter debugging mode
# fork=no
# log_stderr=yes
*/

check_via=no    # (cmd. line: -v)
dns=no         # (cmd. line: -r)
rev_dns=no     # (cmd. line: -R)
disable_dns_blacklist=yes
disable_tcp=yes
listen=udp:172.16.4.32:5060
port=5060
children=64

# ----- module loading -----
mpath="/usr/local/lib/openser/modules"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "maxfwd.so"
loadmodule "rr.so"
loadmodule "textops.so"
```

```

loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "auth.so"
loadmodule "mi_fifo.so"
loadmodule "options.so"
loadmodule "xlog.so"
loadmodule "avpops.so"
loadmodule "nathelper.so"
# Load OSP module
loadmodule "osp.so"

# ----- setting module-specific parameters -----

#
# PEERING PARAMETERS:
# =====
# This section contains OSP parameters that users may need to configure for multi-
# lateral peering. (spl_uri must be configured.) Additional detail on OSP Module
# parameters and functions is provided in the "OSP Module for Secure, Multi-Lateral
# Peering" document located at:http://developer.berlios.de/docman/?group\_id=3799
#
# Configure Peering Servers:
# =====
# OpenSER can be configured to query two peering servers for routing information
# and peering authorization tokens using the spl_uri and sp2_uri parameters. A
# configuration for spl_uri is required, configuring sp2_uri is optional. The
# peering server address should be configured as a standard URL beginning with
# either http:// or https:// followed by the domain name of the OSP server or the
# IP address enclosed in brackets. The domain name or IP address should be followed
# by the peering server TCP port number and uniform resource identifier. Below are
# example configurations.
#
# modparam("osp", "spl_uri", "http://osptestserver.transnexus.com:1080/osp")
# modparam("osp", "sp2_uri", "https://[1.2.3.4]:1443/osp")
modparam("osp", "spl_uri", "http://[172.16.4.75]:1080/osp")

#
# OpenSER IP Address
# =====
# device_ip is a recommended parameter that explicitly defines the IP address of
# OpenSER in a peering request message (as SourceAlternate type=transport). The IP
# address must be in brackets as shown in the example below.
#
# modparam("osp", "device_ip", "[1.1.1.1]")
modparam("osp", "device_ip", "[172.16.4.32]")

#
# Peering Token Validation
# =====
# When OpenSER receives a SIP INVITE with a peering token, the OSP Module will
# validate the token to determine whether or not the call has been authorized by a
# peering server. Peering tokens may, or may not, be digitally signed. This
# parameter defines if OpenSER will validate signed or unsigned tokens or both. The
# values for "token format" are defined below. The default value is 2.
#
# 0 - Validate only signed tokens. Calls with valid signed tokens are allowed.
# 1 - Validate only unsigned tokens. Calls with valid unsigned tokens are allowed.
# 2 - Validate both signed and unsigned tokens are allowed. Calls with valid
#     tokens are allowed.
#
modparam("osp", "token_format", 2)

#
# Crypto files from Peering Server Enrollment

```

```

# =====
# These parameters identify crypto files used for validating peering authorization
# tokens and establishing a secure channel between OpenSER and a peering server
# using SSL. The files are generated using the 'Enroll' utility from the OSP
# toolkit. By default, the proxy will look for pkey.pem, localcert.pem, and
# cacart_0.pem in the default configuration directory. The default config
# directory is set at compile time using CFG_DIR and defaults to
# /usr/local/etc/openser/. The files may be copied to the expected file location
# or the parameters below may be changed.
#
# If the default CFG_DIR value was used at compile time, the files will be loaded
# from:
modparam("osp", "private_key", "/usr/local/etc/openser/pkey.pem")
modparam("osp", "local_certificate", "/usr/local/etc/openser/localcert.pem")
modparam("osp", "ca_certificates", "/usr/local/etc/openser/cacert_0.pem")

#
# Use Remote-Party-ID for calling number
# =====
# This parameter is used to tell OSP module if the calling number should be
# obtained from RPID header. The default value is 1.
#
# 0 - OSP module will use the calling number in From header.
# 1 - OSP module will use the calling number in RPID header if a RPID header exists.
#
modparam("osp", "use_rpid_for_calling_number", 1)

#
# URI Format for Redirection Messages
# =====
# This parameter is used to tell OSP module which URI format should be used for
# redirection messages. The default value is 0.
#
# 0 - "xxxxxxxxxxx@xxx.xxx.xxx.xxx"
# 1 - "<xxxxxxxxxxx@xxx.xxx.xxx.xxx>". This is for Cisco 2600 IP-IP gateway.
#
modparam("osp", "redirection_uri_format", 0)

#
# Source Network ID AVP
# =====
# This parameter is used to tell OSP module which AVP is used to store source
# network ID. The default value is "$avp(s:_osp_source_networkid)".
#
# modparam("osp", "source_networkid_avp", "$avp(s:_osp_source_networkid)")
modparam("osp", "source_networkid_avp", "$avp(s:snid)")

# -- mi_fifo params --
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")

# -- usrloc params --
modparam("usrloc", "db_mode", 0)

# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)

# enable append_fromtag, request's from-tag is appended to record-route;
# that's useful for understanding whether subsequent requests (such as BYE) come
# from caller (route's from-tag==BYE's from-tag) or callee (route's from-tag==BYE's
# to-tag)
modparam("rr", "append_fromtag", 1)

# Timer which hits if no final reply for a request or ACK for a

```

```

# negative INVITE reply arrives (in seconds).  For example - UA server is off-line.
# In other words, if the proxy does not receive a response to an Invite before this
# timer expires, the proxy will retry the call and send an Invite to the next VoIP
# destination in the routing list.
modparam("tm", "fr_timer", 2)

# Timer which hits if no final reply for an INVITE arrives after
# a provisional message was received (in seconds).
# For example - user is not picking up the phone
modparam("tm", "fr_inv_timer", 30)

modparam("nathelper", "rtpproxy_sock", "unix:/var/run/rtpproxy.sock")

# ----- request routing logic -----

# main routing logic
route{
    xlog("L_INFO", "----ROUTE: Route IN - M=$rm RURI=$ru F=$fu T=$tu IP=$si
ID=$ci\n");

    # initial sanity checks
    if (!mf_process_maxfwd_header("10")) {
        xlog("L_WARN", "----ROUTE: Too many hops, $rm from '$fu' to '$tu'\n");
        sl_send_reply("483", "Too Many Hops");
        return;
    };

    if (msg:len >= max_len) {
        xlog("L_WARN", "----ROUTE: Message too big, $rm from '$fu' to '$tu'\n");
        sl_send_reply("513", "Message Too Big");
        return;
    };

    # we record-route all messages -- to make sure that
    # subsequent messages will go through our proxy; that's
    # particularly good if upstream and downstream entities
    # use different transport protocol
    record_route();

    # loose-route processing
    if(loose_route()) {
        if (method=="INVITE") {
            log(2, "----ROUTE: Relay re-INVITE\n");
            # send it out now; use stateful forwarding as it works reliably even
            for UDP2TCP
            force_rtp_proxy("1");
            t_on_reply("1");
            if (!t_relay()) {
                sl_reply_error();
            }
            return;
        } else if (method=="ACK") {
            log(2, "----ROUTE: Relay ACK\n");
            # send it out now; use stateful forwarding as it works reliably even
            for UDP2TCP
            if (!t_relay()) {
                sl_reply_error();
            }
            return;
        }
    } else {
        if (method=="BYE") {
            xlog("L_WARN", "----ROUTE: Processing BYE without route header - F=$fu
T=$tu IP=$si ID=$ci\n");

```

```

        if (t_check_trans()) {
            log(2, "----ROUTE: Duplicated BYE\n");
        } else {
            # NOTE - don't t_relay before reporting usage
            if (is_direction("downstream")) {
                log(2, "----ROUTE: BYE from SOURCE\n");
                if (!reportospusage("0")) {
                    xlog("L_WARN", "----ROUTE: failed to report usage, from
'$fu' to '$tu'\n");
                }
            } else {
                log(2, "----ROUTE: BYE from DESTINATION\n");
                if (!reportospusage("1")) {
                    xlog("L_WARN", "----ROUTE: failed to report usage, from
'$fu' to '$tu'\n");
                }
            }
            sl_send_reply("400", "Bad Request - no route header in BYE
message");
        }
        return;
    }
}

if (method=="REGISTER") {
    log(2, "----ROUTE: Processing REGISTER\n");

    # Stop retransmission
    sl_send_reply("100", "Trying");

    if (uri==myself) {
        log(2, "----ROUTE: Registered\n");
        save("location");
    } else {
        log(2, "----ROUTE: Register from outside domain rejected\n");
        sl_send_reply("488", "Unknown Domain");
    }
} else if (method=="INVITE") {
    log(2, "----ROUTE: Processing INVITE\n");

    # Stop retransmission
    sl_send_reply("100", "Trying");

    if (t_check_trans()) {
        log(2, "----ROUTE: Duplicated INVITE\n");
        return;
    }

    # Authentication
    log(3, "OSP authorization validation logic\n");

    # This function looks for OSP peering token in the message. It will fail
    # if the token is not present
    if (checkospheader()) {
        log(3, "With OSP token, validate it\n");

        # The function validates OSP tokens. It will fail
        # if the token is not valid or has expired
        if (validateospheader()) {
            # Authorization is valid. The proxy can now use its own database
            # registered users for routing information.
            # The proxy could also issue another OSP peering authorization and
            # routing request by calling route(1) function.

```

```

        log(3, "OSP authorization valid\n");

        # Remove the OSP peering token from the received message
        # Otherwise it will be forwarded on to the next hop
        remove_hf("P-OSP-Auth-Token");
    } else {
        log(3, "OSP authorization invalid\n");
        sl_send_reply("401", "Unauthorized");
        return;
    };
} else {
    log(3, "Without OSP token, apply different authentication strategy\n");
    log(3, "Go ahead, everyone is welcomed\n");

    # # Implement authentication strategy here or simply add the
    # # statements below to block all invites without OSP peering tokens
    # sl_send_reply("401", "Unauthorized");
    # return;
}

log(2, "----ROUTE: Authentication passed\n");

# Routing
if (lookup("location")) {
    log(2, "----ROUTE: Registered user, forward the message\n");
    append_hf("P-hint: usrloc\r\n");
force_rtp_proxy();
t_on_reply("1");
    t_relay();
} else {
    log(2, "----ROUTE: Unregistered user, use OSP to get routing\n");
    route(2);
}
} else if (method=="ACK") {
    log(2, "----ROUTE: Processing ACK\n");
    log(2, "----ROUTE: Not to relay ACK\n");
} else if (method=="BYE") {
    log(2, "----ROUTE: Processing BYE\n");

    if (t_check_trans()) {
        log(2, "----ROUTE: Duplicated BYE\n");
        return;
    }

    # NOTE - don't t_relay before reporting usage
    if (is_direction("downstream")) {
        log(2, "----ROUTE: BYE from SOURCE\n");
        if (!reportospusage("0")) {
            xlog("L_WARN", "----ROUTE: failed to report usage, from '$fu' to
'$tu'\n");
        }
    } else {
        log(2, "----ROUTE: BYE from DESTINATION\n");
        if (!reportospusage("1")) {
            xlog("L_WARN", "----ROUTE: failed to report usage, from '$fu' to
'$tu'\n");
        }
    }
}

unforce_rtp_proxy();

    t_relay();
} else if (method=="CANCEL") {
    log(2, "----ROUTE: Processing CANCEL\n");

```

```

        if (t_check_trans()) {
unforce_rtp_proxy();
        t_relay();
        } else {
            xlog("L_WARN", "----ROUTE: CANCEL without matching transaction, from
'$fu' to '$tu'\n");
        }
    } else if ((method=="OPTIONS") && (uri==myself)) {
        log(2, "----ROUTE: Processing OPTIONS\n");
        options_reply();
    } else if (method=="PRACK") {
        log(2, "----ROUTE: Processing PRACK\n");
        t_relay();
    } else if (method=="INFO") {
        log(2, "----ROUTE: Processing INFO\n");
        t_relay();
    } else if (method=="UPDATE") {
        log(2, "----ROUTE: Processing UPDATE\n");
        t_relay();
    } else {
        xlog("L_WARN", "----ROUTE: Unsupported message, $rm from '$fu' to '$tu'\n");
        sl_send_reply("500", "Unsupported Message");
    }

    log(3, "----ROUTE: Route OUT\n");
}

```

```

# OSP Authorization and Routing

```

```

route[2] {
    log(3, "OSP authorization and routing logic\n");

    # Is request to a phone number?
    # A phone number consists of digits (0 through 9)
    # and can begin with +
    #
    # if (uri=~"sip:[+,0-9][0-9]*@") {
        # Requesting OSP peering routing and authorization
        # The request may fail if:
        #   o OSP peering servers are not available
        #   o Authentication failed
        #   o There is no route to destination or the route is blocked
        log(3, "Requesting OSP authorization and routing\n");

        # Get source network ID
        # $avp(s:snid) = $hdr(Call-Owner);

        requestosprouting();
        switch ($retcode) {
            case 1:
                log(3, "Response received\n");

                # Now we have 3 options.
                #   o route(3) - sends a redirect to all available routes
                #   o route(4) - fork off to all available routes
                #   o route(5) in conjunction with failure_route(1) - sequentially
                #   tries all routes

                # route(3);
                # route(4);
                route(5);

                break;
            case -403:

```

```

                xlog("L_WARN", "----ROUTE: Call to '$tU' from source device '$si'
is blocked on OSP Server.\n");
                sl_send_reply("403", "Forbidden - Call is blocked");
                break;
            case -404:
                xlog("L_WARN", "----ROUTE: No route on OSP server for call to '$tU'
from source device '$si'.\n");
                sl_send_reply("404", "Route Not Found");
                break;
            case -500:
                log(3, "----ROUTE: Internal Server Error\n");
                sl_send_reply("500", "Internal Server Error");
                break;
            default:
                log(3, "----ROUTE: OSP Authorization failed\n");
                sl_send_reply("503", "Service Not Available");
        }
#     } else {
#         log(3, "Wrong phone number\n");
#         sl_send_reply("401", "Not Phone Number");
#     }
}

route[3] {
    log(3, "Prepare all routes and redirect\n");

    if (prepareallosproutes()) {
        sl_send_reply("300", "Redirect");
    } else {
        log(3, "Failed to prepare all routes\n");
        sl_send_reply("500", "Internal Server Error");
    }
}

route[4] {
    log(3, "Prepare all routes and fork-off\n");

    if (prepareallosproutes()) {
force_rtp_proxy();
t_on_reply("1");
        t_relay();
    } else {
        log(3, "Failed to prepare all routes\n");
        sl_send_reply("500", "Internal Server Error");
    }
}

route[5] {
    log(3, "Try the 1st route\n");

    if (checkosproute()) {
force_rtp_proxy();
t_on_reply("1");

        t_on_branch("1");

        t_on_failure("1");

        t_relay();
    } else {
        log(3, "Could not use the 1st route\n");
    }
}

```

```

        sl_send_reply("500", "Internal Server Error");
    }
}

onreply_route[1] {
    if (status =~ "183|2[0-9][0-9]") {
        force_rtp_proxy();
    }
}

failure_route[1] {
    if (t_check_status("487")) {
        log(3, "Call canceled (status 487)\n");
        unforce_rtp_proxy();
        return;
    }

    if (t_check_status("486")) {
        log(3, "User busy (status 486)\n");
        unforce_rtp_proxy();
        return;
    }

    if (t_check_status("408")) {
        if (!t_local_replied("last")) {
            log(3, "User unavailable (status 408)\n");
            unforce_rtp_proxy();
            return;
        }
    }

    log(3, "Try the next route\n");

    if (checkosproute()) {
        t_on_reply("1");

        t_on_branch("1");

        append_branch();

        t_on_failure("1");

        t_relay();
    } else {
        xlog("L_WARN", "----ROUTE: All destinations attempted for call ID '$ci'.
Call cannot be completed.\n");
        unforce_rtp_proxy();
        t_reply("503", "Service Not Available - Call cannot be completed");
    }
}

branch_route[1] {
    log(3, "Prepare route specific OSP information\n");

    prepareosproute();

    # Only add/change Remote-Party-ID if calling number is translated
    if (checkcallingtranslation()) {
        log(3, "Calling number translated, add a new RPID header\n");

        # Remove the Remote-Party-ID from the received message

```

```

    # Otherwise it will be forwarded on to the next hop
    remove_hf("Remote-Party-ID");

    # Append a new Remote_Party
    append_rpid_hf();
}
}

```

4.5.2 OpenSER Ver1.3 compile time options

```

[ser@fedora7 sbin]$ ./openser -V
version: openser 1.3.0-notls (i386/linux)
flags: STATS: Off, USE_IPV6, USE_TCP, DISABLE_NAGLE, USE_MCAST, SHM_MEM, SHM_MMAP,
PKG_MALLOC, F_MALLOC, FAST_LOCK-ADAPTIVE_WAIT
ADAPTIVE_WAIT_LOOPS=1024, MAX_RECV_BUFFER_SIZE 262144, MAX_LISTEN 16, MAX_URI_SIZE
1024, BUF_SIZE 65535
poll method support: poll, epoll_lt, epoll_et, sigio_rt, select.
svnrevision: 2:3741M
@(#) $Id: main.c 3590 2008-01-28 17:46:56Z bogdan_iancu $
main.c compiled on 03:15:17 Feb 23 2008 with gcc 4.1.2

```

4.6 SIPp

In order to send a voice stream to test RTPproxy, SIPp must be compiled by *make pcapplay*. A voice file, moh_ulaw.pcap, is used in the performance test. It is in g711 ulaw format and approximately 180 seconds long. Another voice file, stress_ulaw.cap, is used in the call quality test. It is also in g711 ulaw format and approximately 180 seconds long.

4.6.1 SIPp Client XML Scenario

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- This program is free software; you can redistribute it and/or      -->
<!-- modify it under the terms of the GNU General Public License as    -->
<!-- published by the Free Software Foundation; either version 2 of the -->
<!-- License, or (at your option) any later version.                  -->
<!--                                                                    -->
<!-- This program is distributed in the hope that it will be useful,   -->
<!-- but WITHOUT ANY WARRANTY; without even the implied warranty of   -->
<!-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the    -->
<!-- GNU General Public License for more details.                     -->
<!--                                                                    -->
<!-- You should have received a copy of the GNU General Public License -->
<!-- along with this program; if not, write to the                    -->
<!-- Free Software Foundation, Inc.,                                    -->
<!-- 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA           -->
<!--                                                                    -->
<!--          Sipp 'uac' scenario with pcap (rtp) play                 -->
<!--                                                                    -->

<scenario name="UAC with media">
  <!-- In client mode (sipp placing calls), the Call-ID MUST be      -->
  <!-- generated by sipp. To do so, use [call_id] keyword.           -->
  <send retrans="500" start_rtd="1">
    <![CDATA[

      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
    ]>

```

```

Contact: sip:sipp@[local_ip]:[local_port]
Max-Forwards: 70
Subject: Performance Test
Content-Type: application/sdp
Content-Length: [len]

v=0
o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
s=-
c=IN IP[local_ip_type] [local_ip]
t=0 0
m=audio [auto_media_port] RTP/AVP 0 101
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11,16

]]>
</send>

<recv response="100" rtd="1" start_rtd="2">
</recv>

<recv response="100" rtd="2" start_rtd="3" optional="true">
</recv>

<recv response="180" rtd="3" optional="true">
</recv>

<!-- By adding rrs="true" (Record Route Sets), the route sets -->
<!-- are saved and used for following messages sent. Useful to test -->
<!-- against stateful SIP proxies/B2BUAs. -->
<recv response="200" rrs="true">
  <action>
    <ereg regexp="[1-9].*" search_in="hdr" header="Contact:" check_it="true"
assign_to="1" />
  </action>
</recv>

<!-- Packet lost can be simulated in any send/recv message by -->
<!-- by adding the 'lost = "10"'. Value can be [1-100] percent. -->
<send>
  <![CDATA[

    ACK sip:[$1] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
    [routes]
    From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
    To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 1 ACK
    Contact: sip:sipp@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<!-- Play a pre-recorded PCAP file (RTP stream) -->
<nop>
  <action>
    <exec play_pcap_audio="./moh_ulaw.pcap"/>
  </action>
</nop>

```

```

<!-- Pause 180 seconds, which is approximately the duration of the -->
<!-- PCAP file -->
<pause milliseconds="180000"/>

<!-- Play an out of band DTMF '1' -->
<nop>
  <action>
    <exec play_pcap_audio="../pcap/dtmf_2833_1.pcap"/>
  </action>
</nop>

<pause milliseconds="1000"/>

<!-- The 'crlf' option inserts a blank line in the statistics report. -->
<send retrans="500">
  <![CDATA[

    BYE sip:[$1] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
    [routes]
    From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
    To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 2 BYE
    Contact: sip:sipp@[local_ip]:[local_port]
    Max-Forwards: 70
    Subject: Performance Test
    Content-Length: 0

  ]]>
</send>

<recv response="200" crlf="true">
</recv>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="50, 100, 200, 500, 1000, 2000, 3000, 4000, 5000,
6000, 10000"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>

```

4.6.2 Media Source XML Scenario

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- This program is free software; you can redistribute it and/or -->
<!-- modify it under the terms of the GNU General Public License as -->
<!-- published by the Free Software Foundation; either version 2 of the -->
<!-- License, or (at your option) any later version. -->
<!-- -->
<!-- This program is distributed in the hope that it will be useful, -->
<!-- but WITHOUT ANY WARRANTY; without even the implied warranty of -->
<!-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the -->
<!-- GNU General Public License for more details. -->
<!-- -->
<!-- You should have received a copy of the GNU General Public License -->
<!-- along with this program; if not, write to the -->
<!-- Free Software Foundation, Inc., -->
<!-- 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA -->
<!-- -->

```

```

<!--          Sipp 'uac' scenario with pcap (rtp) play          -->
<!--          -->

<scenario name="UAC with media">
  <!-- In client mode (sipp placing calls), the Call-ID MUST be          -->
  <!-- generated by sipp. To do so, use [call_id] keyword.          -->
  <send retrans="500" start_rtd="1">
    <![CDATA[

      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>
      Call-ID: [call_id]
      CSeq: 1 INVITE
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[local_ip_type] [local_ip]
      t=0 0
      m=audio [auto_media_port] RTP/AVP 0 101
      a=rtpmap:0 PCMU/8000
      a=rtpmap:101 telephone-event/8000
      a=fmtp:101 0-11,16

    ]]>
  </send>

  <recv response="100" rtd="1" start_rtd="2">
  </recv>

  <recv response="100" rtd="2" start_rtd="3" optional="true">
  </recv>

  <recv response="180" rtd="3" optional="true">
  </recv>

  <!-- By adding rrs="true" (Record Route Sets), the route sets          -->
  <!-- are saved and used for following messages sent. Useful to test          -->
  <!-- against stateful SIP proxies/B2BUAs.          -->
  <recv response="200" rrs="true">
    <action>
      <ereg regexp="[1-9].*" search_in="hdr" header="Contact:" check_it="true"
assign_to="1" />
    </action>
  </recv>

  <!-- Packet lost can be simulated in any send/recv message by          -->
  <!-- by adding the 'lost = "10"'. Value can be [1-100] percent.          -->
  <send>
    <![CDATA[

      ACK sip:[$1] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
      [routes]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]

```

```

        CSeq: 1 ACK
        Contact: sip:sipp@[local_ip]:[local_port]
        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0

    ]]>
</send>

<!-- Play a pre-recorded PCAP file (RTP stream) -->
<nop>
    <action>
        <exec play_pcap_audio="./stress_ulaw.pcap"/>
    </action>
</nop>

<!-- Pause 180 seconds, which is approximately the duration of the -->
<!-- PCAP file -->
<pause milliseconds="180000"/>

<!-- The 'crlf' option inserts a blank line in the statistics report. -->
<send retrans="500">
    <![CDATA[

        BYE sip:[$1] SIP/2.0
        Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
        [routes]
        From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[pid]SIPpTag00[call_number]
        To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
        Call-ID: [call_id]
        CSeq: 2 BYE
        Contact: sip:sipp@[local_ip]:[local_port]
        Max-Forwards: 70
        Subject: Performance Test
        Content-Length: 0

    ]]>
</send>

<recv response="200" crlf="true">
</recv>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="50, 100, 200, 500, 1000, 2000, 3000, 4000, 5000,
6000, 10000"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>

```

4.6.3 SIPp Server / Media Destination XML scenario

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">

<!-- This program is free software; you can redistribute it and/or -->
<!-- modify it under the terms of the GNU General Public License as -->
<!-- published by the Free Software Foundation; either version 2 of the -->
<!-- License, or (at your option) any later version. -->
<!-- -->
<!-- This program is distributed in the hope that it will be useful, -->
<!-- but WITHOUT ANY WARRANTY; without even the implied warranty of -->
<!-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the -->
<!-- GNU General Public License for more details. -->

```

```

<!-- -->
<!-- You should have received a copy of the GNU General Public License -->
<!-- along with this program; if not, write to the -->
<!-- Free Software Foundation, Inc., -->
<!-- 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA -->
<!-- -->
<!-- Sipp default 'uas' scenario. -->
<!-- -->

<scenario name="Basic UAS responder">
  <!-- By adding rrs="true" (Record Route Sets), the route sets -->
  <!-- are saved and used for following messages sent. Useful to test -->
  <!-- against stateful SIP proxies/B2BUAs. -->
  <recv request="INVITE" crlf="true">
  </recv>

  <!-- The '[last_*]' keyword is replaced automatically by the -->
  <!-- specified header if it was present in the last message received -->
  <!-- (except if it was a retransmission). If the header was not -->
  <!-- present or if no message has been received, the '[last_*]' -->
  <!-- keyword is discarded, and all bytes until the end of the line -->
  <!-- are also discarded. -->
  <!-- -->
  <!-- If the specified header was present several times in the -->
  <!-- message, all occurrences are concatenated (CRLF seperated) -->
  <!-- to be used in place of the '[last_*]' keyword. -->

  <send>
    <![CDATA[

      SIP/2.0 180 Ringing
      [last_Via:]
      [last_Record-Route:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Length: 0

    ]]>
  </send>

  <send retrans="500">
    <![CDATA[

      SIP/2.0 200 OK
      [last_Via:]
      [last_Record-Route:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip:[local_ip]:[local_port];transport=[transport]>
      Content-Type: application/sdp
      Content-Length: [len]

      v=0
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
      s=-
      c=IN IP[media_ip_type] [media_ip]
      t=0 0
      m=audio [media_port] RTP/AVP 0
      a=rtpmap:0 PCMU/8000

    ]]>
  </send>

```

```

    ]]>
</send>

<recv request="ACK"
      optional="true"
      crlf="true">
</recv>

<recv request="BYE">
</recv>

<send>
  <![CDATA[

    SIP/2.0 200 OK
    [last_Via:]
    [last_From:]
    [last_To:]
    [last_Call-ID:]
    [last_CSeq:]
    Contact: <sip:[local_ip]:[local_port];transport=[transport]>
    Content-Length: 0

  ]]>
</send>

<!-- Keep the call open for a while in case the 200 is lost to be    -->
<!-- able to retransmit it if we receive the BYE again.          -->
<pause milliseconds="4000"/>

<!-- definition of the response time repartition table (unit is ms) -->
<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200"/>

<!-- definition of the call length repartition table (unit is ms) -->
<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000"/>

</scenario>

```

5 SIPp Client Parameters

5.1 Transport Mode

SIPp supports multiple transport modes. The transport mode can be set by command line, **-t mode**, from the client end. The default setting is UDP one socket. To simulate product environments, multiple sockets may be used.

5.2 Call Limit

This parameter is used to control the maximum number of simultaneous calls. It can be set by command line, **-l limit**, from the client end. It should be set to a large number, such as 5000, for heavy load.

5.3 Timer Resolution

This parameter can be set by command line, **-timer_resol ms**, from the client end. The default timer resolution is 200ms. We want a more precise scheduling, this value is set to 50ms.

5.4 Frequency

The call rate can be set by command line, *-r rate* and *-rp period*, from the client end. The real call rate is rate/period cps. This parameter should be set according to two facts. First, the total SIP traffic load should reach a certain level. For example, CPU usage of OpenSER with RTPproxy should reach a certain level. Another fact is the capability of the OSP servers. If the OSP servers cannot handle the heavy traffic load, more OSP servers should be used.

5.5 Call Numbers

This parameter is used to stop the test when the total number of the calls reaches this parameter. It can be set by command line, *-m number*, from the client end.

5.6 Screen Flush Frequency

This parameter is used to set the flush frequency of displaying test results on the screen. It can be set by command line, *-f Ns*, from the client end. In order to reduce the overhead, this value should be set to a large number, such as 30.

6 Scripts for Running the Test

6.1 Data Collection Scripts

Sar is used to collect CPU, network and memory usage data. Sar write script is used to start sar to write test info into a data file. Sar read script is used after test to collect the test info from the data file.

6.1.1 Sar Write Script

```
# Usage: wsar.sh count
# Count is based on 10s internal.
rm -rf ./data.sar
sar -o ./data.sar 10 $1 >/dev/null 2>&1 &
```

The count should be set to a value longer than test duration. For example, the test duration is around 15 minutes, since sar normally is started minutes earlier than the test, the count should be at least 90.

Note: The old data.sar must be erased. Otherwise, sar read script may collect wrong data.

6.1.2 Sar Read Script

```
# Usage: rsar.sh start end
# Time in hh:mm:ss format
sar -u -r -n DEV -f ./data.sar -s $1 -e $2
```

Since sar records the test data based on local time, to display the test data, the local time must be remembered.

Note: sar on t2000 does not support *-n* options.

6.2 RTPproxy Script

```
rtpproxy -l 172.16.4.32 -s unix:/var/run/rtpproxy.sock
```

6.3 SIPp Scripts

For SIPp client, we need to collect data on the following time intervals:

- Time from INVITE sent until first Trying message.
- Time from first Trying message to second Trying message.
- Time from second Trying message to RINGING message.

Note:

1. SIPp server ends should be started before any SIPp client is started.
2. Both SIPp client and server ends need root rights to start to use pcap feature.

6.3.1 SIPp Client

```
sipp -sf client.xml -m 600 -r 1 -rp 1s -l 5000 -s 14040000099 -nd -pause_msg_ign -
timer_resol 50 -f 30 172.16.4.32:5060 -trace_stat
```

Note:

1. This script is for 1cps, 10 minutes duration test.
2. *-trace_screen* can be used to collect the test results.

6.3.2 Media Source

```
sipp -sf media.xml -m 1 -s 14040000000 -nd -pause_msg_ign -f 10 172.16.4.32:5060
```

6.3.3 SIPp Server / Media Destination

```
sipp -sf server.xml -nd -p 5060 -rtp_echo
```

Note: The SIPp server ends can be run at background using *-bg* command line parameter.

7 Test Procedure

7.1 Start Test

1. Start Asterisk on 172.16.4.56.
2. Start SIPp servers on 172.16.4.60/25.
3. Stop OSP Server instance on 172.16.4.75.
4. Erase all old CDRs.
5. Start OSP server instance.
6. Start RTPproxy on 172.16.4.32.
7. Start OpenSER on 172.16.4.32.
8. Start sar write script on 172.16.4.32.
9. Start SIPp clients on 172.16.4.34/35/22/36 using root account.
10. Start tethereal to capture traffic track and start media destination.
11. Start tethereal to capture traffic track and start media source.

7.2 After Test

1. Collect statistics data from client.xml_<pid>_csv on the SIPp clients.
2. Collect call data from screen or client_<pid>_screen.log on the SIPp clients if *-trace_screen* is set.
3. Collect CDRs from all OSP server instances and calculate the numbers of different CDRs.
4. Use */usr/local/sbin/openserctl fifo get_statistics shmем:* on the OpenSER host to collect maximum used shared memory data for OpenSER.
5. Collect test results using sar read script on the OpenSER host.
6. Collect the traffic tracks on the media source/destination.

7. Run CDR pull/file audit/sort/assemble/expire on NexOSS to generate traffic report.

8 Test Results

- The test results will be displayed on the SIPp client end screens. The total number of calls, successful calls, and failed calls will be displayed.

```
----- Scenario Screen ----- [1-9]: Change Screen --
Call-rate(length)      Port  Total-time  Total-calls  Remote-host
1.0(0 ms)/1.000s     5060    786.41 s      600  172.16.4.32:5060(UDP)

Call limit reached (-m 600), 6.415 s period  0 ms scheduler resolution
0 calls (limit 5000)                          Peak was 187 calls, after 203 s
0 Running, 0 Paused, 0 Woken up
0 out-of-call msg (discarded)
0 open sockets
5273094 Total RTP pkts sent                    0.000 last period RTP rate (kB/s)

Messages  Retrans  Timeout  Unexpected-Msg
INVITE -----> B-RTD1 600      0        0          0
  100 <----- B-RTD2 600      0        0          0
  100 <----- B-RTD3 600      0        0          0
  180 <----- E-RTD3 600      0        0          0
  200 <-----      600      0        0          0
ACK ----->      600      0
  [ NOP ]
Pause [ 3:00]      600
  [ NOP ]
Pause [ 1000ms]    600
BYE ----->      600      0        0          0
  200 <-----      600      0        0          0

----- Test Terminated -----

----- Statistics Screen ----- [1-9]: Change Screen --
Start Time      | 2008-07-22 02:22:29:096 | 1216707749.096190
Last Reset Time | 2008-07-22 02:35:29:101 | 1216708529.101083
Current Time    | 2008-07-22 02:35:35:516 | 1216708535.516823
-----+-----+-----
Counter Name    | Periodic value          | Cumulative value
-----+-----+-----
Elapsed Time    | 00:00:06:415           | 00:13:06:420
Call Rate       | 0.000 cps               | 0.763 cps
-----+-----+-----
Incoming call created | 0                       | 0
OutGoing call created | 0                       | 600
Total Call created  | 5                       | 600
Current Call       | 0                       | 0
-----+-----+-----
Successful call    | 5                       | 600
Failed call        | 0                       | 0
-----+-----+-----
Response Time 1   | 00:00:00:000           | 00:00:00:000
Response Time 2   | 00:00:00:000           | 00:00:00:047
Response Time 3   | 00:00:00:000           | 00:00:02:737
Call Length      | 00:03:05:216           | 00:03:03:859
-----+-----+-----
----- Test Terminated -----
```

Note:

- The best condition is all calls complete without any unexpected message and without any message loses.

2. Missing SIP 180 Ringing messages, if the calls are completed, is acceptable when the traffic is heavy.
3. A low percentage call failing may be acceptable for very heavy traffic conditions.

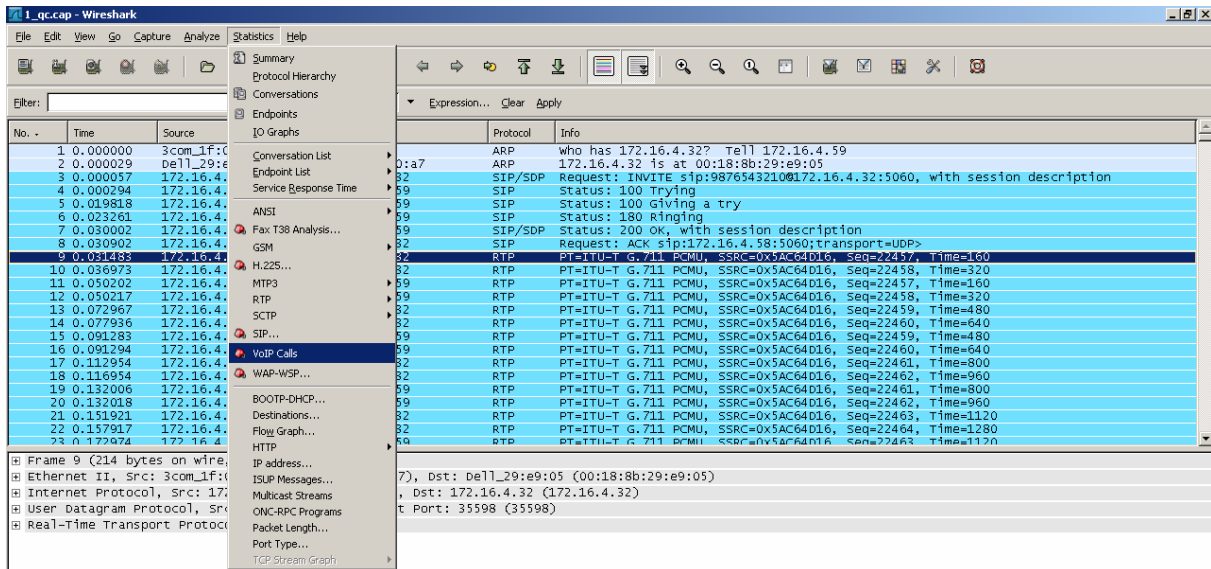
- The number of current calls can be obtained from SIPp client statistics files.

...	TotalCallCreated	CurrentCall	SuccessfulCall(P)	SuccessfulCall(C)	FailedCall(P)	FailedCall(C)	...
	0	0	0	0	0	0	
	29	29	0	0	0	0	
	59	59	0	0	0	0	
	89	89	0	0	0	0	
	119	91	28	28	0	0	
	150	93	29	57	0	0	
	180	93	30	87	0	0	
	210	92	31	118	0	0	
	240	94	28	146	0	0	
	270	92	32	178	0	0	
	300	93	29	207	0	0	
	300	64	29	236	0	0	
	300	33	31	267	0	0	
	300	4	29	296	0	0	
	300	0	4	300	0	0	

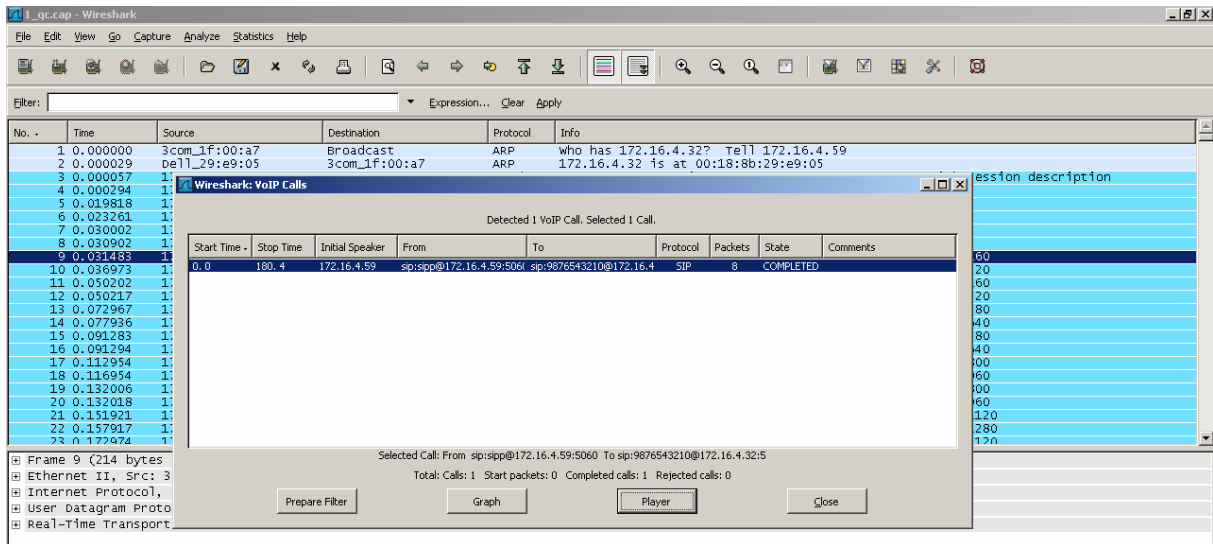
We use the middle 3 data values to calculate the average CPU usage.

- The numbers of different CDRs can be calculated using the command `grep -v QUALITY *.cdr |grep -P "\tTCCode\t" | wc -l`
- The NexOSS report may contain the calls for other tests.
- The call quality data can be obtained from the traffic tracks using wireshark.

1. Select Statistics->VoIP Calls from the main window

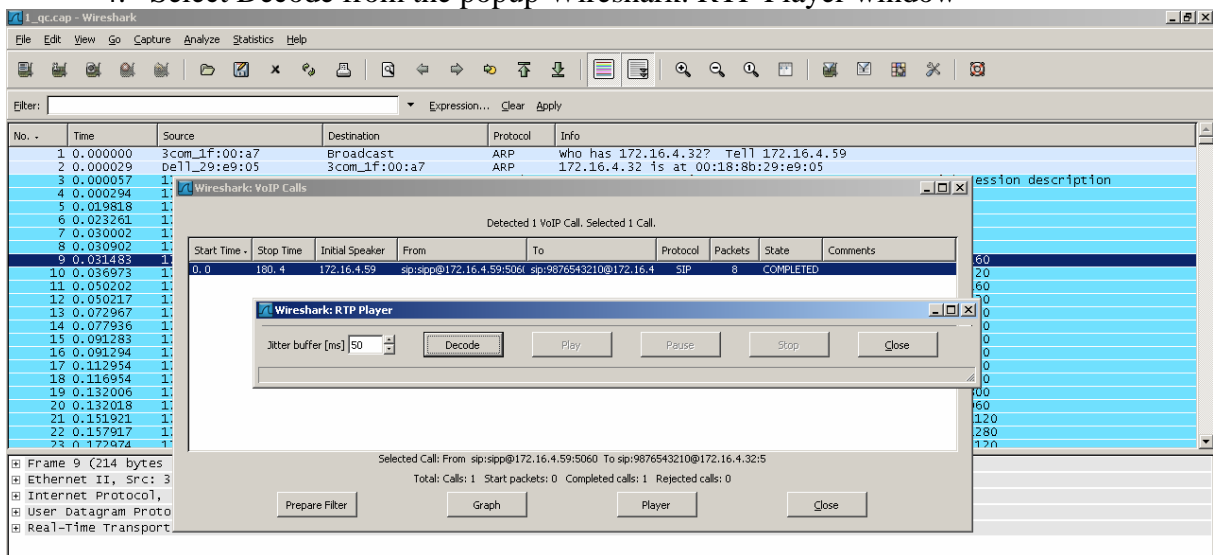


2. Select the call quality test call in the popup Wireshark: VoIP Calls window

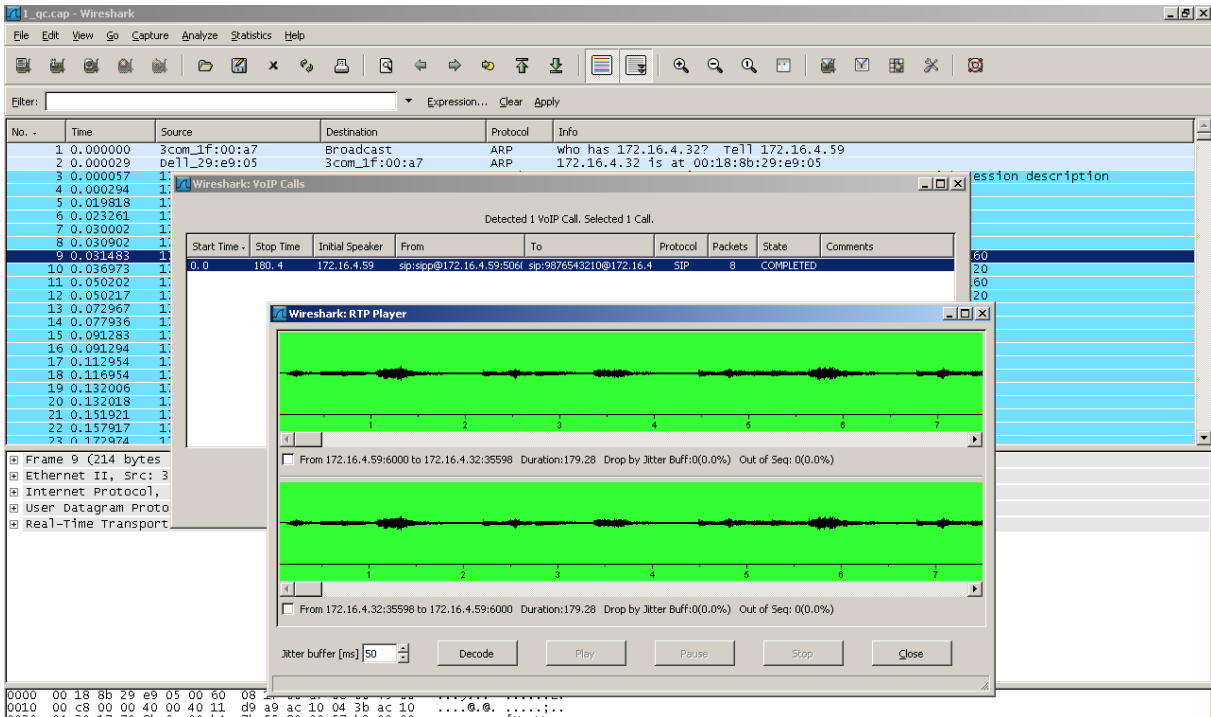


3. Select Player button

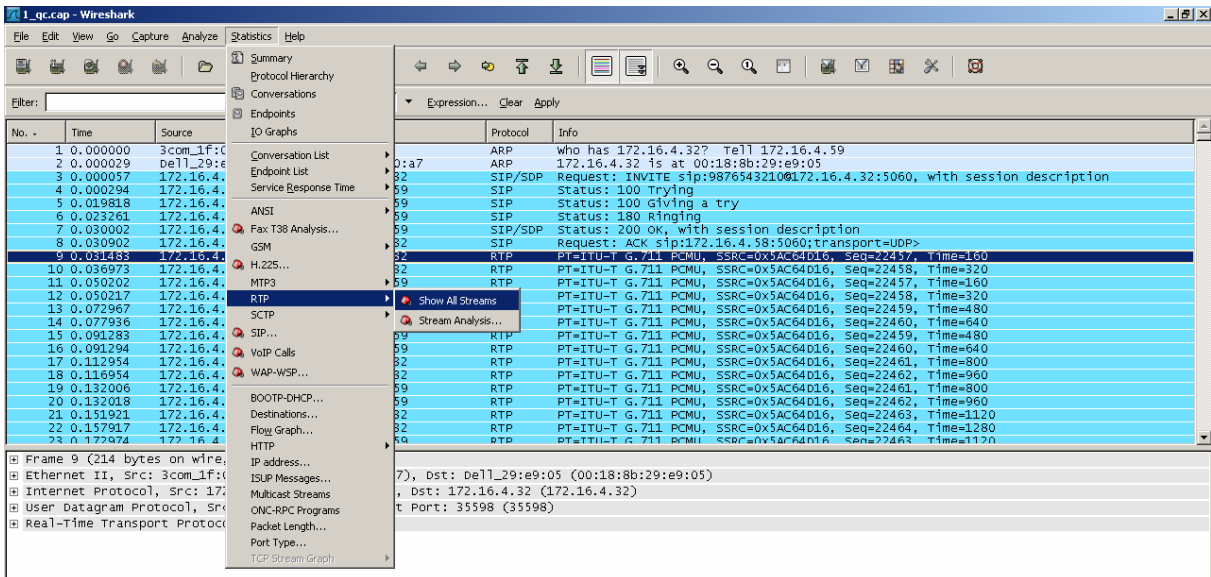
4. Select Decode from the popup Wireshark: RTP Player window



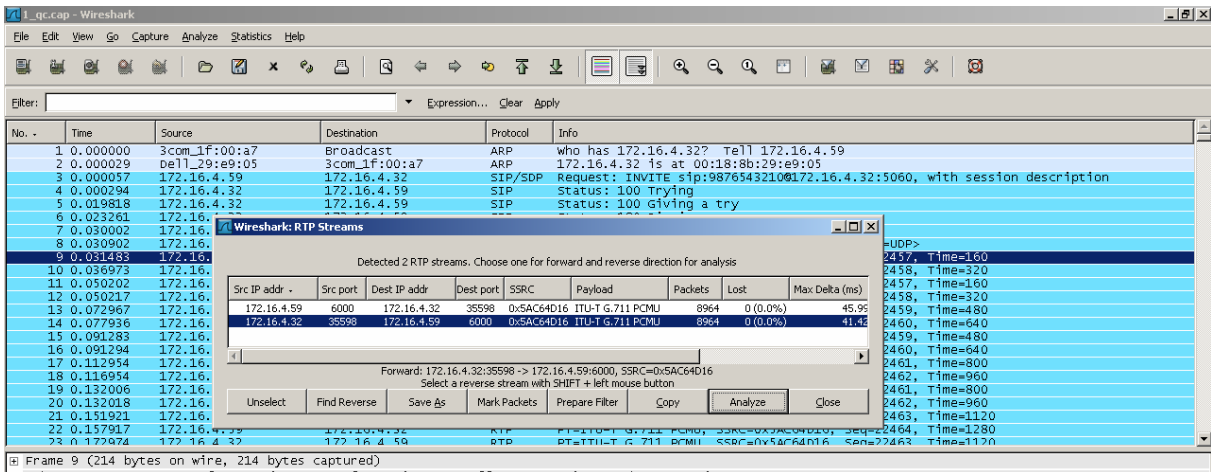
5. It will show the number of RTP packet dropped by Jitter Buffer and Out of Seq.



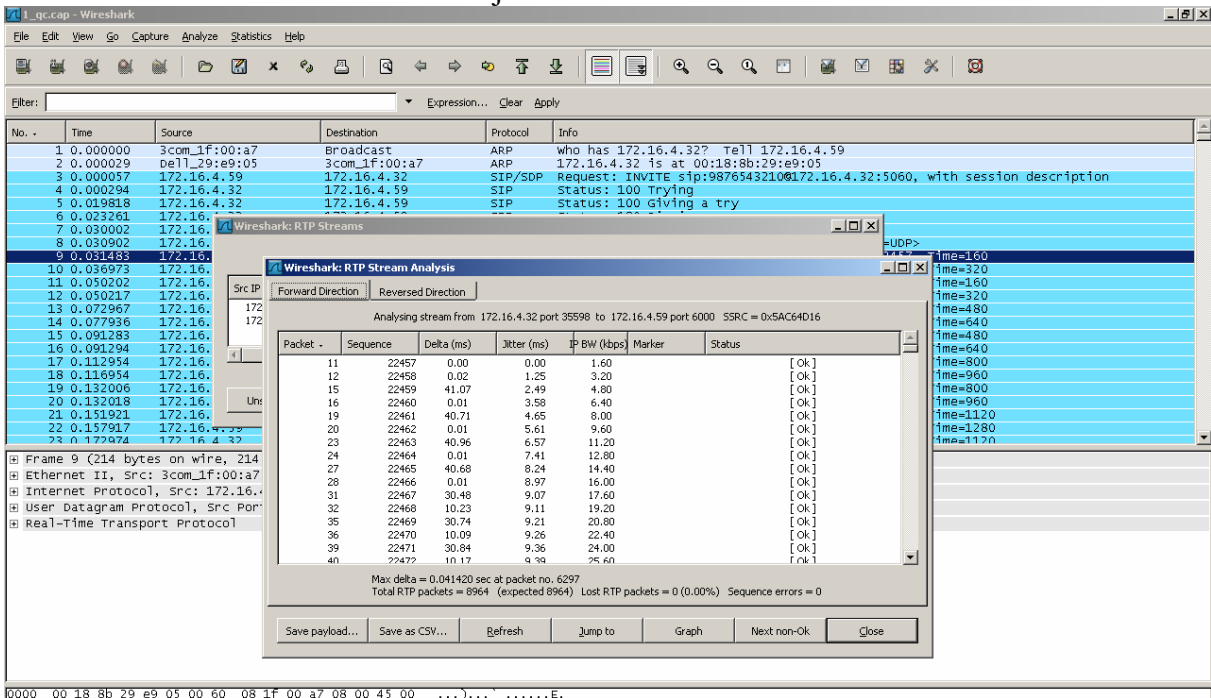
6. Select Statistics->RTP->Show All Streams from the main window



7. Select the echo stream in the popup Wireshark: RTP Streams window

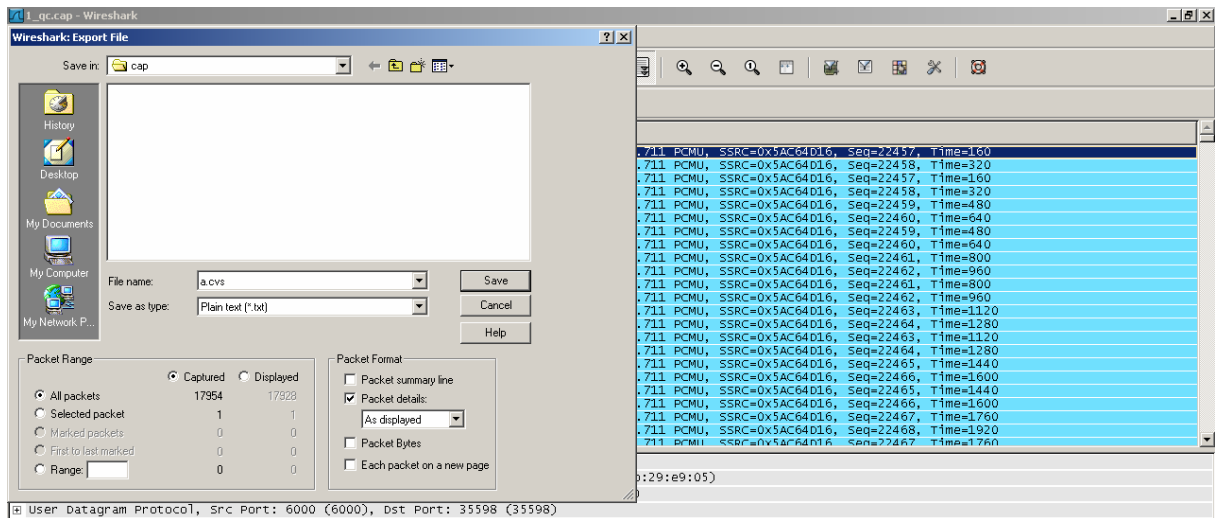


8. Select Analyze button
9. It will show the calculated jitter buffer data.



10. Save it to a CSV file then use Excel to calculate max, min and average values.

- The round trip delay data can be calculated from the captured track since the RTP stream was echoed back. Export the RTP messages to a CSV file, then use Excel to calculate the arriving time difference of the matched RTP messages.



Note: It is a little bit tricky that you have to match every pair of RTP messages using Seq and Time attributes.

Note: There are several data about the OpenSER with RTPproxy performance:

- Number of attempts
- SIPp report
- OSP Server CDRs
- NexOSS report

The number of completed calls reported by the SIPp clients is most important. This is the number of completed calls reported by the call source.

NexOSS reports the number of completed calls, the number of failed attempts and the reasons that come from the CDRs the OSP server reports. The number of completed calls reported by the OSP server and NexOSS should be equal. But the OSP server may report more 10016 CDRs caused by the re-sending BYE message out of OpenSER absorb timeout.

The relationship of the numbers should be:

- Number of internal CDRs == 5 * number of test calls
- Number of SIPp completed calls <= number of 16 CDRs

There is another reason that the number of 10016 CDRs may be more than the number of completed calls except the duplication explained above. A 200 OK message for a BYE message may reach OpenSER, it reports a 10016 CDR for it, but the 200 OK does not reach the SIPp client, it counts this call as failure.

8.1 Actual Results

The following pages present raw data collected from test:

Date Test Performed	3/26/08	Host OS	2.6.21-1.3194.fc7 #1 SMP		
Test Performed by	Di-Shi	Host CPU	Intel Xeon 5140 @ 2.33 GHz - 1 core used		
OpenSER Version	1.3	Host Memory	4 GB		
Test Rate (cps)		0.5	1	1.5	2
SIPp Traffic Report	Completed	300	600	900	1200
	Completed%	100	100	100	100
	Simu. Expected	90	180	270	360
	Simu. Actual	93.00	184.00	276.00	368.67
Call Quality					
Lost RTP Packets	Jitter Buffer	0	0	0	0
	Jitter Buffer (%)	0	0	0	0
	Out of Seq	0	0	0	0
	Out of Seq (%)	0	0	0	0
Round Trip Delay (ms)	Max	21.385	22.229	22.985	23.954
	Min	7.273	8.698	9.988	9.749
	Average	15.578	15.813	16.111	16.258
	Median	15.589	15.791	16.094	16.269
Jitter Buffer (ms)	Max	18.64	18.85	19.28	19.37
	Average	15.07	15.67	15.85	16.47
	Median	15.08	15.71	15.92	16.61
Analysis of OSP Server CDRs					
Internal CDRs	Expected	1500	3000	4500	6000
	Actual	1500	3000	4500	6000
408 CDRs	Expected	~450	~900	~1350	~1800
	Actual	467	816	1188	1970
404 CDRs	Expected	~150	~300	~450	~600
	Actual	150	285	399	507
200 CDRs	Expected	300	600	900	1200
	Actual	300	600	900	1200
10016 CDRs	Expected	300	600	900	1200
	Actual	300	600	900	1200
All Other Error CDRs	Expected	0	0	0	0
	Actual	0	0	0	0
NexOSS Traffic Report					
SIP 408	Expected	~450	~900	~1350	~1800
	Actual	467	816	1188	1970
SIP 404	Expected	~150	~300	~450	~600
	Actual	150	285	399	507
16 Normal Call Clearing	Expected	300	600	900	1200
	Actual	300	600	900	1200
All Other	Expected	0	0	0	0
	Actual	0	0	0	0
System Utilization					
OpenSER with RTPproxy	CPU idle (%)	92.38	84.05	76.22	65.55
	IO wait (%)	0.05	0.05	0.05	0.05
	RX/TX (MB/s)	1.94/1.94	3.89/3.89	5.76/5.76	7.70/7.71
	Memory (%)	29.79~30.22	10.90~11.39	10.80~11.32	11.92~12.47
	Shmem (MB)	1.90	2.13	2.38	2.66
SIPp Average Response Time Repartition					
Invite to 1st 100 message	< 50 ms	300	600	900	1200
	< 100 ms	0	0	0	0
	< 200 ms	0	0	0	0
	< 500 ms	0	0	0	0
	> 500 ms	0	0	0	0
1st 100 to 2nd 100 message	< 50 ms	283	466	737	931
	< 100 ms	17	134	163	269
	< 200 ms	0	0	0	0
	< 500 ms	0	0	0	0
	> 500 ms	0	0	0	0

2nd 100 to Ringing	< 50 ms	64	149	260	274
	< 100 ms	0	0	0	0
	< 200 ms	0	0	0	0
	< 500 ms	0	0	0	0
	< 1 s	0	0	0	0
	< 2 s	73	181	268	242
	< 3 s	0	0	1	4
	< 4 s	95	175	190	314
	< 5 s	0	0	0	2
	< 6 s	68	95	179	336
	< 10 s	0	0	0	28
	> 10 s	0	0	0	0

Date Test Performed	3/26/08	Host OS	2.6.21-1.3194.fc7 #1 SMP		
Test Performed by	Di-Shi	Host CPU	Intel Xeon 5140 @ 2.33 GHz - 1 core used		
OpenSER Version	1.3	Host Memory	4 GB		
Test Rate (cps)		2.5	3	3.5	4
SIPp Traffic Report	Completed	1500	1800	2100	2400
	Completed%	100	100	100	100
	Simu. Expected	450	540	630	720
	Simu. Actual	461.00	552.00	647.00	733.67
Call Quality					
Lost RTP Packets	Jitter Buffer	0	0	0	0
	Jitter Buffer (%)	0	0	0	0
	Out of Seq	0	0	0	0
	Out of Seq (%)	0	0	0	0
Round Trip Delay (ms)	Max	25.070	27.643	26.716	37.516
	Min	5.568	9.746	9.665	9.209
	Average	16.428	16.695	16.973	17.758
	Median	16.427	16.687	16.947	17.757
Jitter Buffer (ms)	Max	20.12	20.32	20.27	20.93
	Average	16.96	17.55	18.04	18.41
	Median	17.16	17.97	18.66	19.49
Analysis of OSP Server CDRs					
Internal CDRs	Expected	7500	9000	10500	12000
	Actual	7500	9000	10500	12000
408 CDRs	Expected	~2250	~2700	~3150	~3600
	Actual	2291	2628	3255	3693
404 CDRs	Expected	~750	~900	~1050	~1200
	Actual	736	778	917	1135
200 CDRs	Expected	1500	1800	2100	2400
	Actual	1500	1800	2100	2400
10016 CDRs	Expected	1500	1800	2100	2400
	Actual	1500	1800	2100	2400
All Other Error CDRs	Expected	0	0	0	0
	Actual	0	0	0	0
NexOSS Traffic Report					
SIP 408	Expected	~2250	~2700	~3150	~3600
	Actual	2291	2628	3255	3693
SIP 404	Expected	~750	~900	~1050	~1200
	Actual	736	778	917	1135
16 Normal Call Clearing	Expected	1500	1800	2100	2400
	Actual	1500	1800	2100	2400
All Other	Expected	0	0	0	0
	Actual	0	0	0	0
System Utilization					
OpenSER with RTPproxy	CPU idle (%)	55.54	44.16	30.87	14.50
	IO wait (%)	0.03	0.03	0.02	0.00
	RX/TX (MB/s)	9.61/9.62	11.55/11.56	13.52/13.53	15.37/15.38
	Memory (%)	10.71~11.29	12.19~12.83	11.11~11.75	10.68~11.34
	Shmem (MB)	2.79	3.00	3.23	3.36
SIPp Average Response Time Repartition					
Invite to 1st 100 message	< 50 ms	1500	1800	2100	2400
	< 100 ms	0	0	0	0
	< 200 ms	0	0	0	0
	< 500 ms	0	0	0	0
	> 500 ms	0	0	0	0
1st 100 to 2nd 100 message	< 50 ms	666	1048	1110	818
	< 100 ms	833	751	989	1560
	< 200 ms	1	0	0	22
	< 500 ms	0	1	1	0
	> 500 ms	0	0	0	0

2nd 100 to Ringing	< 50 ms	349	496	524	636
	< 100 ms	0	0	0	0
	< 200 ms	0	0	0	0
	< 500 ms	0	0	0	0
	< 1 s	0	0	0	0
	< 2 s	361	359	425	454
	< 3 s	11	12	13	9
	< 4 s	396	519	574	645
	< 5 s	22	23	23	28
	< 6 s	339	361	509	593
	< 10 s	22	30	32	35
	> 10 s	0	0	0	0

Date Test Performed	7/18/08	Host OS	2.6.21-1.3194.fc7 #1 SMP		
Test Performed by	Di-Shi	Host CPU	Intel Xeon 5140 @ 2.33 GHz - 1 core used		
OpenSER Version	1.3	Host Memory	4 GB		
Test Rate (cps)		4.25	4.5	4.75	
SIPp Traffic Report	Completed	2550	2700	2850	
	Completed%	100	100	100	
	Simu. Expected	765	810	855	
	Simu. Actual	781.00	826.33	873.67	
Call Quality					
Lost RTP Packets	Jitter Buffer	0	108	2050	
	Jitter Buffer (%)	0	1.2	27.0	
	Out of Seq	0	0	634	
	Out of Seq (%)	0	0	8.4	
Round Trip Delay (ms)	Max	0.050280	0.102812	14.713074	
	Min	0.009201	0.009568	0.154134	
	Average	0.019703	0.035728	9.164218	
	Median	0.019174	0.034590	11.686529	
Jitter Buffer (ms)	Max	21.56	26.39	42.99	
	Average	18.92	19.68	37.44	
	Median	18.68	19.79	37.34	
Analysis of OSP Server CDRs					
Internal CDRs	Expected	12750	13500	14250	
	Actual	12750	13500	14250	
408 CDRs	Expected	~3925	~4050	~4275	
	Actual	3830	4105	4573	
404 CDRs	Expected	~1275	~1350	~1425	
	Actual	1292	1303	1357	
200 CDRs	Expected	2550	2700	2850	
	Actual	2550	2700	2850	
10016 CDRs	Expected	2550	2700	2850	
	Actual	2550	2700	2850	
All Other Error CDRs	Expected	0	0	0	
	Actual	0	0	0	
NexOSS Traffic Report					
OSIP 408	Expected	~3925	~4050	~4275	
	Actual	3830	4105	1357	
SIP 404	Expected	~1275	~1350	~1425	
	Actual	1292	1303	1357	
16 Normal Call Clearing	Expected	2550	2700	2850	
	Actual	2550	2700	2850	
All Other	Expected	0	0	0	
	Actual	0	0	0	
System Utilization					
OpenSER with RTPproxy	CPU idle (%)	5.46	0.43	0	
	IO wait (%)	0	0	0	
	RX/TX (MB/s)	16.35/16.36	17.02/17.03	17.03/15.80	
	Memory (%)	22.91~23.13	33.89~34.18	35.34~35.34	
	Shmem (MB)	3.52	3.66	3.81	
SIPp Average Response Time Repartition					
Invite to 1st 100 message	< 50 ms	2550	2700	2850	
	< 100 ms	0	0	0	
	< 200 ms	0	0	0	
	< 500 ms	0	0	0	
	> 500 ms	0	0	0	
1st 100 to 2nd 100 message	< 50 ms	532	680	534	
	< 100 ms	1948	1913	705	
	< 200 ms	62	106	891	
	< 500 ms	8	1	700	
	> 500 ms	0	0	0	

2nd 100 to Ringing	< 50 ms	616	642	661	
	< 100 ms	0	0	0	
	< 200 ms	0	0	0	
	< 500 ms	0	0	0	
	< 1 s	8	20	23	
	< 2 s	596	656	567	
	< 3 s	8	19	40	
	< 4 s	750	695	772	
	< 5 s	5	22	47	
	< 6 s	565	642	733	
	< 10 s	2	4	7	
	> 10 s	0	0	0	