SURF Communication Solutions

SURF HMP getting started guide

Document version: 1.3
Date: May 2015

SURF Communication Solutions, Ltd.

Tavor Building, P.O. Box 343

Yokne'am 20692 Israel

Tel: +972 (0)73 714-0700

Fax: +972 (0)4 959-4055

Web site	http://www.surfsolutions.com/
Email	info@surfsolutions.com

Table of Contents

1. Introduction

1.1 Abstract

The document's main goal is to help system integrators to quickly setup SURF HMP solution and start using it.

1.2 Architecture

SURF HMP solution is designed to be flexible and support as many hardware platforms as possible. Currently it is able to run on a platform with single or multiple Intel 64 bit processors. Limitation of using Intel processors is caused only by Voice codecs that are optimized for Intel SIMD command set extensions.

Several key guidelines of the SURF HMP are:

- Linux was chosen for the OS layer of the solution because of stability, flexibility and wide range of hardware supported.
- The solution is implemented as a multi-threaded process. The number of threads is configurable according to the system capabilities.
- In the context of the SURF HMP product, the above process is referred to as "MediaProcessor".

1.3 API protocol quick guide

To perform on-the-fly configurations of the SURF HMP solution, an API protocol should be used. This protocol is described in the API specification document. This protocol is implemented over TCP/IP as a transport layer and uses the JSON standard as an API syntax standard. Several key points about the SURF HMP API protocol are:

- The API is running on top of TCP/IP
- The API is message based
- Each message consists of 4 byte length in little endian format + byte array of length that is contained in the 4 first bytes.
- A byte array contains text message in UTF-8 encoding
- A text message is in JSON format
- Each text message is represented by { <message type> : { <message description> }} format
- Messages are asynchronous
- Most of parameters in messages have default values, if not specified – default values will be used. This allows to use this API both as a high level API when not specifying most of

the fields, as well as very low level API when using all possible configurations and features

1.4 JSON standard

JavaScript Object Notation (JSON, RFC 4627) is a lightweight, text-based, language-independent data interchange format. JSON is a text format for the serialization of structured data. It defines a small set of formatting rules for the portable representation of structured data.

JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).

JSON is widely used today in mobile applications and the main purpose of this language is to replace heavy XML messages in various configuration protocols that do not use wide range of XML features. It is natively supported in internet browsers like mozilla firefox and google chrome.

1.5 About sample applications

When the MediaProcessor is run, it does nothing except listening for TCP connections in order to receive commands for its configuration.

In order to configure the MediaProcessor a control application is sending API commands over TCP to the MediaProcessor. Since the protocol is designed to be human readable and user-friendly, the application can be written in a very short time.

Any programming language can be used to write such a control application; SURF is providing applications simple to use written in PERL language in this document. These applications are called "sample applications" further in this document.

1.6 The "Tool" concept

The MediaProcessor operates with "tools" from different types. Tool is the main building block for SURF HMP users. To implement given scenario using SURF HMP one must configure set of tools (either of different types of the same type, depends on specific scenario). This configuration includes the connections between tools: tools can pass data from one to another and this is also configured by the configuration messages.

2. Package directory structure

- Doc – contains the documentation including this document
- HMP – contains the main executable, required shared libraries and the HMP configuration file
    - MediaProcessor/MediaProcessorEvaluation – executable
    - config.json – configuration file to be used with the HMP
- HostApp – host application related data
    - Scripts – sample test scripts described later in this document
    - SurfView – sample application with GUI which monitors SURF HMP activity
- RtpLoader – data for loading the SURF HMP with RTP input that should be used in part of supplied test scenarios

3. Setting up the system

3.1 System requirements

The following is the list of minimal system requirements:

- Intel 64-bit processor based machine
- 64-bit linux system

3.1.1 File descriptors configuration

Linux operating system limits the number of file descriptors that can be opened simultaneously. Network sockets are also represented as file descriptors in linux systems.

Each VoIP tool in the system uses UDP socket for RTP receive/send and another socket for RTCP (if configured). Moreover file descriptors are also used to save debug traces (if configured).

All this should be taken into account when calculating the max number of file descriptors that can co-exist in the system.

Most linux systems limit the number of FDs to 1024 by default, but it can be configured.

Look for specific linux distribution help to change this configuration; in most distributions this configuration is located in /etc/security/limits.conf file

Add the following text at the end of this file (replace <username> with linux user name that you are using):

<username> hard nofile 16384

<username> soft nofile 16384

Reboot the machine after that.

3.1.2 Ethernet configuration

Every Ethernet device in linux has its internal transmit queue that is limited. Usually its size is configured to 1000. In some cases this can be a bottle neck in UDP transmission and can cause packet loss. To avoid this tx queue length can be increased by the following command (root privileges is needed to do so):

Ifconfig <ifc_name> txqueuelen <queue_len>

For example:

Ifconfig eth0 txqueuelen 6000

For advanced users only:

The following system parameters may be also subject for configuration in case the packet loss on rx or tx still exists:

- net.ipv4.udp_mem
- net.ipv4.udp_rmem_min
- net.ipv4.udp_wmem_min
- net.core.rmem_default
- net.core.rmem_max
- net.core.wmem_default
- net.core.wmem_max

These parameters can be configured by the following shell command (required root privileges):

```
sysctl –w <param_name>=<value>
```

3.1.3 firewall configuration

Sometimes linux built-in firewall would block network UDP connections. In order to avoid it, use the following command (on several linux distributions firewall service can only be stopped using different commands, in this case refer to the specific distribution manual)

service iptables stop

or (depends on specific linux distro)

systemctl stop firewalld

3.2 Running the SURF HMP solution

To run the MediaProcessor use linux shell to change to the HMP directory inside the package.

Then run **./MediaProcessor <config file name>**

Example: **./MediaProcessor config.json**

config.json file is already included in the package, but it should be tuned for specific machine's settings to achieve the best performance.

3.2.1 Configuration file format

Configuration file is written in JSON format (RFC 7159)

It contains the following parameters:

- worker_threads – number of concurrent worker threads created by MediaProcessor. Number of physical threads supported by the CPU should be used in order to reach highest density
- tcp_configuration_port – TCP listen port, this port will be used to listen for incoming connection of the controller application that will configure the MediaProcessor
- block_size – block size in milliseconds. All voice and voice related tools are processed once in a "block_size"; valid values are 5,10,15,20,25 and 30 milliseconds
- logger - enables/disables logger thread; can be used to save CPU time
- file_play – enables/disables file read thread; can be used to save CPU time

- error_trace – if set, automatically enables error trace and sets the file name for it. It is recommended to use this option

Example:

```
{"configuration":{
   "worker_threads":8,
   "tcp_configuration_port":9000,
   "block_size":20,
   "logger":true,
   "file_play":true,
   "error_trace":"error_trace.txt"
}}
```

3.3 Checking CPU consumption

In order to check whether the MediaProcessor has enough time to handle all the configured tools, CPU usage statistics sent by the MediaProcessor should be used.

The CPU consumption number is calculated as a relative time needed to process all the tools out of 20 milliseconds block size. For example, if it takes to the MediaProcessor 10 milliseconds to process all the tools once, the CPU consumption will be calculated as 50%.

If the CPU consumption is 100%, it means that there are tools that are late for their deadlines and its output will not be sent on time. This can cause a serious delay.

Additional value that is also provided by the MediaProcessor is "number of late iterations". This value represents number of run iterations (during one iteration the MediaProcessor runs all the "periodic" tools for one time) that have not met the real-time requirement. "Periodic" tools are voice_p2p and voice_mixer tools. Number of late iterations value is relevant only to periodic tools, other tools should be monitored according to "CPU" value.

Provided sample tests are designed to receive and print those CPU statistics messages. It is possible to look at those. This information is printed to the terminal in the following format:

CPU usage: 45 nof_late_iterations: 0

Note that CPU value should be under 100 and nof_late_iterations should be 0 in case everything functions normally.

3.4 Checking packet loss

When testing the SURF HMP under high network traffic, for example voice trans-coding scenarios, it is recommended to check the global packet loss statistics to make sure that all the voice tools receives its input normally.

This can be done using "sys_inf" message with "type":"network" field. Refer to the SURF HMP API document for more details.

The supplied sample tests in the package already handle this type of statistics and periodically prints it to the terminal in the following format:

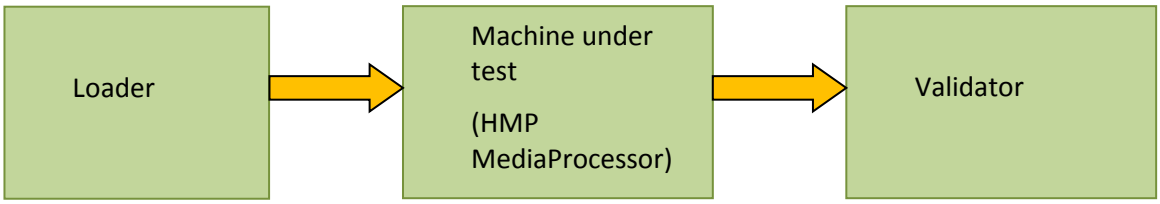
Network packet loss: 0

This value represents accumulative global packet loss counter for all voice_p2p channels in the system since its startup. In heavy load scenarios always check that this value remains constant.

4. Testing Scenarios

4.1 Testing setup

Up to 3 entities will be used in this testing setup, depending on specific scenario



All 3 entities can be run on the same physical host, or can be divided into 2 or 3 different hosts, but for heavy load tests it is recommended to separate all 3 entities to 3 different hosts.

Loader is used to "load" the system with input – it supplies the HMP with the required input RTP streams. It also runs python testing scripts which configure the HMP.

Machine under test runs HMP MediaProcessor process. This is the part of the setup that is actually tested.

Validator is used to collect HMP output and analyze it.

These machines will be reffered later as "loader machine", "under test machine" and "validator machine"

4.2 Sample scripts

There are several sample scripts available in the package for demonstration purposes. Demo scripts are written in python.

Each script connects to a MediaProcessor and configures its predefined scenario. After that some of the scripts can perform some other tasks like handling MediaProcessor event and status messages.

Every script has its own internal parameters like number of streams to be opened and so on. However, there are some general parameters for all scripts, which are contained in general_params.py file:

- HMP_IP – IP address of the under test machine
- HMP_PORT – TCP port on which the MediaProcessor process on listens on the under test machine for incoming connections
- REMOTE_IP – validator machine IP address. Some scripts (but not all) will configure the HMP to send its output to this IP

Except general_params.py, there is another python module that is used by all other scripts that implements common functionalities like connecting to HMP, receiving and sending JSON messages, etc. This script is named hmp_connection.py

To run a testing scenario, follow these steps:

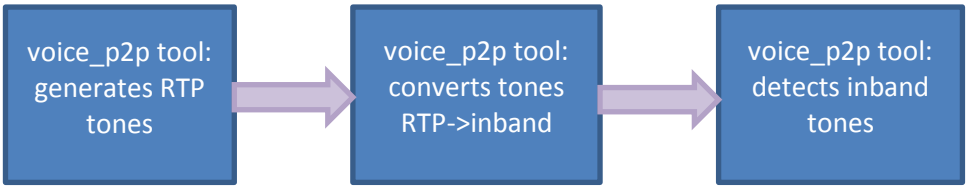
- Run MediaProcessor as described earlier in this document (on "under test machine")
- Run required test python script on "loader machine"
- Perform additional steps for a specific script as described below
- MediaProcessor's output (if exists) will be sent to "validator machine" – it can be captured by tcpdump/wireshark and validated
- When finished, run clear.py script on the "loader machine" to clean up the MediaProcessor's configuration

4.2.1 clear.py sample script

This script connects to the MediaProcessor and sends to it command to remove all existing tools. Must be used between executions of other scripts in order to clean MediaProcessor configuration.

4.2.2 dtmf_conversion.py sample script

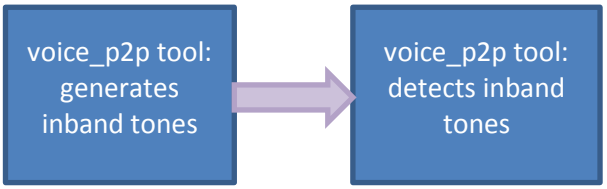
General HMP tool scheme:



This script demonstrates conversion of RTP DTMF tones to inband DTMF tones. 3 voice_p2p tools are created for this purpose. The first one generates RTP tones; its output is connected to the second tool's input. The second tool converts RTP tones to inband voice tones and sends it to its output which is in turn connected to the third tool's input. The third tool detects those inband tones and notifies the script.

4.2.3 dtmf_det_inband.py sample script

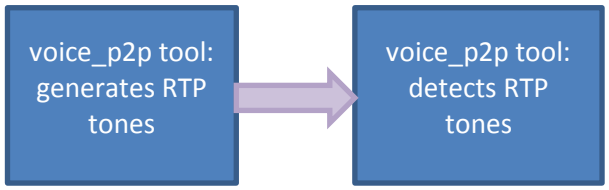
General HMP tool scheme:



This script demonstrates inband DTMF detection functionality. This scenario contains 2 voice_p2p tools: the first one generates inband DTMFs and the second one detects it and notifies the script.

4.2.4 dtmf_det_rfc4733.py sample script

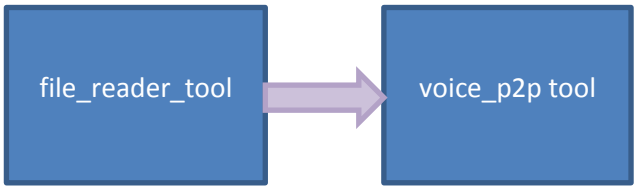
General HMP tool scheme:



This script demonstrates RTP DTMF detection functionality. This scenario contains 2 voice_p2p tools: the first one generates RTP DTMFs and the second one detects it and notifies the script.

4.2.5 voice_file_reader_demo.py sample script

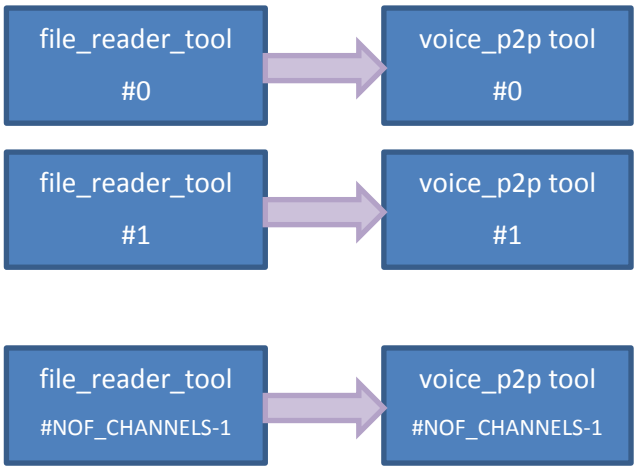
General HMP tool scheme:



This script demonstrates file play functionality. It configures 2 tools: file_reader and voice_p2p. File reader tool receives a play list from configuration, reads corresponding files and passes its content to voice tool. The voice tool trans-codes the received voice and sends it to RTP destination.

4.2.6 voice_file_reader_load.py sample script

General HMP tool scheme:



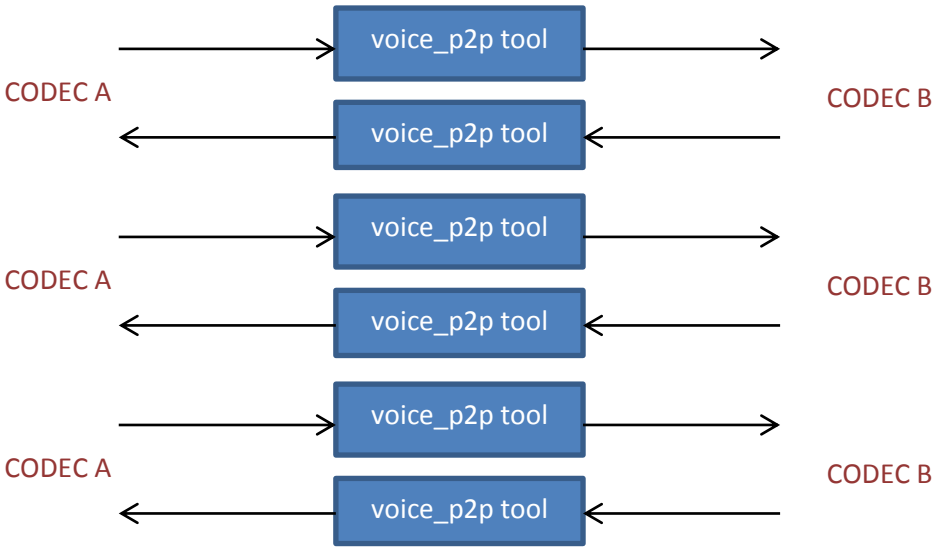
This script can be used for density testing in file play scenario. It creates required number of tools pairs when each pair contain file_reader and voice_p2p tools. In order to run this script

find and configure 'NOF_CHANNELS' variable in the script to required number of such pairs. The output will be sent to the IP configured in REMOTE_IP variable in the script

Note: this test results can be heavily affected by the file system and storage hardware that is used to store the played files. (For example tmpfs is the fastest, SSD ext4 is slower and HD ext4 is the slowest). See "File playing density testing" section.

4.2.7 voice_transcoding_load.py sample script

General HMP tool scheme:



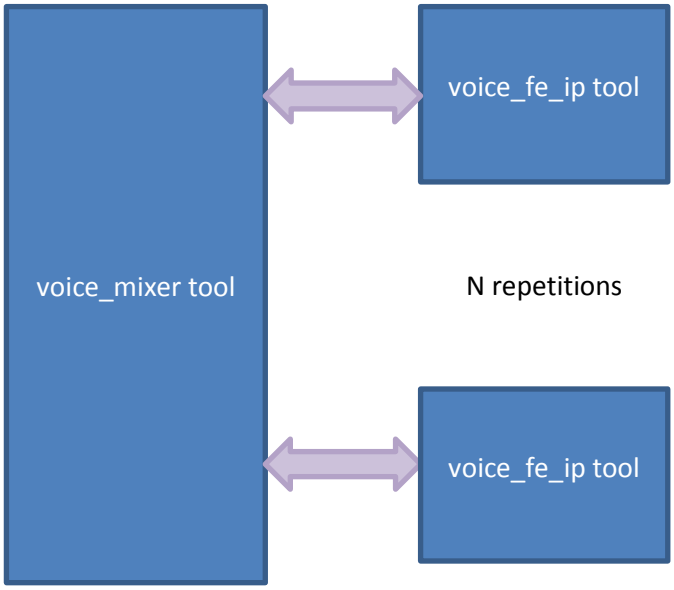
This is generic script for load testing of voice_p2p trans-coding tools. It configures N voice trans-coder tools from codec A to codec B and N tools from codec B to codec A. In order to run this script, several parameters must be configured:

- open the script in text editor
- at the very beginning of the script update NOF_CHANNELS with number of full-duplex trans-coders that need to be tested
- update REMOTE_IP value with destination IP of the machine that will receive all RTP traffic generated by the MediaProcessor
- update CODEC_A value with required first codec
- update CODEC_B value with required second codec

Then use A_B_load_rtp.sh script from capture directory of the package where A is the first codec and B is the second codec. Note that rtptools package needs to be installed

4.2.8 voice_mixer_load.py sample script

General HMP tool scheme:



This is generic script for load testing of voice conferences. It creates configurable number of voice conferences with configurable number of participants. In order to configure those parameters, open the script in a text editor:

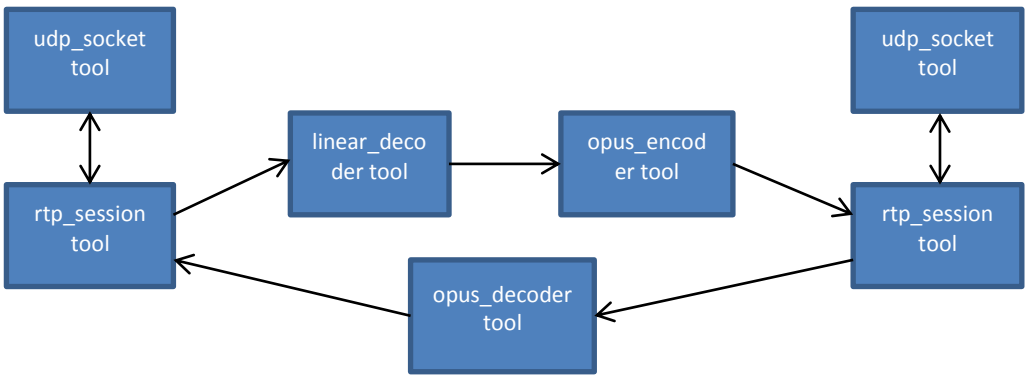
- NOF_CONFERENCES can be changed to required number of conferences for testing
- NOF_PARTICIPANTS can be changed to required total number of participants in each conference
- NOF_ACTIVE_PARTICIPANTS can be changed to required number of participants in each conference that actually can talk in the conference ("regular" type according to API). Other participants will be defined as listeners only. Note that high number of active participants can reduce performance/density.

Codec can get the following values: "G.711alaw","G.711ulaw","G.729","AMR_NB","AMR_WB"

After the mixer_load.py script is running and all the tools are configured, use CODEC_mixer_load.sh script where CODEC is the participants codec type to simulate tools RTP input. The output will be sent to the IP configured in REMOTE_IP variable in the script

4.2.9 opus_linear_transcoding_load.py sample script

General HMP tool scheme:



This script allows testing of NOF_CHANNELS instances of the above tool configuration. Each channel represents full duplex trans-coder between linear and opus clients.

The following parameters can be configured in the script:

- NOF_CHANNELS – number of full duplex trans-coders
- LINEAR_SIDE_REMOTE_IP – IP address where the linear output will be sent to
- OPUS_SIDE_REMOTE_IP – IP address where the opus output stream will be sent to
- OPUS_ENCODER_SAMPLING_RATE – sampling rate that will be used by the encoder, if the incoming sampling rate differs, it will be converted automatically
- OPUS_ENCODER_BIT_RATE – bit rate that will be used for opus encoding

After the script is running and all the tools are configured use opus_linear_transcoding_load.sh from the Captures directory to load the configured tools with input.

4.3 File play density testing

In order to make file play work, media files need to be present and available via file system access on the machine that runs the HMP (MediaProcessor).

Type of the file system and the hardware affect the performance/density that can be achieved. The best way to achieve highest performance is to use tmpfs (file system located in local RAM) and read the files from it. tmpfs mount point can be created in the following way:

mkdir mnt_tmpfs

mount –t tmpfs –o size=1024m tmpfs mnt_tmpfs

Then mnt_tmpfs directory can be used to store the media files which will be read by the MediaProcessor (HMP)

If tmpfs option is not available, SSD can be used to improve the performance comparing to regular mechanical hard drive.

Original file play testing scripts suppose that the files are read from input_files directory under MediaProcess's folder. This can be easily changed by modifying this path inside the script in order to use or tmpfs mount point or SSD disk mount point.

4.4 Loading RTP for load tests

For the purpose of loading the SURF HMP solution with RTP traffic, the "rtptools" is used. It can be downloaded from:

http://www.cs.columbia.edu/irt/software/rtptools/

Follow this link for download and installation instructions of "rtptools"

Within the rtptools, only the rtpplay utility is used. The rtpplay utility is capable of streaming RTP stream saved in ".rtp" format that is supported also by wireshark: wireshark is able to export a captured stream to .rtp format.

http://www.wireshark.org/

In SURF HMP package the user can find .rtp files used for testing.

To run rtpplay, the following command should be used:

rtpplay –T –f [.rtp filename] [remote IP address]/[remote UDP port]

In some test scenarios this command is run from shell script

To simulate real scenarios, a separate machine must be used as a "loader" – machine that streams all RTP streams into the Media Processor. This will also help not to use the "tested" machine resources for sending RTP. In load tests the "loader" machine must be configured to have enough operating system file descriptors in order to provide required number of RTP streams. This machine will be also used to run the sample applications.

In order to check the results, the wireshark utility should be used on a separate machine (validator). It captures the resulting streams and check its content. Wireshark has the capability of decoding and playing G.711 streams as well as calculation of RTP jitter and other parameters that are used for validation.

In order to achieve high throughput, at least 1Gb Ethernet switches must be used.