Pi-to-Pi Conferencing Overview

Video

On the source computer, the camera is accessed by the `camera` program using the Video4Linux2 library (driver: `bcm2835_v4l2`). `camera` compresses each frame using the `turbojpeg` library and then sends it to the target IP in tcp packets.

On the display computer, the `viewer` program receives the data, decompresses it using the `turbojpeg` library, and displays it in the video framebuffer using the Video4Linux2 library.

Audio

The `talk` program uses the `alsa` sound system library to receive the microphone data, the `opus` library to compress it, and the tcp to send it over the network. I originally tried to use the `ortp` library which sends the sound data as `utp` packets, but found that `ssh` tunnels only can forward `tcp` packets. Simply skipping the `rtp` protocol and using `tcp` appears to work well. Each frame of compressed sound data is preceded by a single byte containing its length. The frames are typically 50 to 100 bytes in length so a single byte seems sufficient.

It is then played by the `listen` program which receives the `tcp` packets, uncompresses it with the `opus` library, and plays it using the `alsa` sound system library. The player has a buffer that collects a few kB of sound data before playing it so that irregularities in the Internet delivery can be ironed out.

Getting Through the Network and Firewalls

The network connections are illustrated in the diagram on the following page. At the left side is a Raspberry Pi called `raspberrypi5` (address 10.0.0.2) on the server `64.118.100.243` running our local network. On the right side is a Raspberry Pi called `raspberrypi` (address 192.168.1.13) on a local network connected to an Internet Service Provider server with address `172.72.143.30`. The server at `64.118.100.243` has a firewall that can be modified us, but the server at `172.72.143.30` cannot. Therefore, a user on `raspberrypi` can ssh to `raspberrypi5`, but the reverse is not normally possible.

A user on raspberrypi can, however, use ssh to create a reverse tunnel using

```
ssh -R 2345:localhost:22 pi@64.118.100.243
```

that connects the `ssh` port (port `22`) of `raspberrypi` to a local port (port 2345) on `raspberrypi5`. This requires an adjustment (shown below) to the firewall on `64.118.100.243`. With that tunnel, a user on `raspberrypi5` can `ssh` to his local port 2345 using

```
ssh -p 2345 pi@localhost
```

and acquire a command shell on `raspberrypi`.

To run the video conference, a user on `raspberrypi` starts both camera-viewer pairs using

```
cd ~/Programming/Pi2Pi
./VideoConf.py
```

Passwords or `ssh` key handshakes are required. The script can be edited to select different sizes and frame rates for the video. The user at `raspberrypi5` only needs have his computer on and his firewall set to forward from her dynamic IP address. That address can be found by using

```
netstat -t -n
```

and seeing which address is connected to port 2345.

The audio communication is set up by the initiating user doing

```
./AudioConf.py
```

Both video and audio scripts are stopped by doing a typing a Ctrl-c which causes the scripts to shutdown cleanly after a few seconds and writing a stats file with the average transfer information for the session.

The firewall on 64.118.100.243 needs to have the following `iptables` entries in its `nat` (network address translation) table in order to permit a user on the masqueraded network with address `172.72.143.30` to reach `10.0.0.2` on the local network of 64.118.100.243. The audio is uses port 1350 and the video uses port 5010.

```
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       tcp  --  172.72.143.30        0.0.0.0/0            tcp dpt:22 to:10.0.0.2
DNAT       tcp  --  172.72.143.30        0.0.0.0/0            tcp dpt:1350 to:10.0.0.2
DNAT       tcp  --  172.72.143.30        0.0.0.0/0            tcp dpt:1360 to:10.0.0.2
DNAT       tcp  --  172.72.143.30        0.0.0.0/0            tcp dpt:5000 to:10.0.0.2
DNAT       tcp  --  172.72.143.30        0.0.0.0/0            tcp dpt:5010 to:10.0.0.2

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  10.0.0.2             0.0.0.0/0
```

Video Sources

The sources for the video camera and viewer programs are in the subdirectory and `Video`. That directory hs a suitable `Makefile` for compilation of those programs.

The `turbojpeg` library is needed: `sudo apt-get install libturbojpeg-dev`

The module `bcm2835_v4l2` and related modules are usually already available in Raspbian. Its APIs are used for accessing the camera. The framebuffer APIs are used for the viewer which is why the video output is not under control of the X-window system.

Audio Sources

The audio sources are in the directory `Audio` and have been considerably altered from the original tx and rx programs provided by Mark Hills under the name trx. The talk program and listen program both use a `defaults.h` file that sets some command-line defaults. I have set those for what seems to work adequately with my USB microphone (GoMic) and for using the built-in sound driver. I have tentatively concluded that using the USB microphone's output jack for driving speakers introduces clicking into the audio. Using the audio sent out the HDMI connector should be best.

The alterations allow from writing the audio to one file or a set of individual files and using the tcp protocol instead of ortp. Also, on the listen end, a buffer is used so that several seconds of audio can be acquired before it is played. The parameters controlling the buffering in `listener.c` are

```
#define BUFFER_SIZE 8192*16
#define BUFFER_THRESHOLD_SIZE 2048
#define BUFFER_REWIND_TARGET 1024*16
```

Controlling Programs

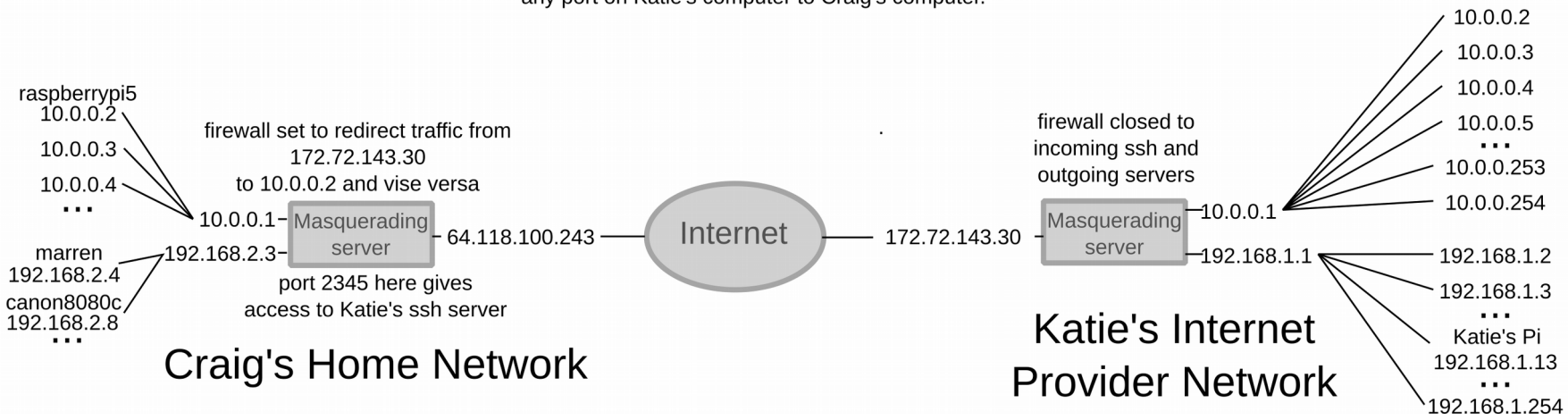
The `VideoConf.py` and `AudioConf.py` programs run the video and audio respectively. They set the parameters for the camera, viewer, talk, and listen programs, set up ssh tunnels, set volumes, and help them gracefully close down when a <Ctrl-c> is received. The camera and talk programs write summaries in `Stats-Video-Camera`, `Stats-Video-Viewer`, and `Stats-Audio` files when shutdown. If a debug parameter is set to non-zero values of 1 or 2 in their header files, diagnostic output is placed in files `diagnostic_camera`, `diagnostic_viewer`, `diagnostic_talk`, and `diagnostic_listen`. The camera program is currently set to automatically shutdown after 3 hours.

A Tkinter Python GUI program called `ConfGUI` can be used to run the video and audio communication from touch screen or mouse activation. It is placed in ~/Desktop and then double-clicked to start it.

Using SSH Tunneling with its -R Option to Access Hidden Servers

Katie on a Raspberry Pi with address 192.168.1.13 on the Stratton house network (shared address 172.72.143.30) sets up an ssh tunnel to Craig's raspberrypi5 with address 10.0.0.2 on Craig's network (static address 64.118.100.243, yosemitefoothills.com)

The command "ssh -R 2345:localhost:22 pi@64.118.100.243" when run by Katie on computer 192.168.1.13 of her Internet provider network allows Craig as user pi on 10.0.0.2 of the network served by 64.118.100.243 to access Katie's computer. Katie must know Craig's password and Craig must know Katie's password. After Katie has set up the connection, Craig uses the command "ssh -p 2345 pi@localhost" to obtain access to Katie's ssh server port 22 and anything on Katie's computer. Both Craig and Katie must have ssh servers running on their computers, but Katie's server cannot be accessed from the Internet because it is blocked by her Internet provider. 64.118.100.243 has a "static IP address" which can be directly accessed from the Internet and therefore by Katie's ssh client program. Also, server 64.118.100.243 must have its firewall forward port 2345 to 10.0.0.2. An encrypted "ssh tunnel" is created by Katie between the computers that can, in general, channel any port on Katie's computer to Craig's computer.



Pi – to – Pi Video Conferencing

CPU Hardware: Raspberry Pi 2 (or 3) from Raspberry Pi Foundation using ARM hardware architecture
https://en.wikipedia.org/wiki/Raspberry_Pi_Foundation
https://www.raspberrypi.org/

Operating System: Raspbian-Jessie, a version of Debian Linux
https://www.raspbian.org/

Microphone: USB GoMic from Samson (Raspberry Pi 3 would use a Bluetooth microphone)
http://www.samsontech.com/samson/products/microphones/usb-microphones/gomic/

Audio System: ALSA, Advanced Linux Sound Architecture
https://en.wikipedia.org/wiki/Advanced_Linux_Sound_Architecture
http://www.alsa-project.org/main/index.php/Main_Page

Audio Compression: Opus
https://en.wikipedia.org/wiki/Opus_%28audio_format%29
https://www.opus-codec.org/

Audio Networking: RTP – Real-Time Transport Protocol (Not used since cannot go through ssh tunnels.)
https://en.wikipedia.org/wiki/Real-time_Transport_Protocol
http://www.linphone.org/technical-corner/ortp/overview

Camera: Raspberry Pi Camera
https://www.raspberrypi.org/products/camera-module/

Video Compression: Turbo JPEG
http://www.libjpeg-turbo.org/Main/HomePage

Camera Driver: V4L2 – Video for Linux 2
https://en.wikipedia.org/wiki/Video4Linux

Normal Display: Either Raspberry Pi Display (800x480 with touch screen) or HDMI
https://www.raspberrypi.org/products/raspberry-pi-touch-display/

Video Overlay: Linux Frame Buffer
https://en.wikipedia.org/wiki/Linux_framebuffer
http://tldp.org/HOWTO/Framebuffer-HOWTO/
http://elinux.org/RPi_Framebuffer

Networking: IP/TCP – Internet Protocol/Transport Control Protocol
https://en.wikipedia.org/wiki/Transmission_Control_Protocol
http://www.linuxhowtos.org/C_C++/socket.htm

Coordination: Python programs using `subprocess` library
https://docs.python.org/3/library/subprocess.html
Tkinter provides a GUI window that can run the python command-line programs.
https://docs.python.org/3.5/library/tk.html and https://www.tcl.tk/man/tcl8.5/
An outstanding set of Tkinter examples by Stephen Ferg can be found at
http://www.ferg.org/thinking_in_tkinter/all_programs.html
(For Python 3, they need to be edited to use parentheses in `print` statements, `tkinter` for `Tkinter`, and `input()` for `raw_input()`.

Memory Leak Testing: valgrind is a useful program. Here are examples of its use:
`sudo apt-get install valgrind`
valgrind --leak-check=yes ./camera 592 444 5 3600 127.0.0.1 5010
valgrind --leak-check=yes ./viewer 10.0.0.6 5010 592 444 208 36 0 8