Introduction to Streaming Video

This tutorial explores a couple of ways of viewing continuous video sent from the RaspberryPi camera to laptops on a local area network. This shows the general situation from start to finish:

```
Scene of interest ->
                ---->  PiCam (raspivid)
                ---->  RasPi nanocomputer
                ---->  NETWORK (could be local or internet)
                ---->  latency of some sort
                ---->  you@YOUR COMPUTER
                ---->  DISPLAY (vdu, projector)
```

Steep Learning Curve ahead

There are various ways of achieving this. Two methods are investigated: using `netcat` and using a specialised `gstreamer` program. We look at ways of reducing latency, and of displaying video on other computers in a Local Area Network. But first, a bit about `netcat` and `ports` ...

Ports

TCP/IP packets arriving at a network interface card contain information about source IP address, destination IP address, port, service and data. To use `netcat` we need to check the usual suspects and choose port numbers (up to 65535) that are unlikely to conflict with any current usage.

```
$ less /etc/services
# Network services, Internet style
tcpmux          1/tcp          # TCP port service multiplexer
echo            7/tcp
discard         9/tcp
systat          11/tcp
daytime         13/tcp
netstat         15/tcp
qotd            17/tcp
msp             18/tcp         # message send protocol
chargen         19/tcp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp          # SSH Remote Login Protocol
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
...
...
tfido           60177/tcp       # fidonet EMSI over telnet
fido            60179/tcp       # fidonet EMSI over TCP
```

For convenience in seeing what is going on, we shall always choose `port=12345` on our local machine, and if a further port is needed we shall use `port=54321`. To make it stand out in the examples, we shall use `11` rather than just `1` when referring to the `local` laptop, and `22` rather than just `2` when referring to the `raspberry pi`, and `33` rather than just `3` when referring to the `other` laptop.

What is this netcat thing?

```
$ man nc
NAME nc - TCP/IP swiss army knife
DESCRIPTION
       netcat is a simple unix utility which reads and writes data across net
       work connections, using TCP or UDP protocol. It is designed to be a
       reliable "back-end" tool that can be used directly or easily driven by
       other programs and scripts. At the same time, it is a feature-rich
       network debugging and exploration tool, since it can create almost any
       kind of connection you would need and has several interesting built-in
       capabilities. Netcat, or "nc" as the actual program is named, should
       have been supplied long ago as another one of those cryptic but stan
       dard Unix tools.

       In the simplest usage, "nc host port" creates a TCP connection to the
       given port on the given target host. Your standard input is then sent
       to the host, and anything that comes back across the connection is sent
       to your standard output. This continues indefinitely, until the net
       work side of the connection shuts down. Note that this behavior is
       different from most other applications which shut everything down and
       exit after an end-of-file on the standard input.

       Netcat can also function as a server, by listening for inbound connec
       tions on arbitrary ports and then doing the same reading and writing.
       With minor limitations, netcat doesn't really care if it runs in
       "client" or "server" mode -- it still shovels data back and forth until
       there isn't any more left. In either mode, shutdown can be forced after
       a configurable time of inactivity on the network side.
```

OK, so `nc` rules. So let's get familiar with it — we are going to use it a bit.

Getting comfortable with netcat

In what follows, we'll perform some heuristic exercises on three computers connected in our LAN:

local: which is our laptop with `eth0=192.168.0.11` and `wlan0=10.0.0.11`;

rpi: which is a raspberry pi with `eth0=192.168.0.22` and a `camera module`;

other: which is another wireless-only laptop with `wlan0=10.0.0.33`.

Disconnect from the internet. Over the internet you'll have to configure a firewall to let `nc` through, but we cannot deal with internet or firewalls here. While we work on our LAN, you might need to disable any firewall that could stop `nc` connecting. Just do this: `sudo /sbin/iptables -F`.

Always sketch the data flow.

Our network consists of a local computer (with you logged in) that has both a wired network interface `eth0` and a wireless network interface `wlan0`, connected by the `eth0` interface to the RaspberryPi, and there is some other computer with only a wireless interface `wlan0`.

Establish IP addresses

Your local computer is assigned the `eth0` IP address `192.168.0.11`, the RaspberryPi gets the `eth0` IP address `192.168.0.22` and the other computer has the `wlan0` IP address `10.0.0.33`.

```
local~$ sudo /sbin/ifconfig eth0 192.168.0.11 up
rpi~$ sudo /sbin/ifconfig eth0 192.168.0.22 up
local~$ [create a new ad-hoc network using your network manager with wlan0=10.0.0.11]
local~$ /sbin/route -n
Kernel IP routing table
Destination  Gateway    Genmask        Flags Metric Ref   Use Iface
10.0.0.0     0.0.0.0    255.255.255.0  U     0      0       0 wlan0
192.168.0.0  0.0.0.0    255.255.255.0  U     0      0       0 eth0
other~$ [connect to the ad-hoc network created above on local, and assign wlan0=10.0.0.33]
```

Let's try out some commands. Notice that we always start `nc` on the *listening* computer first.

First, some idle chat between two computers. After these commands are run, whatever is typed on either appears on the other until one process is cancelled.

```
other~$ nc -l -p 12345
local~$ nc 10.0.0.33 12345
hello world!
```

Second, a file transfer from `local` to `other`.

```
other~$ nc -l -p 12345 > file.out
local~$ nc 10.0.0.33 12345 < file.in
```

Third, play a movie that is stored on `local` but plays on the `other` computer.

```
other~$ nc -l -p 12345 | /usr/bin/mpv -
local~$ nc 10.0.0.33 12345 < movie.mp4
```

Finally, run a shell on `other` from `local`.

```
other~$ ls
f1 f2 f3
other~$ nc -l -p 12345 -e /bin/bash
local~$ nc 10.0.0.33 12345
ls
f1 f2 f3
```

Putting it all Together

We are now in a position to stream the RPi Camera over a netcat connection. There are various `youtube` videos on all aspects of this topic, and these recipes are tested and the results follow.

First, from one terminal emulator on `local`, start listening for video to display:

```
local~$ nc -l -p 12345 | /usr/bin/mpv -
Playing: -
[file] Reading from stdin...
```

Then, from another terminal emulator on `local`, log in to the `rpi` and start the camera:

```
local~$ ssh pi@192.168.0.22
rpi~$ raspivid -vf -hf -n -w 1024 -h 768 -t 0 -fps 20 -o - \
         | nc 192.168.0.11 12345
```

This plays on `local` with a latency of between 5-8 seconds.

Now for lower latency

From blogs, it appears that one simple trick is to read more frames (e.g. `fps=90`) on the listening side than you send out (e.g. `fps=20`) from the `Pi`; this makes sure the buffer stays empty.

First, from one terminal emulator on `local`, start listening for a video stream to display:

```
local~$ nc -l -p 12345 | /usr/bin/mpv -fps 90 -
Playing: -
[file] Reading from stdin...
```

Then, from another terminal emulator on `local`, log in and start the camera:

```
local~$ ssh pi@192.168.0.22
rpi~$ raspivid -vf -hf -w 1024 -h 768 -t 0 -fps 20 -o -|nc 192.168.0.11 12345
```

This plays on `local` with a latency of about 250 milliseconds.

Display on another laptop

First, from a terminal emulator on `other`, start listening for video to display:

```
other~$ nc -l -p 54321 | /usr/bin/mpv -fps 90 -
Playing: -
[file] Reading from stdin...
```

Second, from a terminal emulator on `local`, pass the stream on to `other` directly:

```
local~$ nc -l -p 12345 | nc 10.0.0.33 54321
```

Finally, from another terminal emulator on `local`, log in to the `rpi` and start the video:

```
local~$ ssh pi@192.168.0.22
rpi~$ raspivid -vf -hf -w 1024 -h 768 -t 0 -fps 20 -o -|nc 192.168.0.11 12345
```

This video stream goes from the `rpi` to `local` and is immediately passed on to `other` and plays on `other` with a latency of less than half a second.

Doing all this with gstreamer

This is part of the `https://en.wikipedia.org/wiki/GStreamer` project. You need to compile the latest experimental code on the `rpi` like this:

```
rpi~$ sudo apt-get install libv4l libv4l-dev libjpeg8-dev \
                   imagemagick build-essential cmake subversion
rpi~$ cd /usr/src
rpi~$ sudo mkdir mjpg-streamer
rpi~$ sudo chown $(whoami):users mjpg-streamer
rpi~$ cd mjpg-streamer/
rpi~$ git clone https://github.com/jacksonliam/mjpg-streamer .
rpi~$ cd mjpg-streamer-experimental/
rpi~$ make
rpi~$ export LD_LIBRARY_PATH=.
rpi~$ ./mjpg_streamer -o "output_http.so -w ./www" -i "input_raspicam.so \
                         -x 640 -y 480 -vf -hf -fps 20 -ex night"
```

To display it on `local`, open a terminal emulator on `local` and execute this: `./rpi-streamer.py` which runs the following python program:

```
#!/usr/bin/python
import cv2
import urllib
import numpy as np
stream=urllib.urlopen('http://192.168.0.22:8080/?action=stream')
bytes=''
while True:
    bytes+=stream.read(1024)
    a = bytes.find('\xff\xd8')
    b = bytes.find('\xff\xd9')
    if a!=-1 and b!=-1:
        jpg = bytes[a:b+2]
        bytes= bytes[b+2:]
        i = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8),cv2.CV_LOAD_IMAGE_COLOR)
        cv2.imshow('i',i)
        if cv2.waitKey(1) == 27:
            exit(0)
```

The video runs until you hit the ESC key when the cursor is in the frame. Nicely, you can also view it in a browser, by browsing to `http://192.168.0.22:8080` and clicking 'stream'.

Conclusion

We have examined two methods of viewing a continuous video taken by the RaspberryPi camera. The methods are tried on a secure local network, not over the internet, in order to illustrate the principles and practicalities involved. One method, using `netcat`, can send that video stream to any other computer on the network; the other method, using `gstreamer`, can view it on the computer connected to the `Pi`. Both methods can achieve very low latency of about a quarter of a second.