

Test/Demo Setup for GSE with the Ayecka SR1 Pro and GNU Radio Companion

Paul Williamson at GNU Radio Conference 2018-09-21

As of this writing, we have DVB-S2 transmit capability in GNU Radio but we don't have DVB-S2 receive capability. In order to create a test setup that uses the DVB-S2 waveforms, we need to be able to receive as well as transmit. As an interim measure, and as a third-party reference implementation, we use the commercial SR1 Pro demodulator from Ayecka.

<http://www.ayecka.com/products-SR1.php>

The SR1 comes in several flavors. The Generic Stream Encapsulation (GSE) and Adaptive Coding and Modulation (ACM) features of DVB-S2 are only available on the SR1 Pro with particular firmware. Ayecka offers a Java application to monitor and control the SR1, but we did not try it.

Our SR1 identifies itself as follows:

```
SR1g Serial No. 105891 30AAC
SW Ver 6.00b75 V HW Ver 2.00 FW Ver 9.01b023 V BL Ver 1.01b14
```

Hooking Up the SR1

Checklist:

- Power (12VDC 2A)
- RF input
- USB configuration/control port
- Gigabit Ethernet Traffic port

Hook up a 12V DC power supply capable of 2 amps. There should be an appropriate wall adapter with the SR1.

The SR1 has two RF input ports with F connectors. They can be used separately or together. For our demo we connected a small antenna to RF1 (and left RF2 unconnected). For controlled tests it would probably be better to cable directly to the transmitter through an appropriate attenuator.

The SR1 also has a mini-B USB port for configuration and control. This exposes a CP2102-based USB serial port at 115200 bps. Drivers for this chip are built-in for Linux. If you're using a Mac or Windows machine and you haven't used a serial port with this chip, you may need to install a driver. The USB port is only needed for configuration and monitoring. It doesn't have to be on the same computer as any other part of the test setup. Connect to it with a terminal emulator

program. On Linux or macOS, you can use screen. On Linux, the command will look something like this:

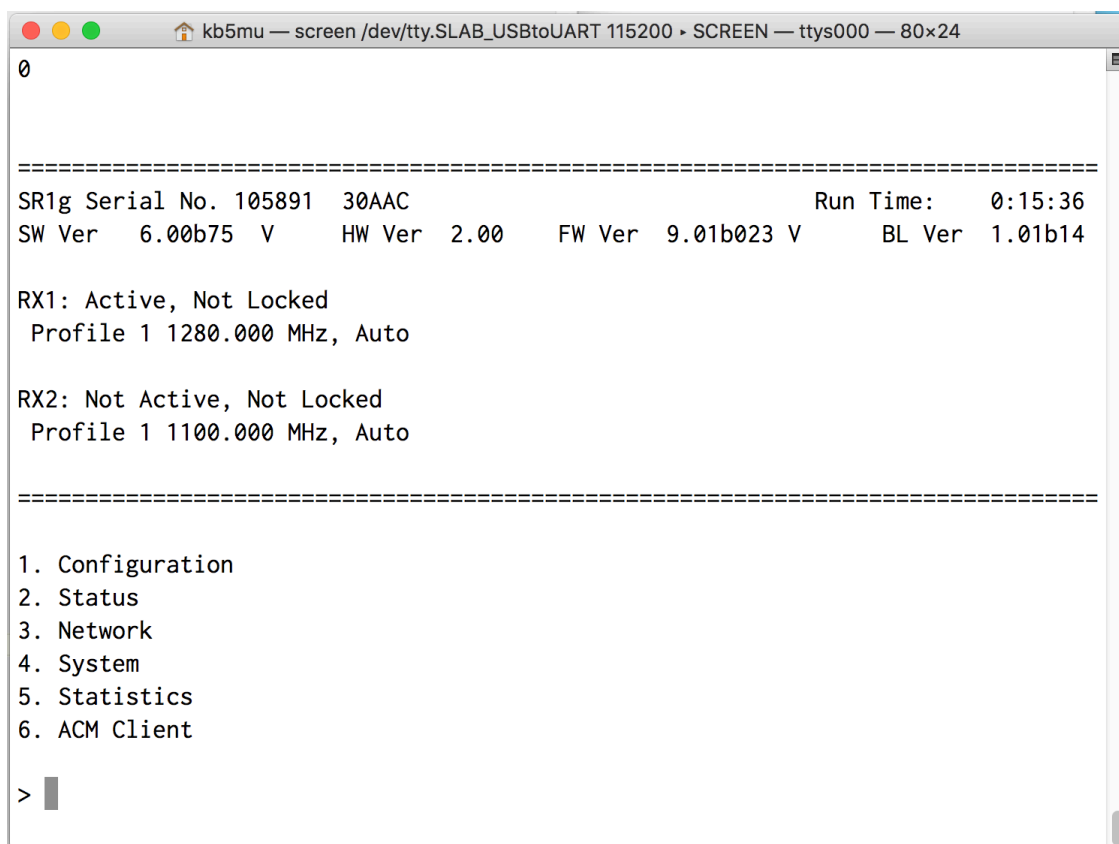
```
screen /dev/ttyUSB0 115200
```

while on macOS it will look more like this:

```
screen /dev/tty.SLAB_USBtoUART 115200
```

Windows users can probably use Hyperterminal or whatever terminal emulation program is handy.

After launching the terminal emulator you'll need to type 0 (zero) once or twice before the SR1 will send any text.

A screenshot of a macOS terminal window. The title bar shows 'kb5mu — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — ttys000 — 80x24'. The terminal content starts with a '0' on the first line. After a blank line, a separator line of equals signs appears. Below it, system information is displayed: 'SR1g Serial No. 105891 30AAC' and 'Run Time: 0:15:36'. The next line shows 'SW Ver 6.00b75 V HW Ver 2.00 FW Ver 9.01b023 V BL Ver 1.01b14'. This is followed by two sections: 'RX1: Active, Not Locked' with 'Profile 1 1280.000 MHz, Auto', and 'RX2: Not Active, Not Locked' with 'Profile 1 1100.000 MHz, Auto'. Another separator line of equals signs follows. At the bottom, a numbered menu is listed: '1. Configuration', '2. Status', '3. Network', '4. System', '5. Statistics', and '6. ACM Client'. The prompt '>' is at the bottom left with a cursor.

The SR1 menus respond to single keystrokes. Hit '0' to back up. Status screens generally auto-update every few seconds, scrolling up. It may be hard to tell that updates are happening if nothing is changing, depending on how your terminal emulator handles scrolling. Configuration screens may time out in a relatively short time if you don't make a selection promptly.

When the SR1 is demodulating a DVB-S2 signal with GSE-encapsulated IP packets, it will send those packets out via the Ethernet port marked Traffic. For this to work, a few settings have to be made. These are under Network (hit 3).

1. Management IP Address	192.168.2.55
2. Management IP Mask	0.0.0.0
3. Management Ethernet Address	CC-F6-7A-04-D8-E8
4. Management IP Multicast	Off
5. Management DSCP	0
6. Management VLAN ID	0
7. Management Default Gateway	192.168.2.1
8. Management DHCP Client	Off
9. Management Port State	Enabled
A. LAN IP Address	192.168.2.38
B. LAN IP Mask	255.255.255.0
C. LAN Ethernet Address	CC-F6-7A-04-D8-E7
D. LAN IP Multicast	Off
E. Router IP Address	192.168.2.1
F. LAN DHCP Client	Off
G. ARP Management	
H. Jumbo Frame Mode	1522
I. Air IP Address	192.168.1.161
J. Air Ethernet Address	CC-F6-7A-04-D8-E6
K. Isolate Networks	Isolated
>	

Connect the SR1 Ethernet port marked Traffic to an Ethernet port on the same Linux computer that's going to run the DVB-S2+GSE flowgraph. This can be a direct connection or through a router. We used a direct connection and configured the IP address of the Linux computer manually. If you use a router, it will probably assign an IP address to the Linux computer. Set the LAN IP Address (menu 3 A on the SR1) to an available address in the same subnet as the Linux computer. Set the Router IP Address (menu 3 E on the SR1) to the same address as the Linux computer. This enables the SR1 to route received packets through the Ethernet port to the Linux computer. These packets will have the source IP address and destination IP address as received over GSE, regardless of these settings.

Test the network configuration of the SR1 by pinging from the Linux computer to the LAN IP Address you set in menu 3 A.

```

abraxas3d@ghostkitti:~/prefix/default$ ifconfig enp3s0
enp3s0    Link encap:Ethernet  HWaddr 00:0b:0e:0f:00:ed
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:1025722931 errors:0 dropped:0 overruns:0 frame:0
          TX packets:512893019 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:807194628168 (807.1 GB)  TX bytes:29747880533 (29.7 GB)

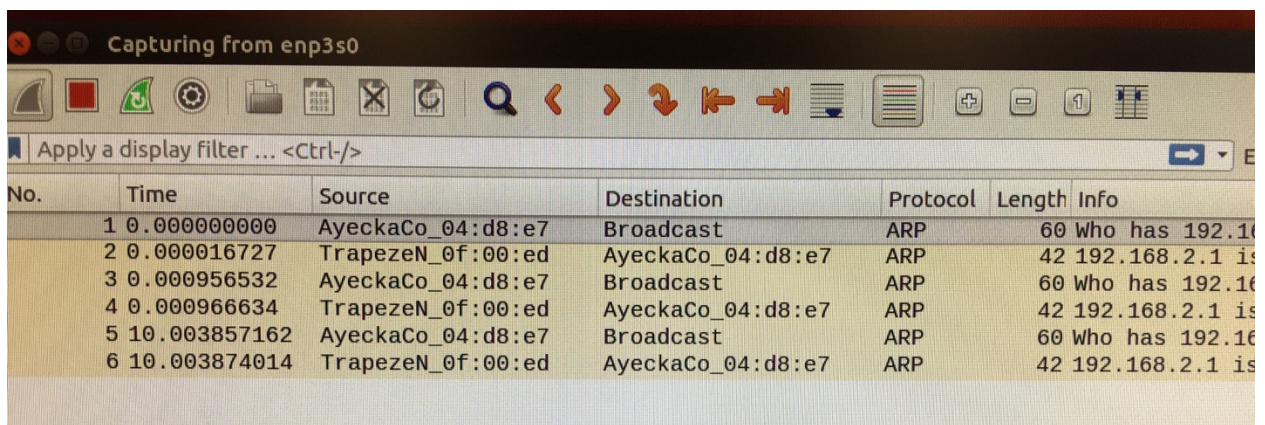
abraxas3d@ghostkitti:~/prefix/default$ sudo ifconfig enp3s0 192.168.2.1
[sudo] password for abraxas3d:
abraxas3d@ghostkitti:~/prefix/default$ ifconfig enp3s0
enp3s0    Link encap:Ethernet  HWaddr 00:0b:0e:0f:00:ed
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:1025722931 errors:0 dropped:0 overruns:0 frame:0
          TX packets:512893019 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:807194628168 (807.1 GB)  TX bytes:29747880533 (29.7 GB)

abraxas3d@ghostkitti:~/prefix/default$ ping 192.168.2.38
PING 192.168.2.38 (192.168.2.38) 56(84) bytes of data.
64 bytes from 192.168.2.38: icmp_seq=1 ttl=64 time=4.24 ms
64 bytes from 192.168.2.38: icmp_seq=2 ttl=64 time=4.58 ms
64 bytes from 192.168.2.38: icmp_seq=3 ttl=64 time=4.12 ms
64 bytes from 192.168.2.38: icmp_seq=4 ttl=64 time=4.78 ms
^C
--- 192.168.2.38 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 4.125/4.433/4.784/0.274 ms
abraxas3d@ghostkitti:~/prefix/default$

```

To see more of what's going on, you might want to run Wireshark and trace on this Ethernet port. If you do that right now, you should see some occasional network frames coming out of the SR1. When everything is set up and working, you'll also see the packets that have been carried over GSE.

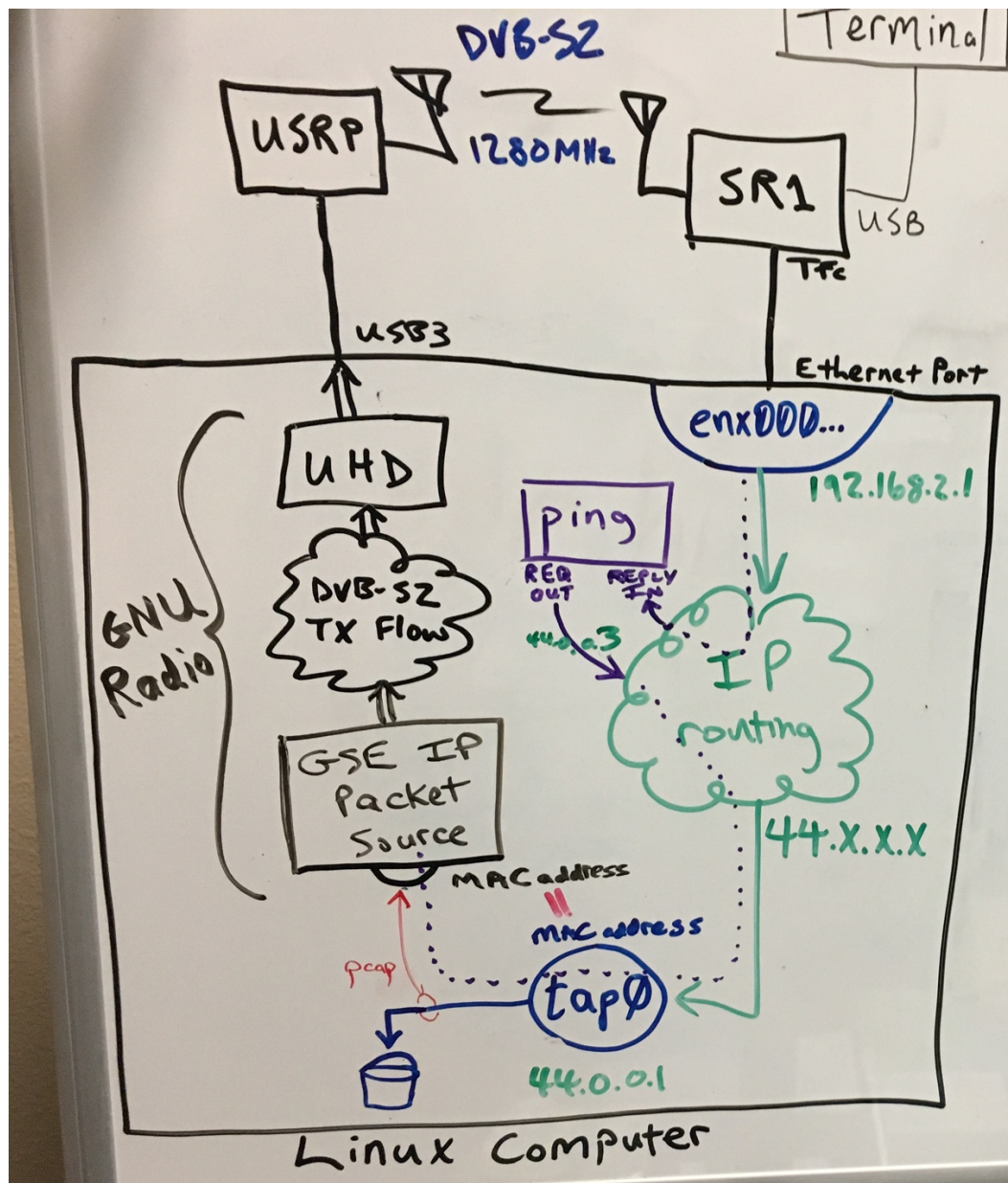
If you just installed Wireshark, you'll probably have to set up special permissions to let it capture packets. See <https://wiki.wireshark.org/CaptureSetup/CapturePrivileges> for details. If the procedure there doesn't seem to work, you may need to log out and back in.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	AyeckaCo_04:d8:e7	Broadcast	ARP	60	Who has 192.168.2.1 is at
2	0.000016727	TrapezeN_0f:00:ed	AyeckaCo_04:d8:e7	ARP	42	192.168.2.1 is at
3	0.000956532	AyeckaCo_04:d8:e7	Broadcast	ARP	60	Who has 192.168.2.1 is at
4	0.000966634	TrapezeN_0f:00:ed	AyeckaCo_04:d8:e7	ARP	42	192.168.2.1 is at
5	10.003857162	AyeckaCo_04:d8:e7	Broadcast	ARP	60	Who has 192.168.2.1 is at
6	10.003874014	TrapezeN_0f:00:ed	AyeckaCo_04:d8:e7	ARP	42	192.168.2.1 is at

The Network Setup

For this demo, we're going to use a somewhat tricky network setup so that we can run the entire demo on a single Linux computer with a single physical network adapter (Ethernet port). There is special support for this setup built into the gr-dvbgse package at this writing. Note that future versions of gr-dvbgse might change the data interface, in which case you'll need different procedures.



The trick here is the tap0 interface. This is a virtual interface created in software that looks to the network stack like an Ethernet interface. Using such a virtual interface, we can pretend to

be part of a network that we're not physically connected to. (Tap interfaces have the primary function of copying the packets to a character device for file-like access, but we aren't using that function here.) We create a tap0 interface and assign it to the IP address subnet we want to use over the air, in this case the AMPRnet network 44.x.x.x. Now packets addressed to network 44 will be routed to the tap0 interface, and programs can use network 44 addresses as source addresses. The tap0 interface also has an associated MAC address, just like the unique address a physical Ethernet interface would have from the factory.

The GSE IP Packet Source block from the gr-dvbgse package obtains its input data (packets to send over GSE) by using pcap to eavesdrop on packets on the tap0 device (tap0 is currently hard-coded) with an Ethernet source address equal to the tap0 device's MAC address (the MAC address is setttable in the block settings, and it must match). That way it gets exactly those packets that would have gone out on the tap0 interface, had it been an actual physical interface. Aside from this eavesdropping, packets from the tap0 interface don't go anywhere. They'll show up as dropped packets if you check the tap0 interface statistics.

For initial testing we use the program ping as both source and sink of data. Ping is convenient because it makes it easy to control the data rate and it automatically checks the received data. Ping works by sending out ICMP Echo Request packets to a specified IP address, and expects to receive ICMP Echo Reply packets in return. In a typical use case, there would be another host computer at that IP address to receive the requests and generate the responses. In our test setup, we don't have another computer. Instead, the GSE IP Packet Source block knows a trick. If the Ping Reply switch in the block is turned on, the block watches for ICMP Echo Request packets and modifies them to create ICMP Echo Reply packets, swapping the source and destination addresses. When these modified packets pass over the DVB-S2 link and come back in on the Ethernet interface from the SR1, they look just like typical responses to the ping program.

This takes some setting up.

First, we need to enable pcap to capture packets. This is not normally allowed, for security reasons, and Linux has tightened up on these restrictions lately. Running as root would be one possible method, but that's a really bad idea even in a test configuration. Instead, we can grant the appropriate networking capabilities to the executable program. Unfortunately, when running a block in a flowgraph in GNU Radio, the executable program is the Python interpreter. We will go ahead and grant networking capabilities to Python, but be aware that this opens a gaping security hole in your system. You should probably remove the capabilities from Python when you're done with the demo, and don't run the demo at all on critical systems.

To grant the capabilities:

```
sudo setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /usr/bin/python2.7
```

To rescind the capabilities:

```
sudo setcap -r /usr/bin/python2.7
```

To check the status of the capabilities:

```
getcap /usr/bin/python2.7
```

Unfortunately, this causes a side effect. For security reasons, programs that have enhanced capabilities do not honor the `LD_LIBRARY_PATH` environment variable that normally controls where the program will look for shared libraries to load. In particular, if you use PyBOMBS to install GNU Radio into a prefix directory, you rely on `LD_LIBRARY_PATH` pointing to `prefix/lib` to enable it to find the shared libraries that contain GNU Radio packages. By granting capabilities to the Python interpreter, we've defeated this mechanism. Arrgh! To work around this, we can effectively add our library directory to the system library search path. We will then have to modify this part of the configuration every time we change to a different GNU Radio installation (prefix directory). To implement this hack, create a file in `/etc/ld.so.conf.d` that contains a line naming the directory where our shared libraries are stored. I named my file `gnuradio-prefix.conf`, and it contains this single line:

```
/home/kb5mu/prefix/lib
```

You'll need to run

```
sudo ldconfig
```

for this to take effect. To undo this, just delete the file and rerun `ldconfig`. If you forget to undo this, you will cause yourself (or some future user of PyBOMBS on that computer) a lot of confusion.

Here's the recipe for setting up the `tap0` interface:

```
sudo ip tuntap add dev tap0 mode tap
sudo ip link set dev tap0 address 02:00:48:55:4c:4b
sudo ip addr add 44.0.0.1/24 broadcast 44.0.0.255 dev tap0
sudo ip link set tap0 up
sudo ip link set tap0 arp off
```

The MAC address in the second command is arbitrary but must match the setting in the GSE block. The interface must be named `tap0`.

One last networking trick. By default, Linux systems will refuse to accept incoming packets that come in on the “wrong” interface. That is, if a packet arrives on a certain port with a source address that is not reachable through that port, it's considered bogus and dropped. We have to defeat this, though, because we need to accept packets coming in from the SR1 to our physical

Ethernet port with source addresses on our over-the-air network. To turn this *reverse path filtering* off, here's the recipe:

```
sudo sysctl -w net.ipv4.conf.all.rp_filter=0
sudo sysctl -w net.ipv4.conf.eth0.rp_filter=0
```

In the above, you need to change `eth0` to the actual port name of your Ethernet port. This may be something nice like `eth0`, or something ugly like `enx000ec6ca0d5c` or `enp3s0`.

Note that these `ip` and `sysctl` commands do not "stick". If you reboot the system, you will need to do them again. You can also explicitly remove the `tap0` interface like this:

```
sudo ip tuntap del dev tap0 mode tap
```

Note that some of these networking commands may be specific to Debian-derived systems like the Ubuntu 16.04 we used. Other systems may require different syntax.

Setting Up gr-dvbgse

If you haven't already, install GNU Radio. I used PyBOMBS to install in a prefix directory `/home/kb5mu/prefix`. I won't go over those details in this document.

First, let's test that we can transmit DVB-S2, before we try to install and run the GSE support. Run `gnuradio-companion` and open the example flowgraph for DVB-S2 transmitting:

```
gnuradio-companion ~/prefix/share/gnuradio/examples/dtv/dvbs2_tx.grc
```

By default the flowgraph is set up to use the `osmocom` sink. If you have a USRP instead, disable the `osmosdr` sink block (or delete it if you don't even have `osmocom` installed) and enable the USRP Sink block. If you have some other SDR, you'll need to add its sink block and configure it to match the existing ones.

On the USRP X310, I also had to delete the `master_clock_rate` setting from the device address field in the USRP Sink block.

This flowgraph transmits video from a file in MPEG Transport Stream format, with appropriate settings for the modulation and coding in use. You'll need to obtain such a file and reconfigure the File Source block to point to your file. I downloaded a copy of the same 220-megabyte file the flowgraph uses by default from

<http://w6rz.net/adv16apsk910.ts>

Run the flowgraph and watch the LOCK LED next to the RF connector on the SR1. If all is well, it will turn green after a few seconds. If not, debug!

Go to the serial terminal emulator and look at RX Channel 1 status (hit 2 1). You should see that the Tuner Status and Demodulator Status are Locked. Check the Demodulator Link Margin. If it's not well above 0 dB, say maybe 10 dB, you might want to increase the tx_gain value in the GUI of the flowgraph to make things a bit more reliable. (This doesn't seem to work on the X310 but it does on the B210.)

OK, that's good enough for testing DVB-S2 by itself. Stop the flowgraph and note that the SR1 immediately loses lock. Now let's install the GSE support.

If you're using PyBOMBS, you need to have the environment set up for your prefix. Open a fresh terminal and:

```
cd ~/prefix
source ./setup_env.sh
```

Make sure we have the prerequisite:

```
sudo apt-get install libpcap-dev
```

Clone the gr-dvbgse repository:

```
cd ~/prefix/src/gnuradio
git clone http://github.com/drmpeg/gr-dvbgse
```

Build the package:

```
cd gr-dvbgse
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=~/prefix ../
make
```

At this point I see compile errors, starting with "error: 'O_RDWR' was not declared in this scope". I don't understand why this error doesn't occur for the developer. The fix is to add this line:

```
#include <fcntl.h>
```

near the top of ~/prefix/src/gnuradio/gr-dvbgse/lib/bbheader_source_impl.cc and re-run make.

Once you have it making without errors, install the package. If you use PyBOMBS, be sure to do this from a terminal window with an environment configured for your prefix.

```
sudo make install
sudo ldconfig
```

And try to run the demo GSE transmit flowgraph:

```
gnuradio-companion ~/prefix/src/gnuradio/gr-dvbgse/apps/dvbs2_txip.grc
```

Once again, you'll need to adjust the sink block to match your SDR, just as before. Open the parameters of the GSE IP Packet Source block at the beginning of the flowgraph and make sure the MAC Address matches the one assigned to your tap0 interface and Ping Reply is On.

Run it.

If you get "RuntimeError:Error calling ioctl()", that probably means you haven't set up the tap0 interface properly since the last time you rebooted. Check the instructions above and try again.

If it runs successfully, you should see a nice spectrum and the lock light on the SR1 should again be green. Check the RX Channel 1 status again (hit 2 1 on the SR1) and verify lock and link margin. Hit 0 twice to get back to the main menu, and show RX1 Transport Statistics (hit 5 1). You'll see a count of BBFrames, GSFrames, Bad Frag CRC, and Bit Rate (bps). We haven't sent any data yet, so the last three are all zero.

Open a terminal on the Linux machine and try a nice slow ping:

```
ping 44.0.0.3
```

You should see the Bit Rate jump up to about 704 bps. You're moving data over GSE!

You should also see the ping replies coming in at a normal ping rate (about one per second). Hit Control-C to stop ping.

If you don't see any replies coming back, there's something wrong with your network trickery. In my case, I forgot to configure the Router IP Address to match the address of the physical Ethernet port. And in the end, I wasn't able to use the X310 because it required the one Ethernet port I had on the machine.

Now try opening the floodgates and pinging like crazy:

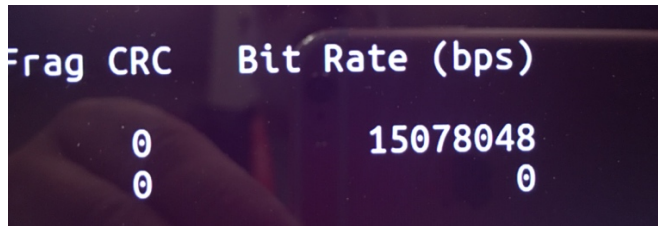
```
sudo ping -i 0.001 -s 10000 44.0.0.3
```

You need sudo here because ping limits its rate for non-privileged users. The -i value sets the interval between ping attempts (0.001 seems to be the smallest value that's honored). The -s value sets the number of bytes in each ping attempt.

You should see the data rate climb to about 13.9 Mbps.

Watch the latency reported by ping. If it starts to climb, then packets are backing up in a queue before being transmitted. Try adjusting the interval and packet size to eliminate the backlog.

You can change the modulation and coding used in the flowgraph, as long as you change it the same way in every block and choose a combination that exists in the MODCOD table in the DVB-S2 specification. The SR1 will automatically adapt to whatever valid MODCOD you send it. You can also vary the symbol rate over some range. The data rate you can achieve will depend on the MODCOD and symbol rate you're using. Faster bit rates will require a stronger signal. With the antenna-to-antenna setup I was using, I was unable to close the link at the highest MODCODs, even over a path of just a few inches.



Using ping as a data source and sink is very handy for basic testing, but not very useful or flashy as a demo. The README file for gr-dvbgse describes transporting video over GSE by using VLC as the source and, separately, the sink. We didn't have time to try that at the conference.

Using PCAP as a way to get data into the flowgraph is what drives the requirement for special privileges for this demonstration, and is perhaps not the best design for normal use. Other modules in GNU Radio take input data on a socket. I'd like to investigate switching to that type of interface for GSE. That will require a different test/demo setup, of course.