- MMDVM_Bridge
    - Overview
        - MMDVM_Bridge is used to provide an interconnection between digital networks. It does this by decomposing each network protocol and creating an agnostic version of the stream. This generalized form can then be re-constructed into a new network stream of a different type. MMDVM_Bridge works in conjunction with other bridging tools to allow for the connection of networks with differing audio formats and metadata components. Support for DMR, D-Star, Yaesu Fusion, P25 and NXDN networks are provided. In addition, bridging to analog networks (AllstarLink, Echolink) can be accomplished with little effort.
        - One of the main benefits of MMDVM_Bridge is that it allows the bridge to be administered by you. No need for a network provider to set up the interconnect, you are in full control. You can add or remove interconnects at any time depending on your needs. The system can be set up on a local server or deployed in the cloud and uses very little system resources.
        - We see two main use cases for this technology. Using it as a way to create interconnects between established talkgroups on different networks in order to allow a user to access the system with the mode of their choice. And, to allow for the creation of ad-hoc interconnects in times of emergency when multiple modes are being used.
        - Discussion for this project can be found on the DVSwitch message board, https://dvswitch.groups.io/g/main
        - Wow, does it wash my car too?

            Would you like hot wax with that?
    - Architecture
        - DVSwitch Partner
            - A Partner transforms a stream from one protocol to another
                - A stream is composed of
                    - Signaling (start/stop tx)
                    - Metadata (IDs, callsign, slow data)
                    - Audio
            - MMDVM_Bridge separates streams into two types; foreign and generic (or TLV).
                - On one side of the bridge we have the foreign network. The foreign network is a protocol specific network such as DMR, P25, YSF, D-Star or NXDN. Each of these foreign networks has a different encoding, error correction, metadata and signaling.
                - In order to create the generic side of the bridge, we use TLV. TLV (Tag, Length, Value) is a protocol agnostic version of the network stream. Foreign network streams

are converted into TLV so they can be further transformed into a second type of foreign network stream (a bridge).
                - Tag
                    - Tags are commands or states
                - Length
                    - Length in bytes of the payload
                - Value
                    - Tag dependant
        - Ports
            - All ports are UDP
            - Most modes are defined with a RX port, a TX port and a IP address. The RX port waits for incoming data and processes it. The TX port sends data to a second Partner instance at the IP address specified.
        - Payload
            - TLV types
                - Begin Transmission
                    - Source ID (24 bit)
                    - Destination ID (24 bit)
                    - Repeater ID (32 bit)
                    - Slot (8 bit)
                    - Color Code (8 bit)
                    - Extended metadata (callsign)
                - End Transmission
                    - Slot
                - Audio Data
                    - Slot
                    - AMBE - 72 bit : DMR, Yaesu Fusion Narrow, NXDN
                    - IMBE - 88 bit : P25, Yaesu Fusion Wide
                    - DSAMBE - 48 bit : D-Star
                    - Depending on mode, audio TLV may contain 1 or more AMBE / IMBE / DSAMBE frames.
- Installation
    - MMDVM_Bridge is part of a Linux suite of applications which is designed to run on a variety of hardware platforms. You can install these tools on single board computers, servers or cloud based systems. Because the suite has a significant number of components, installation packages have been created to help you get started. A set of “metapackages” have also been created that group common packages together so installation gets even easier for most situations.
    - Installing MMDVM_Bridge or any other DVSwitch package is easy provided that you follow the directions. You must have the proper

hardware and operating system version or the install will not be successful.

- Prerequisites
    - A rudimentary knowledge of Linux. This is needed to navigate to operating system including how to become root, change directories, edit configuration files and other basic system administration knowledge. We understand that not everybody is a coder or a sysadm. You do not have to be (but it **does** help). But without the basic understanding of how to use Linux from the command line, everything that follows will be much harder.
    - Hardware platforms
        - Raspberry Pi / SBC (Single Board Computer)
            - The code is very portable and runs on a variety of ARM processors. We have tested on about every SBC you can think of with good results. However, there is one component that is VERY processor specific, the md380-emulator. If you plan on running this component, you must be running on a supported processor with the proper underlying command structure. Pi2, Pi3, H3, H5 are good matches. Older RPI and possibly other ARM processors can run with qemu-arm even though they are already on an ARM processor.
        - X86 / AMD64
        - USB
            - DV3000U. If you are planning on using the DV3000U please make sure that you have a solid 5V power source for the USB dongle. Any voltage drops below 5V will surely cause problems. AMBETest4.py can help you determine if your USB port is supplying enough power for reliable operation.
    - Operating systems
        - Debian Stretch is the main supported platform. Other platforms can be built, but for pre-compiled installation packages, Stretch will be the platform of choice. We support x86 (32 bit) and x64/AMD64 for intel hardware and ARM64 for ARM processors.
    - Virtual Machines
        - Yes, provided that they are running a supported processor and have a proper version of Linux installed. If you are planning on also installing ASL, please make sure you

have access to the kernel header package before going forward.
        - Required disk storage
            - Datafiles
                - The data files (DMR IDs and NXDN IDs) are small and should never take up much space on the disk. They should not be a concern to the system admin.
            - Logs
                - Log files can grow forever if not managed. First, make sure you have the proper log level set for your stage of development and deployment. If you are in development of your bridge, a verbose log level will help you debug any issues you have with the bridge. But the size of the log will get very large as time goes on, so you probably do not want to keep the logLevel set to 1 for long. When you are ready to deploy your bridge, a log level of 2 is probably a good choice.
                - You should have a plan for log rotation and possibly keeping logs in a ram disk or other file system.
        - Required memory
    - Root privileges
        - Either sudo or root are required to administer the system. We do not recommend allowing ssh login as root, but rather using `sudo -s` to create a new shell with root privileges.
    - Packages
        - The available packages are:
            - dvswitch_base, Analog_Bridge, md380-emu, MMDVM_Bridge, P25Gateway, P25Parrot.
        - The metapackages are named **dvswitch** and **quantar**.
        - **dvswitch** contains
            - dvswitch_base, Analog_Bridge, md380-emu and MMDVM_Bridge
        - **quantar** contains
            - dvswitch_base, MMDVM_Bridge, Quantar_Bridge P25Gateway, P25Parrot
        - After installing the repository, you can install either **dvswitch** or **quantar** or both
        - Installation
            - To install the repository (as root):

```
cd /tmp
```

```
wget http://dvswitch.org/install-dvswitch-repo
chmod +x install-dvswitch-repo
./install-dvswitch-repo
apt-get update
apt-get upgrade
apt-get install dvswitch quantar
```

- Getting updates
  - This is as simple as it could be (as root):

```
apt-get update
apt-get upgrade
```

  - Updates and existing configuration files
    Don't worry about an update overwriting a configuration file that you may have edited. One of the best parts of the package system is that it will only overwrite clean files. It will ask you if you want to backup your existing changes, so update early, update often!

- File locations
  - Programs
    In /opt/program_name
    /opt/MMDVM_Bridge/MMDVM_Bridge
  - Configuration files
    With the program
    /opt/MMDVM_Bridge/MMDVM_Bridge.ini
  - Datafiles
    In /var/lib/mmdvm or /var/lib/dvswitch
    /var/lib/mmdvm/DMRIds.dat
  - Logs
    In /var/log/mmdvm or /var/log/dvswitch
    /var/log/mmdvm/MMDVM_Bridge-2018-05-22.log
  - Scripts
    In /usr/local/sbin
    /usr/local/sbin/DMRIDUpdate.sh
  - Sample Configuration files
    /usr/share/doc/dvswitch
  - Daily cron tasks to update data files
    /etc/cron.daily/DMRIDUpdate (this is a symbolic link to the script In /usr/local/sbin Note: If you want to add a link, the link name should not have a extension (.sh)

- Optional components
  - Analog_Bridge

        - For bridging to an analog network or transcoding
    - DMRLink
        - For bridging to an IPSC DMR network
    - HBLink
        - For bridging to a HomeBrew network
    - Gateways : P25Gateway, NXDNGateway, ircDDBGateway
        - Some modes require a gateway to communicate with their network “reflector”.
    - Reflectors
        - If you are also setting up your own reflector, install the reflector code from github.
    - Quantar_Bridge
        - Used for including a Quantar repeater in your network.
    - AllstarLink
        - For linking to analog users on ASL or Echolink
        - For including users on mobile handsets
    - Dashboards
- Configuration files
    - *MMDVM_Bridge.ini*
        - Defines the network side of the stream for each supported mode
        - Split in a general section, mode specific and mode network sections
    - *DVSwitch.ini*
        - Used to define the “generic” side of the network
        - One stanza per mode: [MODE] where mode is DMR, DSTAR, etc
        - All modes begin with settings for
            - Address
            - RXPort
            - TXPort
        - Optional settings per mode
            - Slot
            - FallbackID
            - Translate
- File placement best practices
    - Either with the program or in /etc. However, you can place the configuration files anywhere you want and name them anything you desire. We often create multiple versions of the configuration files with different names depending on the need. For example, a version of Analog_Bridge.ini that is used to convert PCM to DMR may be called Analog_Bridge_DMR.ini and invoked with the command:

```
./Analog_Bridge ./Analog_Bridge_DMR.ini
```

        - Program is started with *program config.ini*
    - Building Bridges
        - Let's talk philosophy for a moment. We think that if you understand our mindset in how these components are intended to be used, you will likely be more successful in getting them to do what you want. We are widget makers. A widget serves a set of purposes, has a known interface and intended use cases.
        - For MMDVM_Bridge, the purpose is to convert network streams into a generic format so that it can further be reconstructed into a new format. Pretty simple. Network on one side and TLV on the other. The network side is a magic bag of bits representing a mode like DMR, NXDN, P25, etc. The goal of the TLV is to represent (transform) all of that information in a generic (or abstracted) data structure.
        - The interface to MMDVM_Bridge is the network. This is defined in MMDVM_Bridge.ini. On the mode side, the connection to the outside world depends upon the mode being used. Some modes connect to gateways, others to reflectors and still others to other bridges. Each mode is different and you have to have some knowledge of the network you are connecting to in order to configure this side of the bridge. Google and the links in this document can help you in this research, but it would be beyond the scope of this tome to try to detail all of the information you need to understand. On the plus side, we offer the most common scenarios as samples and for most people, this is a great starting place.
        - On the TLV side, we speak UDP. This is defined in DVSwitch.ini. A TLV packet is transmitted to a specific IP and port of the receiver. The receiver is the intended “consumer” of the TLV. It can be thought of as the goal of the transform. For example, if I wanted to bridge DMR to YSFn, I would connect the network side if the bridge to Brandmeister and the TLV would point at a YSFn partner. The goal is to take my input, DMR, and pass it on to a YSFn centric partner. So, I set my IP and TXPort to the YSFN listener (RXPort).
        - In the reverse direction, I would like to take any TLV I receive (consume) and send it to my network (DMR). So, I define a RXPort and anyone who wants to send me TLV that should go to my DMR network would use my RX port as the target. See?
        - Here is the greatest hint I can give you: **Draw a picture.** Label it with modes and directions and ip:ports. For example

```
BM < -- > MMDVM_Bridge < -- > MMDVM_Bridge < -- > YSFn
                127.0.0.1:xxxx -- >          RXPort: xxxx
                  RXPort: yyyy          < -- 127.0.0.1:yyyy
```

- See how the transmit port (TXPort in the ini file) on one side matches up with the receive port (RXPort in the ini file) on the other side? Once you have this picture, it is just a matter of going to the various ini files and setting the values to match your diagram.
- MMDVM_Bridge can be
    - Used to bridge like and unlike audio formats
    - Used in a single instance if desired
    - In general, every bridge can be thought of as a stream with an input and an output. The object of the bridge is to transform all of the data presented to the input to a form compatible with the output network. This transform can be simple and accomplished in a single step, or complex requiring several steps and components. The more alike the two networks are, the easier it is to perform the transform.
- Like audio types
    - Bridging like audio types means that the encoded audio is preserved without modification. It is not transcoded or modified in any way except for error correction. Audio quality between like networks is preserved with no loss. The job of these bridges is to extract the audio and metadata from the network packet (envelope) and present them in a generic way to a cooperating partner. That partner then adds a new envelope around the audio and metadata and sends the result to the new network.
    - There are 3 digital audio types AMBE, IMBE and DSAMBE. DMR, NXDN and Yaesu Fusion Narrow are all AMBE audio types (ambe3600x2450). P25 and Yaesu Fusion Wide are both IMBE (imbe7200x4400) audio types. And finally, D-Star is all by itself with DSAMBE (ambe3600x2400). The three AMBE networks and the two IMBE networks can be bridged together using only MMDVM_Bridge.
    - *MMDVM_Bridge.ini*
        - Enable both modes including any callsigns, IDs or settings
        - Enable network for mode
            - Make sure that ports and IP address properly point to the gateway or server
    - *DVSwitch.ini*
        - Make RX -> TX port and TX -> RX port
        - IP address should be loopback (127.0.0.1) for single instance
        - Set up fallback IDs and other metadata
- Un-alike audio types (transcoding)
    - Bridging un-alike audio types requires the same transforms as the “like audio” case above. However, in addition to the basic

transforms, the audio must also be converted from one format to another. This audio transform is accomplished by decoding the source audio frame into PCM (analog) and then re-encoding the PCM into the new audio format. This process is lossy and audio quality suffers in the transformation. You will see references to USRP in the documentation and the ini files. USRP is the packet format in which Analog_Bridge places the PCM audio data.

- Uses Analog_Bridge to transcode
  - 2 instance of AB which
    - Decode first mode into PCM
    - Encode PCM into second mode
  - AB PCM ports are crossed over
  - AB TLV ports point to respective ports in *DVSwitch.ini*
  - Setting audio levels when transcoding
- *DVSwitch.ini*
  - Each mode points to the ports in one of the AB instances
  - Set fallback IDs and other metadata
- This is where the diagram gets much more complex, but becomes even more valuable to getting the operation correct. Break down the job into chunks. We often recommend that you build a bridge from the outside in. What that means is to make sure you can connect to the foreign network first, with all login, metadata, call signs, etc functioning before proceeding. Do you see your data flowing at the edges? If so, then move inside one step.

```
DSTAR <-> MB1    <->       AB1      <->       AB2       <-> MB2 <-> BM
          I1:p1 -> {[rx:p1 i2:p2] -> [rx:p2 i3:p3]} -> rx:p3
          rx:p4 <- {[i4:p4 rx:p5] <- [i5:p5 rx:p6]} <- i6:p6
```

- So, here we want to verify we can connect to D-Star and Brandmeister first. D-Star connects to ircddbgateway, so make sure your MMDVM_Bridge.ini file is set up with all the right addresses, ports callsign and modules to connect to ircddbgateway. Same goes for your connection to BrandMeister.
- Next we see we have to connect MMDVM_Bridge1 to Analog_Bridge1 and MMDVM_Bridge2 to Analog_Bridge2. The ports we select for each of these must be unique. If you look at DVSwitch.ini you will see that we have already suggested a unique set of ports for each mode. So in this case we would see p1 = 32100 and p6 = 31100. P4 = 32103 and p3 = 31103. How about the two AB USRP ports? P2 = 34001 and p3 = 32001.
- So, 6 ports. Not including ircddbgateway or Brandmeister. MB1 to AB1, AB1 to AB2, AB2 to MB2 and back again. Yes, it is a lot to manage, but it you break it down and do it one step at a time, it makes sense. It is a stream, one transform at a time.

- Lets track through the pipeline:
    - DSTAR -> DMR
        - MB1 will transmit TLV (DSAMBE) packets to AB1 where the IP address is 127.0.0.1 and the TXPort in the [DSTAR] stanza of DVSwitch1.ini matches the RXPort value in the Analog_Bridge1.ini [AMBE_AUDIO] staza.
        - AB1 will transmit USRP packets to AB2 where the IP address is 127.0.0.1 and the TXPort in the [USRP] stanza of Analog_Bridge1.ini matches the RXPort of the [USRP] stanza of Analog_Bridge2.ini.
        - AB2 will transmit TLV (AMBE) packets to MB2 where the IP address is 127.0.0.1 and the TXPort in the [AMBE_AUDIO] stanza matches the RXPort in the DVSwitch2.ini [DMR] stanza.
    - DMR -> DSTAR
        - MB2 will transmit TLV (AMBE) packets to AB2 where the IP address is 127.0.0.1 and the TXPort in the [DMR] stanza of DVSwitch2.ini matches the RXPort value in the Analog_Bridge2.ini [AMBE_AUDIO] staza.
        - AB2 will transmit USRP packets to AB1 where the IP address is 127.0.0.1 and the TXPort in the [USRP] stanza of Analog_Bridge2.ini matches the RXPort of the [USRP] stanza of Analog_Bridge1.ini.
        - AB1 will transmit TLV (DSAMBE) packets to MB1 where the IP address is 127.0.0.1 and the TXPort in the [AMBE_AUDIO] stanza matches the RXPort in the DVSwitch1.ini [DMR] stanza.
- AMBE and DSAMBE encode/decode
    - You will need two instances of Analog_Bridge above. Each instance will require exclusive access to a vocoder either in hardware of software. The system may use a combination of hardware for best performance and cost. D-Star is the audio component you want to pay attention to here. It can be done with a DV3000U and OP25 but not with the emulator. The OP25 vocoder currently does not have the best audio quality and should probably not be used in a production bridge. So, you will end up using a hardware vocoder for D-Star. For DMR, your choices include the DV3000U, emulator and OP25. All are good and the quality is acceptable even from the OP25 vocoder.

        - Other bridges that need transcoding (YSFW, P25 for example) would use the same exact template as above, just changing the stanza names above to your mode of choice.
    - What can you not do?
        - You can not use a mode twice in the same instance
            - You can **not** bridge DMR to YSF **and** bridge YSF to NXDN in the same instance. This type of situation may arise when you are building clusters. You may want to cluster DMR, YSFn and NXDN. To do this, use multiple instances of MMDVM_Bridge and combine the modes using either a common reflector or other tool like confbridge in the hblink package.
    - Do you need a gateway?
      Some modes can function without a gateway (DMR and YSF). Other modes, set your *MMDVM_Bridge.ini* network to work with the gateway. In the sample MMDVM_Bridge.ini we have set the addresses and ports to point to predefined servers so that you can test.
    - Managing talk group translations
        - Simple translations are handled by export rules in *DVSwitch.ini*
          Pay attention to the word export. These rules define what is done to the the data as it is exported to a receiver.
        - More complex rules would be handled by conference bridge
    - Bridges vs Clusters
        - A bridge connects two networks together in a bidirectional mode
        - A cluster is a collections of bridges usually on a single host. The idea of a cluster is to bring many modes together into a single unified place where each mode can be heard by every other mode.
- Debugging
    - General comment. When setting up new software and hardware, change as little as possible. We have tried to set the default in the config file to sane values most people will use. Read the comments associated with the entry. And, last but not least, backup the file BEFORE you edit it.

    - How to report an error
        - Short and precise title (HELP! Is not what we have in mind here)
          The title may be the most important part of the report as it is read the most. If you want help, make a title that people want to read.
        - Concise description, steps to reproduce and if need suggested behavior
          Bug hunting is time consuming work. If you want a bug to get fixed, make sure you give enough information to reproduce the problem. What did you do? Was there something different

between when it works and when it does not? What did you expect it to do? Can you describe the audio issue? If not can you enclose a recording of the audio?

- Environment
  - Hardware platform (ARM, X86, VPS, etc)
  - Operating system and version (uname -a)
- Application version and build time stamp
  The version and build timestamp are found in the log file when the app is launched. Please make sure you are not reporting a bug that has already been fixed!
- Configuration files
  In order to reproduce the issue, we will need to re-create your environment. Your configuration files are the key element. MMDVM_Bridge.ini, DVSwitch.ini and Analog_Bridge.ini are needed at a bare minimum. Please do not just paste them in the message, enclose them. It makes our job easier. You may have multiple versions of a file (transcoders have multiple Analog_Bridge.ini files)
- Log files
  Log files are the other artifact that helps us debug an issue. What did the app do with a specific input? Was there an error? What happened before the error occurred? The log files tell us this and more. Your log files may appear in several directories. There are default locations, but your ini files can change that. You should enclose your log files in the message. The log level is important to our debugging efforts. For most things, a logLevel of 2 is sufficient detail to debug the problem. However, sometimes a very verbose version may be asked for and you would need to set the logLevel to 1.
- PLEASE avoid sending screenshots
- Most times, but not all, the first error and the last error tell you a great deal. You don't have to fully understand the error to trouble shoot. Look for key words that you recognize.
- Follow up
  Once a developer says a bug has been fixed, please re-test and make sure it REALLY has been fixed. Did we fix the bug and cause 3 more? Oops! (nah, that never happens)

- What went wrong?
  - Install errors
    - Are you on the right version of the operating system? Remember the packages require the correct version of Debian.
    - Did you update/upgrade?

        - Firewall enabled?
        - APT sources updated?
        - ASL: kernel headers for dahdi?
        - What error message did you get?
    - Can't connect to Network source
        - Are your ports and IP address correct?
        - Can you ping the IP address?
        - Is the ID correct?
        - Is the gateway running?
        - Is the gateway listening to the ports you think it is? netstat -unap is your friend.
        - Is the firewall blocking the traffic?
        - Is the network available at the time of launch? Can you connect after the system is started and you log in? That is a good indication the network is not ready when the system starts your program.
    - No traffic in either direction
        - Check your network sources
        - Check your *DVSwitch.ini* ports
        - If transcoding, check your *Analog_Bridge.ini* ports
        - Is the Pi-Star firewall enabled?
        - Check the logs. Any errors?
    - One way traffic
        - Check ports
        - Check destination network
        - Log files
            - If transcode, check the AB log files for traffic
            - If like data type check MB log files
    - Port open error messages
        - Cannot bind the UDP address
        - Make sure you do not have another instance launched (ps) and netstat -unap
        - Make sure you have your ports defined properly
        - Use netstat to find port conflicts
    - Choppy or bad audio
        - USB Latency set? Look at the startup of Analog_Bridge and you will see a log line just after the DV3000 is initialized which documents the latency of your USB port. If it is greater than 4, you should set it lower. We recommend 1. To lower the latency as root:

```
echo 1 > sys/bus/usb-serial/devices/ttyUSB0/latency_timer
```

Where USB0 is the USB port defined in Analog_Bridge.ini

        - Emulator not working? When a packet is sent to the emulator for encode/decode and that transaction times out a single log message is generated. Look at the Analog_Bridge log file for “Emulator timed out” messages.
        - Bad network connection?
        - Run AMBETest4.py? See the discussion of this utility in the below.
    - Metadata errors
        - Wrong callsigns
            - Make sure that you have set the callsigns in MMDVM_Bridge.ini and the DMR IDs in MMDVM_Bridge.ini, DVSwitch.ini and Analog_Bridge.ini. Also, make sure you are set to get updates to the ID database files on a periodic basis. The data files will be re-read by the applications every 24 hours.
        - Wrong talk groups
            - DVSwitch.ini defines the export talkgroups most modes will use.
        - Data files: DMR, NXDN
    - Other error messages
        - Got xxx and expected yyy
            - This happens when you set up an import for a specific audio type and do not get the type expected. This can happen if you have the ports wrong or if you have AB set to export the wrong AMBEType.
        - Unknown tag
    - Debugging single instance vs multiple instance
        - It is sometimes hard to find just the right information in a log file when it has the debug output from two modes intermixed in it. So, it is suggested that sometimes it is easier to take a single instance MMDVM_Bridge setup and split it up into two separate instances with each having its own log files and debug settings. This will allow you to isolate each side of the transaction and possibly arrive at a solution to the problem with less effort. Once the problem has been identified and fixed, go back to the single instance version of the application.
- Tools
    - Run program in foreground
        - Run your bridge in the foreground to help debug any problems in its execution. Each application would be

opened up in a separate terminal window so you can watch each part of the bridge operate. Once you have verified that a component is operating without any issues, then start it up in the background. Before calling it "Mission Accomplished", you can run the application in the background and tail -F the log file in the foreground. This will allow you to watch the realtime execution of the bridge while it is running it its intended state.

- Log files
  - Log files are your friend. They help in debugging problems with the bridge, verifying proper execution of the software and finding issues created by users connecting to your bridges (loopers, etc). Get familiar with the log output of each application in the system. They all use a similar format, so once you learn its layout, each one should be easy to consume. The log levels that are defined in the config files are the same, moving from DEBUG (very verbose with hex dumps, etc) to only FATAL errors being output to the logs. Normally you would set your log levels the the least amount of information needed to verify proper execution of the application. But, don't set them too low, because if you have to inspect the logs at a later time and the levels was not verbose enough, you will have lost a valuable source of information.
- netstat
  - The single most common problem with setting up a bridge is getting the port assignments correct. There are a lot of them and they are scattered across several different ini files (*MMDVM_Bridge.ini*, *DVSwitch.ini*, *Analog_Bridge.ini* and rpt.conf). Netstat is a program that will tell you which applications are listening on which ports. This will help you debug and verify that each application is listening where you thought it should be.
  - The bridge only uses UDP packets to communicate between components, so the netstat command would just show UDP type of listeners. On Debian based systems the syntax would be
  - netstat -unap
- top
  - The bridge is very efficient. Typical cpu utilization for any bridge component should be very small, in the single digits. The only component that "can" take up a greater amount of cpu utilization is the software vocoder, OP25, used in

Analog_Bridge. If you suspect that there is a utilization issue, top will allow you to see the resource needs of each of the bridge components and to verify that they are operating correctly

- AMBETest4.py
  - AMBETest4.py is a confidence test program for the DV3000U USB dongle. Over time we have seen several instances where the bridge software has appeared to fail even though the software was set up correctly. In the investigation that followed it was found that the DV3000U was rather sensitive to the voltage on the USB port. We all know that USB should supply 5V to the USB device, but we found that users had their Raspberry PI powered with supplies (wall warts) that allowed the voltage to sag when the processor or dongle was under load. In order to help users have confidence in their hardware setup, AMBETest4.py was created. This program will exercise the USB dongle and inspect the results from the dongle looking for any errors that may be produced. The program will run through a set of tests including reset, mode changes, encode and decode. AMBETest4.py is a python program, so please make sure to have a basic python runtime installed.
  - AMBETest4.py has several command line parameters that you can use
    - -v will cause verbose operation of the program
    - -e will cause the program to stop on any error
    - -s serialPort will tell the test program that the DV3000U is connected to the USB port with a full path name (/dev/ttyUSB0)
    - -i ipAddress will tell the program to use the ipAddress to send commands to ambe server
    - -n will tell the program to use the newer baud rate 460800
- Charles / wireshark
  - Although not needed very often these tools are invaluable in debugging network based problems. Packet corruption, dropped packets, slow arrival, and other conditions can be diagnosed with these tools.
- Hoseline
  If on DMR or bridging to DMR, hoseline is a good tool to use to listen to yourself.
- Parrot

If you are not on DMR, you can still monitor yourself using the parrot.

- Advanced topics
  - MMDVM_Bridge DMR vs HB_Bridge: why use one or the other?
    You might be wondering why there are two different ways of bridging to DMR, MMDVM_Bridge and HB_Bridge (and IPSC_Bridge). At first these seem to be duplicates, but in reality, each program adds unique features.

    HB_Bridge and IPSC_Bridge are part of a larger package of applications called DMRLink and HBLink which allow you to bridge homebrew (MMDVM) and IPSC (Motorola) repeaters. The package also has extensive functions for combining talkgroups together into clusters without the need for involving other network operators. DMRLink/HBLink is a mature package and is able to run on virtual servers as well as real hardware. It is written in Python and is easily installed. If you are looking to create a local cluster of repeaters with a bridge to other modes (Analog or other digital modes) then HBLink is a perfect choice. Conference Bridge is an awesome tool in this package which allows you to customize your talk groups and access rules. You are in control, not some network operator.

    MMDVM_Bridge is a easy to use, self contained application which excels in bridging digital modes together. Its best use case is when two digital modes from foreign networks need to be joined. Its configuration is familiar (the same as MMDVMHost). While you 'could' use HB_Bridge and MMDVM_Bridge together to bridge DMR to another mode, a single instance of MMDVM_Bridge will be lighter on system resources and configuration.

  - Datafile updates
  - Log rotation
  - Systemd units
    - There are several systemd units that have been installed for you. These units are there to help in starting, stopping and getting the status of any module. During development or debugging of a bridge, you will probably want to run a program in the foreground so you can see the output on your terminal, but once a program is ready to be in production, the systemd units can be used to automatically launch each service at boot. If you want to see a log for a running unit, use *tail -F* to watch the output.
    - Services installed
      - mmdvm_bridge, analog_bridge, md380_emu, AllstarLink (asterisk, updatenodelist)

        - Starting a service
            - *systemctl start serviceName*
              Where serviceName can be mmdvm_bridge, analog_bridge, md380_emu, asterisk or updatnodelist
        - Stopping a service
            - *systemctl stop serviceName*
              Where serviceName can be mmdvm_bridge, analog_bridge, md380_emu, asterisk or updatnodelist
        - Getting the status of a service
            - *systemctl status serviceName*

- Dashboards
    - YSF - https://github.com/dg9vh/YSFReflector-Dashboard
    - NXDN - https://github.com/N4IRS/MMDVM-Install/tree/master/NXDN/NXDNReflector-Dashboard
    - P25 - https://github.com/N4IRS/MMDVM-Install/tree/master/P25/P25Reflector-Dashboard
- Links to things on the web
    - Packages
        - http://dvswitch.org/install-dvswitch-repo
    - Protocol descriptions
        - http://www.qsl.net/kb9mwr/projects/dv/nxdn/NXDN-TS-1-A_v0103.pdf
        - https://www.yaesu.com/downloadFile.cfm?FileID=9036&FileCatID=263&FileName=Yaesu_Amateur%20Radio%20Digital%20Specs_1V02_EN-GB.pdf&FileContentType=application%2Fpdf
        - https://wiki.brandmeister.network/index.php/Homebrew_repeater_protocol
    - Cool projects on Git
        - https://github.com/g4klx
        - https://github.com/dl5di/OpenDV
        - https://github.com/n0mjs710/DMRlink
        - https://github.com/n0mjs710/HBlink
        - https://github.com/n0mjs710/dmr_utils
        - https://github.com/n0mjs710/DMRmonitor
        - https://github.com/boatbod/op25
        - https://github.com/juribeparada
        - Anything from DVSwitch
    - Message boards
        - DVSwitch

                - https://dvswitch.groups.io/g/support
                - https://dvswitch.groups.io/g/main
                - https://dvswitch.groups.io/g/Quantar-Bridge
                - https://dvswitch.groups.io/g/allstarlink
            - MMDVM
                - https://groups.yahoo.com/neo/groups/mmdvm/info
            - Networks
                - http://brandmeister.network/
                - http://www.dmr-marc.net/
                - http://www.dstarusers.org/
                - https://allstarlink.org/
                - http://www.nxdninfo.com/
                - http://nxmanager.weebly.com/
                - https://www.yaesu.com/jp/en/wires-x/index.php
    - Metadata transformations
        - The original model was for DMR so you will see most of the fields support this mode
        - The system tries as hard as it can to make a valid DMR ID / callsign that matches the transmission
        - Datafiles files are maintained for DMR and NXDN IDs and callsign
        - The rules are often mode dependant
            - DMR assumes the metadata it imports is perfect and exports DMR ID and callsign if found in the datafile
            - P25 uses the DMR datafile for lookup and rules
            - Yaesu will use callsign then DMR ID upon import and will lookup callsign in datafile and set DMR ID if found. If not found it will export the fallback ID
            - D-Star will use callsign then DMR ID upon import and will lookup callsign in datafile and set DMR ID if found. If not found it will export the fallback ID
            - NXDN will use the callsign then the ID to lookup the NXDN ID on import. On export it will set the DMR ID and callsign if found.
- Common Configurations
    - Analog bridge to digital network
        - Example: NXDN to ASL
        - Important parts to demonstrate
            - ASL -> NXDN metadata (since none comes from ASL)
            - Audio level setting
            - Vocoder setup and usage
            - ASL configuration files
    - Digital bridge to like format
        - Example: YSFn to DMR

        - Important parts to demonstrate
            - Single instance setup
                - Enabling 2 modes in *MMDVM_Bridge.ini*
                  The example of a YSFn to DMR is the simplest bridge you can build. There are no gateways required. You can use a single instance of MMDVM_Bridge.
            - *DVSwitch.ini* port crossover
            - Datafile locations
    - Digital bridge to unlike format
        - P25 to DMR
        - Important parts to demonstrate
            - MB -> transcoder -> MB setup
            - Audio level setting
    - Multi-mode cluster
        - Example: P25, DMR, NXDN, YSFN, D-Star, ASL
- Analog_Bridge
    - Used to convert a network stream of TLV packets into PCM and back again
    - Two usage scenarios
        - Bridge ASL to a digital network
            - ASL has a great channel driver called *chan_usrp*. This driver was developed to connect ASL to specialized hardware, but does not actually require the hardware. The USRP protocol includes signaling (TX/RX transitions) and audio transfer using UDP. The audio format is 8KHZ signed 16 bit PCM samples. Analog_Bridge was developed to consume these packets and transcode them to several digital audio formats.
        - Transcode one digital format to another
            - The process of transcoding one digital audio format to another is to take the source audio format and convert it to a form that can then be encoded into a new format. All of the audio formats encode PCM into digital audio and decode digital audio into PCM, so this can be used to be the basis of the transcoder. In general terms, you use two instances of Analog_Bridge, one to handle the first format of digital and the second instance to handle the other format. These instances are then set to feed each other in the stream and presto, transcoder.
            - To create a transcoder from format A to format B follow these general steps. In the description below substitute A and B for your digital format like DMR, NXDN, etc. First take a “clean” Analog_Bridge.ini file and create two new files Analog_Bridge_A.ini and Analog_Bridge_B.ini. Edit the format A file first and change

            - Below where ever it says **MODE_X** use your mode...
            - [AMBE_AUDIO]
                - ambeMode = **MODE_X**
                - txPort = the.rxPort.for.**MODE_X**.in.dvswitch.ini
                - rxPort = the txPort.for.**MODE_X**.in.dvswitch.ini
            - [USRP]
                - txPort = xxxx
                - rxPort = yyyy
            - Vocoder
                - If **MODE_X** is DMR, YSFn or NXDN, enable the emulator and use it on a unique port
                - If **MODE_X** is IMBE or YSFw disable the emulator and DV3000
                - If **MODE_X** is D-Star set up the DV3000 as a serial or IP connection to AMBEServer
        - Now, edit Analog_Bridge_B.ini and do the same thing, paying SPECIAL attention to the [USRP] txPort and rxPort. They should be reversed from the Analog_Bridge_A.ini version.
- Connecting to ASL
    - Analog_Bridge speaks USRP
    - Turn on the USRP channel driver in *modules.conf*
    - Set up the *rpt.conf* to match the ports in *Analog_Bridge.ini*
    - Do NOT send Allison to the digital network!
    - Private node?
- Transcode
    - Two instances of AB are required
    - Each instance will point (crossover) their PCM data at each other
    - The TLV ports will point at each respective Partner application
    - Partner1 < -- > AB1 < -- > AB2 < -- > Partner2
        - AB1 ambeMode will match Partner1 audio format
        - AB2 ambeMode will match Partner2 audio format
    - Use two versions of the *Analog_Bridge.ini* file, one for each side of the transcode
- Defining input and output networking ports
- Setting audio levels
    - Start with AUDIO_UNITY for both dmrAudio and aslAudio settings
- Defining the vocoder to use
    - Analog_Bridge is capable of using two classes of vocoders broken up into 3 different implementations.
        - Hardware DV3000U, PiDV, DVMega AMBE
            - Support for DMR, YSFN, NXDN and D-Star
            - Direct serial and IP (ambe server) modes are supported
        - Software

            - The md380-emulator
                - Support for DMR, YSFN, NXDN
            - The OP25 vocoder
                - Support for DMR, YSFN, NXDN, D-Star, P25 and YSFW
        - Quality vs cost
        - Will your CPU support all vocoders?
            - When we need to transcode one audio format to another we need to be able to decode from format A into PCM and then encode from PCM into format B. This means we would need to make sure your environment has support for both vocoders in the transcode sequence.
            - As described in other sections, we support AMBE, IMBE and DSAMBE audio formats, but there are differing levels of support and quality for these formats.
            - AMBE is supported in hardware (USB or AMBEServer), md380-emulator on supported environments and OP25 vocoder for all platforms. The md380-emulator is available on x86 and ARM V7 (with div support) which includes the Raspberry Pi 2 and 3, Allwinner H3 and H5 processors. Other ARM processors should invoke the qemu-arm emulator.
            - IMBE is supported in software on all platforms using the OP25 vocoders
            - DSAMBE (D-Star) is supported in hardware (USB or AMBEServer) and (but you will not be happy) by using the OP25 vocoder.

    - Defining the default metadata
    - Debugging
        - DV3000 errors
            - Extensive checking of each DV3000 packet is done
            - Upon finding an error, it is reported in the log
            - Errors include
                - DV3000 header is short
                - DV3000 start byte not found
                - DV3000 packet size does not match bytes in stream
                - DV3000 has unexpectedly reset
                  You can almost bet this is a power problem.
                - DV3000 return type not correct
- Quantar_Bridge
    - Connects a Quantar P25 repeater to an MMDVM network
    - For each talk group there is a reflector.
    - Each reflector can be on a single host or a host can have more then one reflector

- Prerequisites
    - Requires a Cisco router supporting STUN
        - Recommended hardware
        - Recommended software version
    - Requires P25Gateway to connect to the network
- Quantar_Bridge system configuration
    - Cisco router configuration
    - The [QUANTAR] stanza
        - logFilePath should be set to the full path and file name of your log file. The log file may be rotated without stopping Quantar_Bridge (logrotate).
        - logLevel is defined as
            - 0=No logging, 1=Debug, 2=Message, 3=Info, 4=Warning, 5=Error, 6=Fatal
        - Address should be set to the IP address of the Partner application receiving the TLV frames from Quantar_Bridge (probably MMDVM_Bridge or Analog_Bridge)
        - TXPort should be set to the port that the Partner application is listening on for TLV frames
        - RXPort should be set to a unique port number that Quantar_Bridge is listening on for TLV frames
        - quantarPort should be set to the port defined in the Cisco router config. Quantar_Bridge will listen on this port for STUN/HDLC frames from the Cisco router.
    - Partner configuration
        - Make sure there is a listener on the port defined as TXPort in the [QUANTAR] stanza
        - If using Analog_Bridge, make sure that the exported TLV audio format is IMBE
    - Debugging
        - Turn on the debug setting in the [QUANTAR] stanza. This produces a LOT of data. You will not want this enabled for long. This also forces logLevel to be 1, so no need to set that as well.
- Advanced topics
    - Using a hotspot on my P25 network
        - With Quantar_Bridge this is easy. Just make sure that your repeater and the hotspot are connected to the same reflector and you are good to go.
    - Using a Yaesu Fusion radio in wide mode on the P25 network
        - You have two options to use this radio on your network. One is to use the ysf2p25 application to gateway into the p25 reflector. The other is to set up a bridge to a Yaesu Fusion Wide reflector and your P25 reflector using MMDVM_Bridge.

- HB_Bridge
    - Used to connect the bridge to a HomeBrew network
    - Provide a server for HB clients to connect to
    - HBLink is a subclass of HBSYSTEM (hblink.py)
        - The subclass inherits all of the capabilities of its superclass
        - HB_Bridge does not require one to run hblink.py
            - DO NOT RUN IT except for testing (see below)
    - Debugging HB_Bridge
        - Now, and only now can you run hblink.py stand alone
- IPSC_Bridge
    - Used to connect the bridge to a IPSC network
    - Provide a Master for IPSC clients to connect to