DSA Test scripts

Andrew Lunn <andrew@lunn.ch>

v0.1

Chapter 1

Introduction

This document is a brief introduction to the testing scripts i've hacked together for DSA switches. It describes the hardware currently being tested, the hardware setup, software setup, how to run the tests and what the current tests do. Lastly a section lists further work.

Chapter 2

Hardware Test Setup

The tests have been developed in order to test DSA on the Zodiac Development board. This is a somewhat unusual device in that it has three switches, allowing D in DSA development and testing. It has a limited number of ports from each switch coming to Copper PHYs and two ports from switch #2 are connected to SFF fibre modules. The switches are arranged in a chain, with a DSA link between each switch. Figure 2.1 is a block diagram for this board, and how it is connected to the test host.

The test host is a standard desktop machine with two quad Ethernet cards, in addition it its normal Ethernet device. These 8 devices have been cabled to eight of the test devices ports, making use of a media converter to connect to one of the fiber interfaces.

A second test system has been setup using more conventional hardware. A Marvell reference design board, with a single 7 port switch is used. 4 of these ports come to RJ45 sockets. Probably any wireless access point could be used. However it is recommended that the management Ethernet interface, used for SSH access to the device, is not one of the switch ports, but a dedicated Ethernet interface. Four USB-Ethernet dongles are used to provide the host interfaces. Such devices appear as normal Ethernets interfaces in Linux. However it appears that configuring the interface down on such USB-Ethernet devices does not cause the carrier to be dropped.

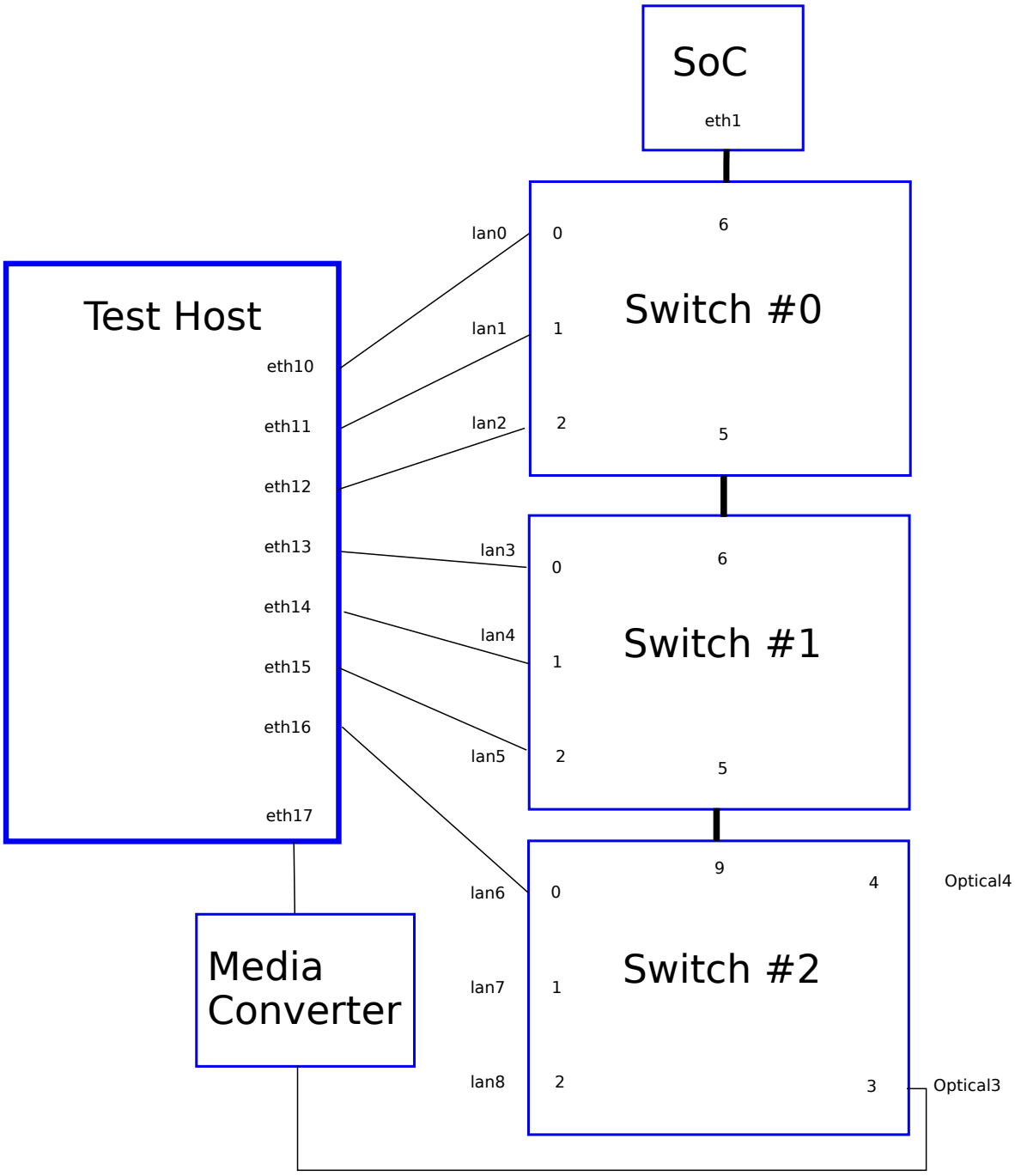

Figure 2.1: Zodiac Development board connections to test Host

Chapter 3

Test Software Setup

3.1 Ostinato

Ostinato is a software packet for generator and capturing test streams of Ethernet frames. It has the home page `http://ostinato.org/`. Although Debian has a package for it, the python bindings are not included. So it has been built from source. This in tern requires that googles protobufs were also built from source, since Debian's version fails to compile Ostinato.

The Ostinato architecture consists of a Drone which generates frames and captures results, and a front end which configures the drone. This front end can either be the Ostinato qt GUI, or any script using the python bindings.

The drone has a configuration file, /etc/xdg/Ostinato/drone.ini. This should contain what is shown in listing 3.1 in order to limit the drone to just the test interfaces.

```
[PortList]

Include=eth1?
Exclude=usbmon*
```

Listing 3.1: Drone configuration File

3.2 udev rules for Interface naming

In order to separate the test interfaces from the normal interfaces on the test host, udev rules have been used to rename the interfaces eth10, eth11,.. eth17. These rules can be placed in /etc/udev/rules.d/70-persistent-net.rules and should look like those in listing 3.2. The MAC address is the most important match and using it also results in deterministic names for the interfaces.

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:26:55:d2:27:a8", ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="enp*", NAME
       ="eth10"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="00:26:55:d2:27:a9", ATTR{dev_id}=="0x0", ATTR{type}=="1", KERNEL=="enp*", NAME
       ="eth11"
```

Listing 3.2: udev rules for test interfaces

3.3 Additional Python packages

The scripts need to execute commands on the system under test. To do this, the python package paramiko is used. This can be installed in the usual way from the distribution repository. For Debian it is called python-paramiko. The tests make use of pythons unittest2. This is commonly not installed by default. For Debian, it is called python-unittest2.

3.4 System Under Test software

A standard ARM Debian distribution has been used on the Zodiac board. However, very few commands are required. Currently these are ip and brctl. Additionally, it must be possible to ssh into the device. It is likely the tests can be made to work with busybox/barebox, and dropbear. However, it is recommended to have the real ip and brctl commands installed, not cut down alternatives.

3.5 Test Scripts

The test themselves can be found at: `https://github.com/lunn/dsa-tests.git`.

Chapter 4

Tests

4.1 Common command line parameters and Configuration file

All tests currently share the same command line parameters, and shown in listing 4.1. –host is mandatory. Login credentials must be provided, either the root password, or an ssh key which allows root to login without a password. Note that the ~/.ssh directory is not searched for the key, so pass either an absolute or relative path to the key file. A verbose level of 2 shows each test as it is run. Enabling failfast is useful for debugging a failed test. The System Under Test will be left in the state the test failed, making it possible to log in and take a look around.

```
./bridge_test.py --help
usage: bridge_test.py  [-h] --config CONFIG [--verbose VERBOSE][--failfast]
Run some Bridge tests.

optional arguments:
 -h, --help            show this help message and exit
 --config, -c CONFIG   Configuration for for system under test
 --verbose VERBOSE, -v VERBOSE
                       Run the test with a verbose level
 --failfast, -f        Exit the test as soon as a test fails
```

Listing 4.1: Test command line options

Some test need to manipulate the test hosts interfaces. Such tests must be run as root, preferably with sudo. An error will be generated when root permission is required but not available.

The configuration file uses standard INI syntax. An example is shown in listing 4.2. The host section contains configuration for the host executing the test. It lists the host interfaces which connect to the system under test. The sut section contains configuration for the system under test. It lists the interfaces which are connected to the host. The interfaces for configuration key 'lan0' in the host and sut section are assumed to be wired together. The hostname key is a host name, or IP address for the system under test. The test scripts will execute SSH commands on this device. In order to allow SSH login, the script will make use of the SSH key indicated in the key key of section sut.

Listing 4.2: Configuration File

```
[host]

lan0 = eth10
lan1 = eth11
lan2 = eth12
lan3 = eth13

[sut]

master = eth1
lan0 = lan0
lan1 = lan1
lan2 = lan2
lan3 = lan3

hostname = 370-rd
key = /home/andrew/.ssh/zii-devel
```

4.2 Test: ping_individual_test.py

This test aims to show that the SUTs interfaces can be used as individual interfaces.

This test creates 8 subnets, one per test host/SUT interface connection. Pings are then performed across each link from the test host to the sut, which is expected to reply. The SUT interfaces are then configured down and the pings repeated. No replies are expected.

4.3 Test: ping_individual_4_ports_test.py

This is similar to the previous test, but makes use of only 4 ports. The test aims to show that the SUTs interfaces can be used as individual interfaces.

This test creates 4 subnets, one per test host/SUT interface connection. Pings are then performed across each link from the test host to the sut, which is expected to reply. The SUT interfaces are then configured down and the pings repeated. No replies are expected.

4.4 Test: ping_bridges_test.py

This test is an extension of the previous test. Once it has been shown that individual interfaces can be pinged, bridges are added. Each bridge contains a single interface, and the IP address is moved from the interface to to bridge. Thus it is expected that the ping continues to be successful. The bridges are then removed and the IP address placed back onto the interface. Again, it is expected the pings continue to be successful.

4.5 Test: ping_bridges_4_ports_test.py

This is similar to the previous test, but makes use of only 4 ports. Once it has been shown that individual interfaces can be pinged, bridges are added. Each bridge contains a single interface, and the IP address is moved from the interface to to bridge. Thus it is expected that the ping continues to be successful. The bridges are then removed and the IP address placed back onto the interface. Again, it is expected the pings continue to be successful.

4.6 Test: bridge_test.py

This test is still WIP.

This test makes use of Ostinato to generate traffic. Thus the drone must be running. This allows testing of bridges with more than one interface, which cannot easily be tested using ping from one test device. A bridge is created with members from each switch. However some ports are also left out or the bridge. In order for the switch and bridge to learn what MAC addresses are on what ports, two broadcast packets are sent to each port using the MAC address unique to each port. Traffic is then send to the ports which are not members of the bridge. These packet are either ingress on ports which are not a member of the bridge, or are addresses such that they would egress out a port which is not a member of the bridge, or have the broadcast destination address. It is expected that there packets do no appear on any other interface. Packets are then sent to ingress ports which are members of the bridge and destined to egress ports which are also members of the bridge. The packets are then received and counted, to ensure packets are not lost or duplicated.

Chapter 5

Further Work

1. Add a ping_individual_vlan_test.py, which passes VLAN tagged frames as well as untagged frames, Make use of vconfig to create vlan devices and place IP addresses on these interfaces, which should be pingable.
2. Add a bridge_vlan_test.py which tests the VLAN capabilities of bridges.
3. More tests...