Label Switch Router (LSR) for NetFPGA

User Manual, Version 1.5
May 10, 2012

Developed by Algo-Logic Systems for Google

http://Algo-Logic.com

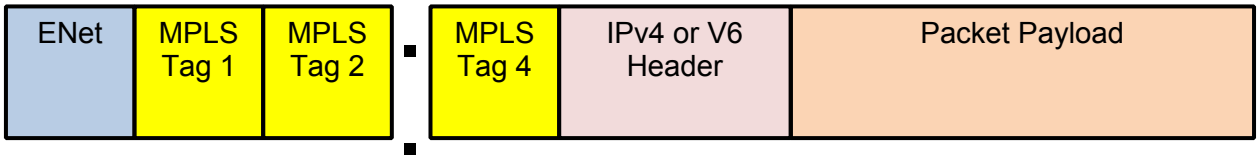
Contents

1. Introduction

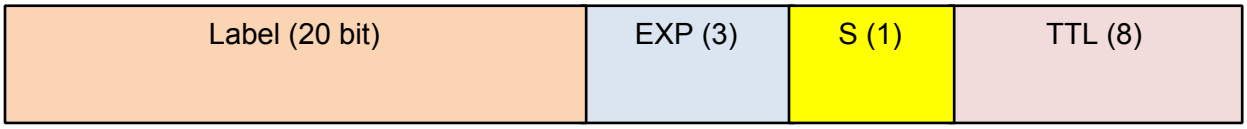
Multiprotocol Label Switching (MPLS) is a mechanism that allows networks to carry data from one network node to the next with the help of short labels. MPLS enables the creation of virtual links between distant nodes and can readily forward packets between data centers by encapsulating packets of multiple network protocols.

The MPLS Label Switch Router (LSR) is a hardware-accelerated LSR implemented built by Algo-Logic Systems for Google that adds MPLS label switching functionality over and beyond the default NetFPGA reference switch.

The structure of an MPLS frame with four MPLS tags is shown below.



MPLS headers can be seen immediately after the EtherType field in the Ethernet Header. Each MPLS header is 32-bits long with the following format:



“Label” is a 20-bit field used as a label for switching operations.

“EXP” is the 3-bit experimental bits field. This implementation does not use it.

“S” is the Bottom of Stack bit. If S is “1” it indicates this header is the last one in the MPLS stack.

“TTL” is an 8-bit Time-To-Live field used to detect forwarding loops.

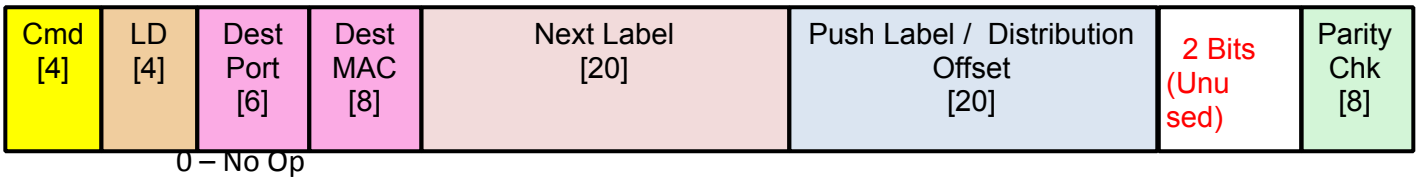
2. MPLS Operation Commands

The MPLS operations implemented are:

- **Swap**
- **Push**
- **Swap+Push**
- **Pop**

- **Pop+Swap**

The operation to be performed on the received MPLS frame is determined by a 72-bit word stored in the SRAM of the 1G NetFPGA card. Whenever a frame enters the NetFPGA hardware pipeline, the 20 bit label is extracted from the top-most MPLS header and used as an address for a lookup operation into the corresponding 72-bit word in SRAM. The format of the 72-bit word stored in SRAM is shown below:



0 – No Op

1 – Swap

2 – Push

3 – Swap & Push

4 – Pop

5 – Pop & Swap

LD - Load Distribution Count: 4 bits

The number of load distribution entries starting from the Load distribution offset. Default value is 0, up to 4 entries are supported.

Dest Port - 6 bits
Address of the physical output port (nf2c0, nf2c1 …)

nf2c0 = port 0
nf2c1 = port 2
nf2c2 = port 4
nf2c3 = port 6

Dest MAC - 8 bits
An index into a 256-entry table of next-hop MAC addresses stored in on-chip BRAM

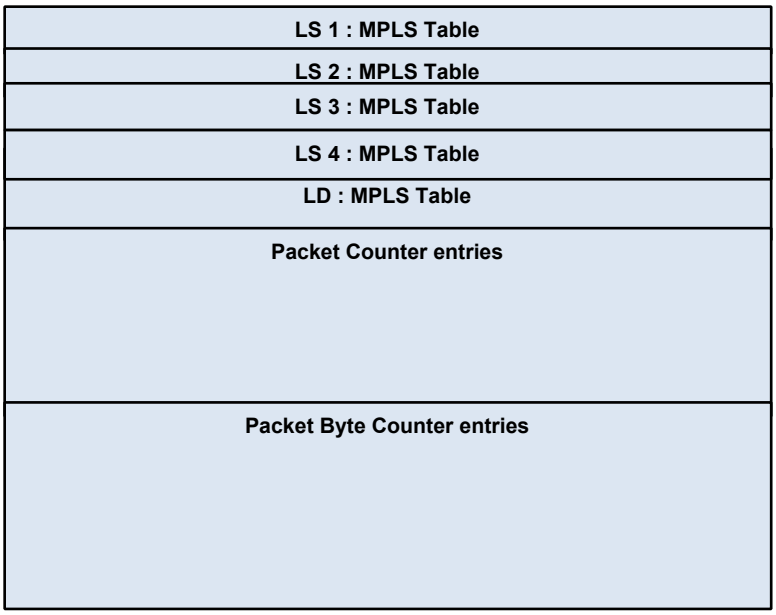
Next Label
A 20-bit MPLS label used as a lookup address into the SRAM table, to determine the label to swap in the first MPLS header.

Push Label / Distribution offset
For commands that push additional MPLS headers, these 20 bits define the MPLS label to be pushed onto the top of the stack. If the LD field > 0 these 20 bits will be the starting address of the load distribution entries.

Parity/Checksum
8-bit check of bits.

3. Label spaces and partitions in SRAM

The SRAM in a 1G NetFPGA board can store 256k words with a width of 72 bits. The total label space (not the total SRAM space) is first divided into two equal halves, one for input label spaces and one for load distribution. The input label spaces are further divided in to four equal-sized spaces and each space is assigned to an input port (LS1, LS2, LS3 and LS4). The load distribution entries have their own space named LD space. Each entry in both the LS space and the LD space has two counters. The first counter indicates the total number of input frames hitting that particular entry and the second counter indicates the total byte count of those frames. Each of these counters takes 1/3 of the total SRAM space. The whole SRAM space allotted for these spaces is shown below.



4. SRAM Lookup

If the Ethernet parser finds an MPLS unicast packet, the first 20 bits from the top-most MPLS header are extracted and added to a constant value (software offset) to determine the lookup address. After the 72-bit SRAM word lookup is performed using that value, the Ethernet parser module will check the 4-bit LD value in the retrieved entry. If the LD value is zero, the received SRAM word is latched and the corresponding packet counter and packet byte counter is incremented. If the LD value is not equal to zero the 20-bit LD offset value is then latched. Let's suppose n is the LD value. This indicates there are n valid LD entries starting from the address 'LD offset', and from these n entries one is to be selected for later MPLS operations. To make this selection, a hash value is generated by making use of the following parameters:

Source IP address (32 bits) of the frame.

Destination IP address (32 bits) of the frame.

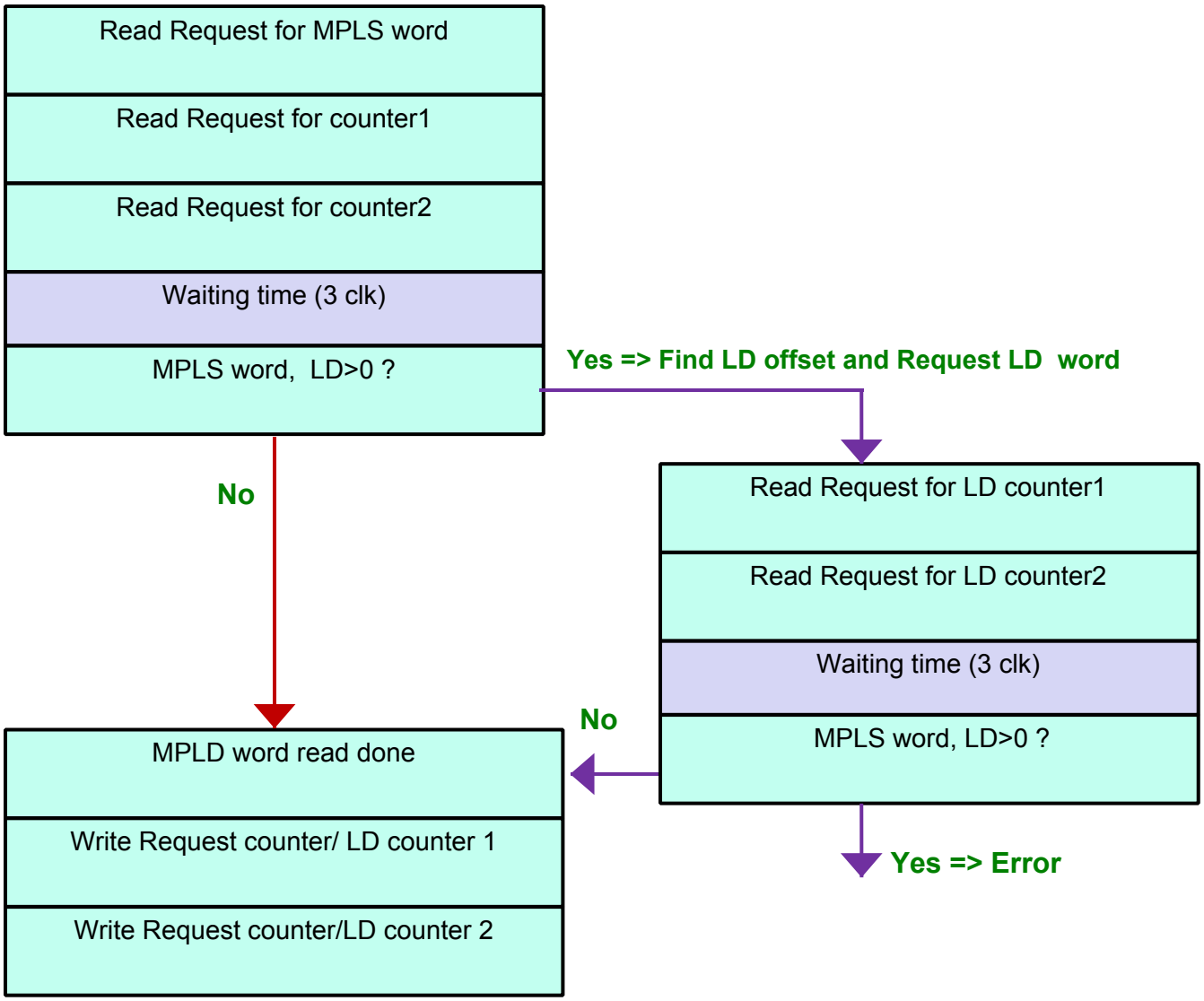
TCP source port (16 bits) if the packet is a TCP packet (or UDP source port if the packet is a UDP packet)

TCP destination port (16 bits) if the packet is a TCP packet (or UDP destination port if the packet is a UDP packet)

Let k is the hash value, then (k modulo n) will give a number p whose value will be in between 0 and n-1.

The SRAM address of the LD entry selected = (LD offset + p)

A **second** SRAM look up operation is performed for this selected address. After latching, the corresponding 72-bit SRAM word packet counter and 72-bit SRAM word packet byte counter is incremented. Because this second lookup is required, LD is not supported with the Pop&Swap opcode as it exceeds available memory bandwidth.



5. BRAM Lookup

The dest MAC field (8 bits) of the latched 72-bit SRAM word is used as an index into a BRAM lookup, to get the destination MAC address of the outgoing frame. A maximum of 256 MAC addresses can be stored in the BRAM.

6. Packet flow through the NetFPGA hardware pipeline

As a frame is delivered to a NetFPGA port, the MAC receiver unit accepts and directs it to the output look-up module via the input arbiter.

In the output look-up module, the header of the frame is parsed by the header parser module. It latches all the header information in a status FIFO. The frames are stored in a FIFO in the look-up module.

The actions in the output look-up module place in the following order:

- If the packet is MPLS multicast it is forwarded to the host CPU.

- If the MAC multicast bit is not set and the destination MAC address is not matching, the frame is dropped and the corresponding counter (the not for us counter) is incremented.

- If the frame is MPLS unicast and the SRAM lookup address derived from its top-most 20-bit MPLS header is outside of the expected label space, the packet is dropped and the corresponding counter (LS error counter) is incremented.

- If the frame is not MPLS unicast it is forwarded to the host CPU.

- If the frame is MPLS unicast, it waits for the MPLS look-up action to complete.

- After getting the 72-bit word from SRAM, the corresponding packet counter and packet byte counter in SRAM is incremented.

- The index for the destination MAC address is latched from the 72-bit SRAM word and a BRAM lookup is performed.

- If there a parity error in the look-up SRAM word it is dropped and the corresponding counter is incremented.

- If there is no error, the MPLS unicast packet with the modified destination MAC and output port are passed to the MPLS processing unit along with the 72-bit look-up word from SRAM.

- The MPLS processing unit check the TTL and if it is expired forwards the frame to the host CPU and the corresponding counter is incremented, otherwise it modifies the frame header according to the “cmd” field in the SRAM word. The frame is then passed to the output queue module.

- The output queue module checks the output port address and forwards the frame to the corresponding DRAM queue and then it is forwarded to the corresponding MAC transmitting unit.

7. Register Interface

The 72-bit word entry in SRAM and the output MAC address entry in BRAM are populated using the register interface. The addresses of all registers, the SRAM base and the BRAM base are defined in ~/netfpga/projects/mpls_switch/lib/C/reg_defines_mpls_switch.h

The NetFPGA register interface supports the reading and writing of 32-bit wide words. In order to write a single 72-bit SRAM entry we make use of SRAM address locations between "0Xxxxxxx0" and "0Xxxxxxxf" as follows. The 32 bits of SRAM entry in the LSB side are buffered if we use the address location 0XxxxxxxC, the next 32 bits (middle of the SRAM entry) are buffered for the address location 0Xxxxxxx8 and the next 8 bits (MSB side) are buffered for the address location 0Xxxxxxx4. These buffered 72 bits are written to SRAM when we write a trigger value of "0xffffffff" to the address location "0Xxxxxxx0". The following example illustrates the procedure to write a 72-bit SRAM entry "0X12345678ABCDFFFF98" to the SRAM locations from address 0X1000000 to 0X100000f.

```
writeReg(&nf2, 0x100000C, 0xCDFFFF98);
writeReg(&nf2, 0x1000008, 0x345678AB);
writeReg(&nf2, 0x1000004, 0x00000012);
writeReg(&nf2, 0x1000000, 0xffffffff);
```

Writing a MAC address in BRAM follows the same procedure. The following example illustrates how we write MAC address "12345678ABCD" to the BRAM region spanned between "0x2002030" and "0x200203f".

```
writeReg(&nf2, 0x100000C, 0x5678ABCD);
writeReg(&nf2, 0x1000008, 0x00001234);
writeReg(&nf2, 0x1000000, 0xffffffff);
```

Other important registers needed for proper MPLS operations are detailed below:

```
#define SWITCH_OP_LUT_MAC0_HI_REG          0x2000100
#define SWITCH_OP_LUT_MAC0_LO_REG          0x2000104
#define SWITCH_OP_LUT_MAC1_HI_REG          0x2000108
#define SWITCH_OP_LUT_MAC1_LO_REG          0x200010c
#define SWITCH_OP_LUT_MAC2_HI_REG          0x2000110
#define SWITCH_OP_LUT_MAC2_LO_REG          0x2000114
#define SWITCH_OP_LUT_MAC3_HI_REG          0x2000118
#define SWITCH_OP_LUT_MAC3_LO_REG          0x200011c
```

The above registers are used for storing the MAC addresses of the NetFPGA ports. This MAC address is used for matching with the destination MAC address of the incoming packets.

```
#define SWITCH_OP_LUT_LS1_BASE_REG          0x2000120
#define SWITCH_OP_LUT_LS1_BOUND_REG         0x2000124
#define SWITCH_OP_LUT_LS2_BASE_REG          0x2000128
#define SWITCH_OP_LUT_LS2_BOUND_REG         0x200012c
#define SWITCH_OP_LUT_LS3_BASE_REG          0x2000130
#define SWITCH_OP_LUT_LS3_BOUND_REG         0x2000134
#define SWITCH_OP_LUT_LS4_BASE_REG          0x2000138
#define SWITCH_OP_LUT_LS4_BOUND_REG         0x200013c
#define SWITCH_OP_LUT_LD_BASE_REG           0x2000140
#define SWITCH_OP_LUT_LD_BOUND_REG          0x2000144
```

The above registers are used for storing the base addresses and bound sizes for the four label spaces and one LD space.

```
#define SWITCH_OP_LUT_SOFT_OFFSET_REG       0x2000150
```

This register is used to load the software offset. The content of this register is added to the 20 bit MPLS label of the topmost MPLS header in the incoming frame to determine the SRAM lookup address.

```
#define SWITCH_OP_LUT_NOT_FOR_US_REG        0x2000154
```

The content of this register counts the number frames dropped due to mismatching destination MAC address errors.

```
#define SWITCH_OP_LUT_PARITY_ERROR_REG      0x2000158
```

The content of this register counts the number of times a parity error was found doing an SRAM word lookup.

```
#define SWITCH_OP_LUT_TTL_ERROR_REG         0x200015c
```

The content of this register gives the TTL error count.

```
#define SWITCH_OP_LUT_LS_ERROR_REG          0x2000160
```

This counter is incremented if the index for the SRAM lookup is not contained within the assigned label space for the incoming frame's port.

```
#define SWITCH_OP_LUT_LD_ERROR_REG          0x2000164
```

This counter is incremented if the index for the SRAM lookup is no contained within the LD label space during an LD indexing.

8. Software

Software programs to help populate SRAM, BRAM, and other register entries are provided in ~/netfpga/projects/mpls_switch/sw

swap is the command for creating a swap entry in SRAM with LD value zero, using the following parameters:

- Destination port number

- SRAM entry number
- BRAM MAC entry number to rewrite the destination MAC address with
- The MPLS label to swap into the topmost MPLS header

push is the command for creating a push entry in SRAM with LD value zero, using the following parameters:

- Destination port number
- SRAM entry number
- BRAM MAC entry number to rewrite the destination MAC address with
- The MPLS label to push onto the existing MPLS stack

pop is the command for creating a pop entry in SRAM with LD value zero, using the following parameters:

- Destination port number
- SRAM entry number
- BRAM MAC entry number to rewrite the destination MAC address with

spush is the command for creating a swap+push entry in SRAM with LD value zero, using the following parameters:

- Destination port number
- SRAM entry number
- BRAM MAC entry number to rewrite the destination MAC address with
- The MPLS label to swap into the existing topmost MPLS header
- The MPLS label to push underneath the swap label

pswap is the command for creating a pop+swap entry in SRAM with LD value zero, using the following parameter:

- SRAM entry number

ld is the command for creating an entry in SRAM with a nonzero LD, using the following parameters:

- LD value
- SRAM entry number
- Distribution offset

mac_out is the command for creating a MAC entry in BRAM, using the following parameters:

- The MAC address to be added, without colon or space in between
- BRAM entry number

lsld_init is the command for creating the label spaces, ld space, and packet counter base and

bound, and it should be followed by

- Label space 1 base address
- Label space 1 bound (number of entries)
- Label space 2 base address
- Label space 2 bound (number of entries)
- Label space 3 base address
- Label space 3 bound (number of entries)
- Label space 4 base address
- Label space 4 bound (number of entries)
- LD space base address
- LD space bound (number of entries)
- Packet counter space base address
- Packet byte counter space base address

mac0_add, mac1_add, mac2_add, mac3_add are the commands for creating MAC addresses of the corresponding MAC ports and these should be followed by

- MAC address without colon or space in between

lsr_init is the command for initializing the label space registers, LD space registers, software offset register, MAC address registers, and error count registers with default values.

Following are some underlying C functions which can be used to perform MPLS operations.

`lsr_fn` is a function which can be used to enter a 72-bit SRAM entry.

```
int lsr_fn(int dest_port,int ld_entry, int mac_bram_entry, int
next_label, int next_label2,int op, int entry_num)
```

- `dest_port` – Destination port
- `ld_entry` - Load distribution entry
- `int mac_bram_entry` – Index of BRAM to read the output MAC Address
- `next_label` – Label used for the swap operation (input any value if it is not used)
- `next_label2` - load distribution offset or push value (input any value if it is not used)
- `op` – MPLS cmd (value between 0 to 5)
- `entry_num` – Index of SRAM to write the MPLS word

`mac_out` is the function to write a MAC address entry to the BRAM lookup Table.

```
int mac_out(int mac_oh, int mac_ol, int entry_num)
```

- `mac_oh` – value for 16 bits on the MSB side of the MAC address
- `mac_ol` – value for 32 bits on the LSB side of the MAC address
- `entry_num` - Index of BRAM to write the MAC address

`lsr1_init` is the function to write Label space 1's starting index and bound (number of entries in SRAM) to the corresponding registers .

```
int lsr1_init(int base_address, int bound)
```

- `base_address` – Starting address of label space 1 in SRAM
- `bound` – Number of entries for tlabel space 1 in SRAM

Similarly `lsr2_init`, `lsr3_init`, `lsr4_init` can be used to enter base and bound values for other label spaces.

`lsr_ld_init` is the function to write the LD space's starting index and bound to the corresponding registers .

```
int lsr_ld_init(int base_addr, int bound)
```

- `base_addr` – Starting address of LD space in SRAM
- `bound` – Number of entries for the LD space in SRAM

`cnt_base_init` is the function to write the packet counter base address and packet byte counter base address to the corresponding registers.

```
int cnt_base_init(int packet_counter_base_addr, int
byte_counter_base_addr)
```

- `packet_counter_base_addr` – Starting address of the packet counter block in SRAM
- `byte_counter_base_addr` – Starting address of the packet byte counter block in SRAM

`soft_offset_init` is the function to write the software offset value to corresponding register.

```
int soft_offset_init(int soft_offset)
```

- `soft_offset` – software offset value

`mac0_init` is the function to enter the MAC address of Port 0 of the NetFPGA board

`int mac0_init(int mac0_hi, int mac0_lo)`

- `mac0_hi` – value for 16 bits of the MSB side of the MAC address
- `mac0_lo` - value for 32 bits of the LSB side of the MAC address

Similarly `mac1_init, mac2_init, mac3_init` can be used to enter the MAC addresses for the other three ports.

`lsr_init` is the function which set up default values in different registers

`int lsr_init()`

This function initializes the following default parameters:

- Label space and LD space base and bound values. Equally distributed entries for all the four label spaces (corresponds to each port) and LD space. default is to assign each port N=8888 h=34,952d entries of the total of 512k entries 3 table regions
- The packet counter and packet byte counter base addresses
- The offset value to be added to input label value (-1,000,000 in 2's complement; this default is useful when interoperating with Junos as its static label range starts at 1,000,000)
- The MAC address of nf2c0: MSB,LSB = 00:90:69:B1:D0:7E
- Resets the packets arriving with MAC != self counter
- Resets the Parity errors in SRAM lookups counter
- Resets the TTL expirations counter
- Resets the Label Space out-of-bound error counter
- Resets the Load Distribution error counter

User can also read and write any registers using the regread and regwrite commands found in ~/netfpga/lib/C/reg_access. For example, one can use regread 0x0600028 to read the number of packets transmitted through the MAC-0 port.

9. Implementation and testing

Step 1. Starting at the top-level of a NetFPGA base distribution (typically /home/user/netfpga), extract the tarball file to Populate the lib and project/mpls-switch sub-directories.

eg:

```
cd ~/netfpga

tar xvf lsr-netFPGA.tar
```

Step 2. Download the LSR bitfile to the NetFPGA:

```
nf_download ./bitfiles/mpls_switch.bit
```

Step 3. Compile the MPLS control and configuration software

```
cd ./projects/mpls_switch/sw
./make
```

Step 4. Run the command-line program to initialize the LSR switch

```
./lsr lsr_init
```

This will initialize the tables and counters with default values as documented earlier.

Step 5. For testing:

* Connect eth1 (NIC) to nf2c0 (Port 0 of NetFPGA) with a short cat5e cable,
* Connect eth2 (NIC) to nf2c1 (Port 1 of NetFPGA) with a short cat5e cable,
* Run wireshark on eth2 to monitor output packets
* Run command like:

```
cd /../pcap_files
tcpreplay intf1=eth1 preswap.cap
```

to verify the operation of the LSR switch with sample packets

The cable connections between the Host machine and the NetFPGA board are shown below:

Step 6. To add entries to SRAM to perform specific swap, pop, push, swap+push, and pop+swap operations, use the command syntax documented above by running

./lsr

10. Results :

A wireshark capture of the packets sent to the NetFPGA port 0, and the packet output from NetFPGA port 1 is shown below.

Swap operation

The following commands are used for setup (see earlier sections for parameter explanations):

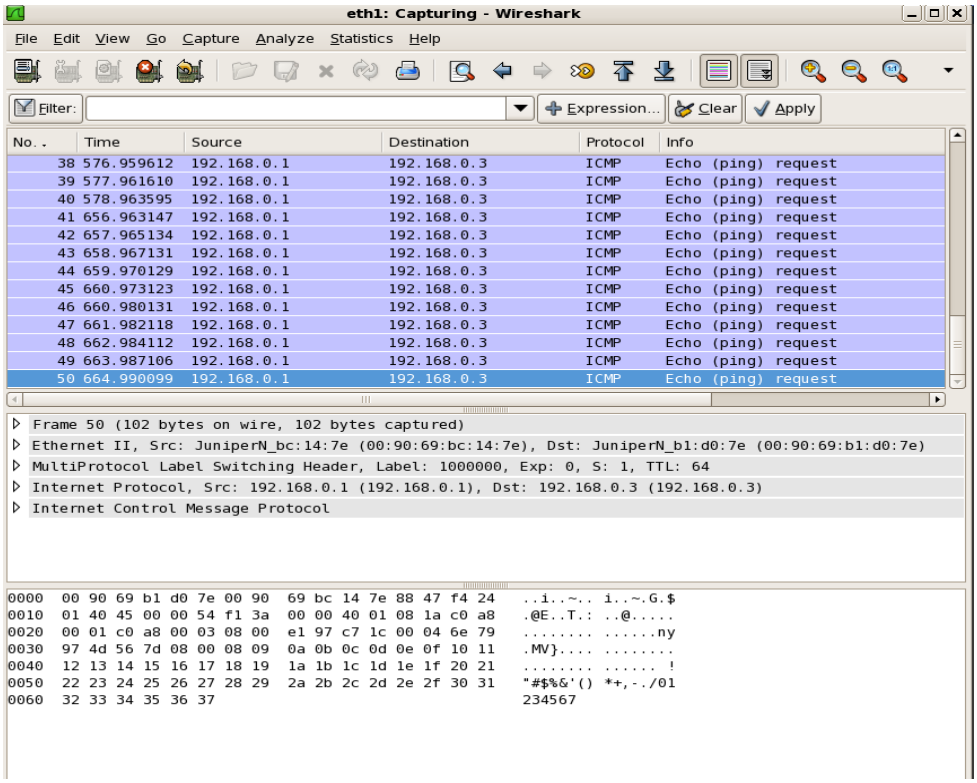
./lsr lsr_init

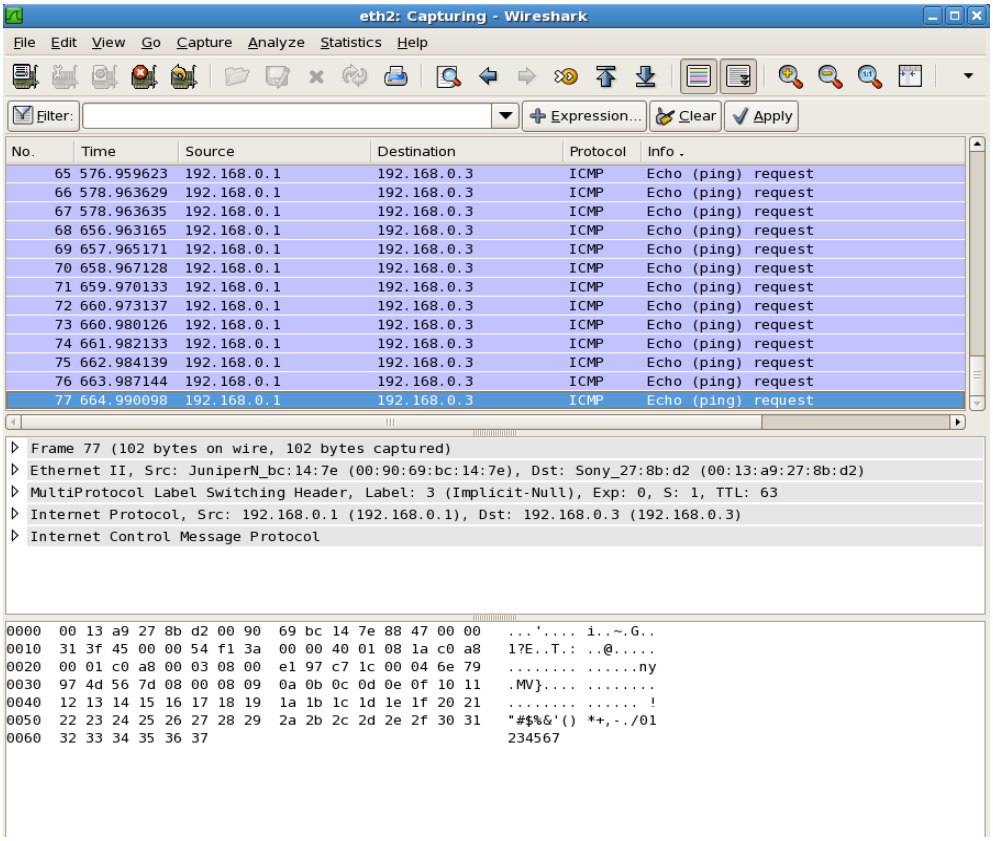
./lsr swap 2 0 3 3

./lsr mac_out 13a9278bd2 3

Then using the tcpreplay utility, run the following command to transmit the captured packets:

tcpreplay intf1=eth1 preswap.cap


eth1: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
38 576.959612 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
39 577.961610 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
40 578.963595 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
41 656.963147 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
42 657.965134 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
43 658.967131 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
44 659.970129 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
45 660.973123 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
46 660.980131 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
47 661.982118 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
48 662.984112 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
49 663.987106 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
50 664.990099 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
Frame 50 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: JuniperN_bc:14:7e (00:90:69:bc:14:7e), Dst: JuniperN_b1:d0:7e (00:90:69:b1:d0:7e)
MultiProtocol Label Switching Header, Label: 1000000, Exp: 0, S: 1, TTL: 64
Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.3 (192.168.0.3)
Internet Control Message Protocol
0000 00 90 69 b1 d0 7e 00 90 69 bc 14 7e 88 47 f4 24 ..i..~.. i..~.G.$
0010 01 40 45 00 00 54 f1 3a 00 00 40 01 08 1a c0 a8 .@E..T.: ..@.....
0020 00 01 c0 a8 00 03 08 00 e1 97 c7 1c 00 04 6e 79 ....ny
0030 97 4d 56 7d 08 00 08 09 0a 0b 0c 0d 0e 0f 10 11 .MV}.... .....
0040 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 .... !
0050 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 "#$%&'() *+,-./01
0060 32 33 34 35 36 37 234567


eth2: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
65 576.959623 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
66 578.963629 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
67 578.963635 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
68 656.963165 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
69 657.965171 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
70 658.967128 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
71 659.970133 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
72 660.973137 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
73 660.980126 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
74 661.982133 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
75 662.984139 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
76 663.987144 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
77 664.990098 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
Frame 77 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: JuniperN_bc:14:7e (00:90:69:bc:14:7e), Dst: Sony_27:8b:d2 (00:13:a9:27:8b:d2)
MultiProtocol Label Switching Header, Label: 3 (Implicit-Null), Exp: 0, S: 1, TTL: 63
Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.3 (192.168.0.3)
Internet Control Message Protocol
0000 00 13 a9 27 8b d2 00 90 69 bc 14 7e 88 47 00 00 ...'.... i..~.G..
0010 31 3f 45 00 00 54 f1 3a 00 00 40 01 08 1a c0 a8 1?E..T.: ..@.....
0020 00 01 c0 a8 00 03 08 00 e1 97 c7 1c 00 04 6e 79 ....ny
0030 97 4d 56 7d 08 00 08 09 0a 0b 0c 0d 0e 0f 10 11 .MV}.... .....
0040 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 .... !
0050 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 "#$%&'() *+,-./01
0060 32 33 34 35 36 37 234567

Push operation

The following commands are used for setup:

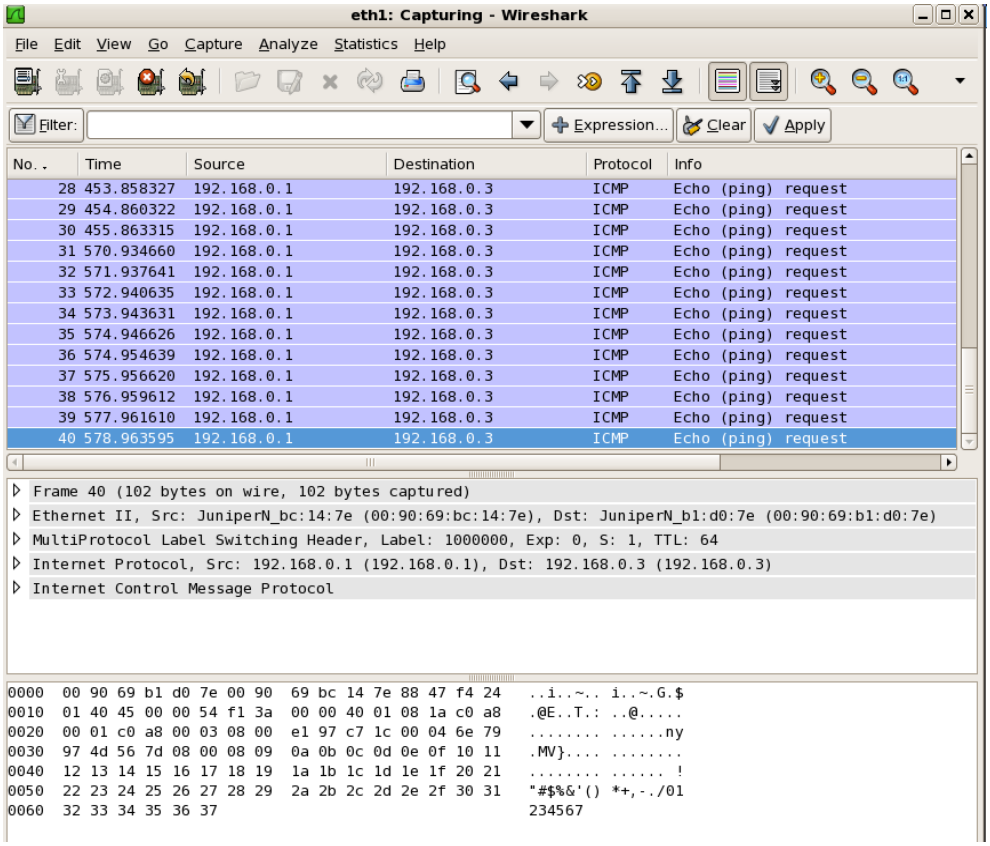
./lsr lsr_init

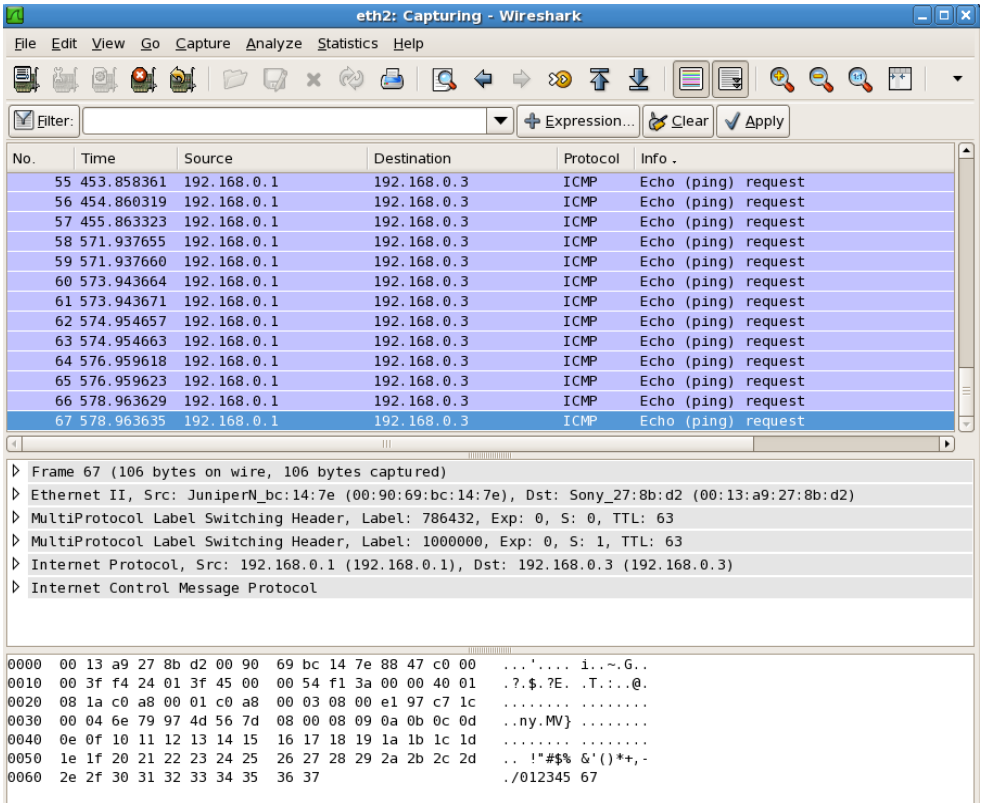
./lsr push 2 0 3 786432

./lsr mac_out 13a9278bd2 3

Then using the tcpreplay utility, run the following command to transmit the captured packets:

tcpreplay intf1=eth1 preswap.cap





Pop operation

The following commands are used for setup

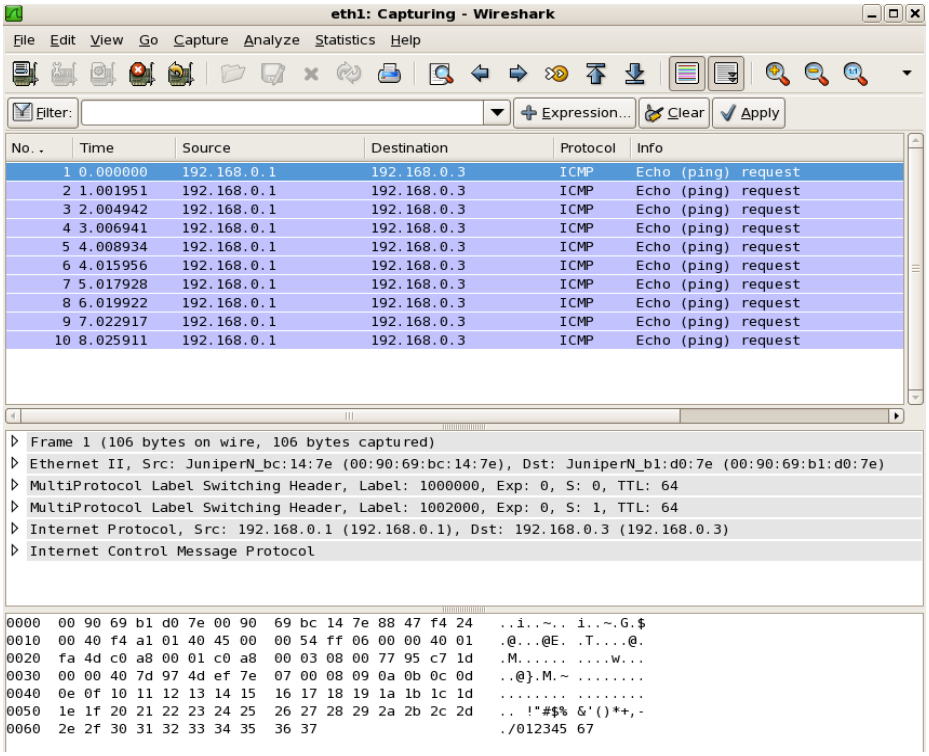
./lsr lsr_init

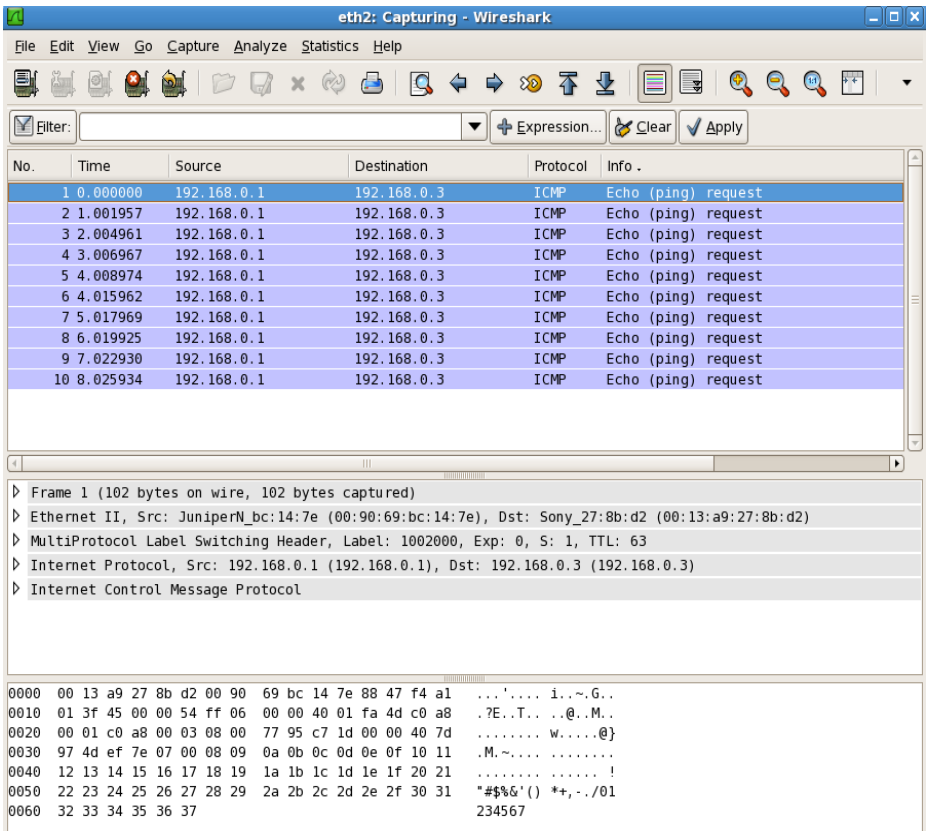
./lsr pop 2 0 3

./lsr mac_out 13a9278bd2 3

Then using the tcpreplay utility, run the following command to transmit the captured packets:

tcpreplay intf1=eth1 predoublepop.cap


eth1: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
2 1.001951 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
3 2.004942 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
4 3.006941 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
5 4.008934 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
6 4.015956 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
7 5.017928 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
8 6.019922 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
9 7.022917 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
10 8.025911 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
Frame 1 (106 bytes on wire, 106 bytes captured)
Ethernet II, Src: JuniperN_bc:14:7e (00:90:69:bc:14:7e), Dst: JuniperN_b1:d0:7e (00:90:69:b1:d0:7e)
MultiProtocol Label Switching Header, Label: 1000000, Exp: 0, S: 0, TTL: 64
MultiProtocol Label Switching Header, Label: 1002000, Exp: 0, S: 1, TTL: 64
Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.3 (192.168.0.3)
Internet Control Message Protocol
0000 00 90 69 b1 d0 7e 00 90 69 bc 14 7e 88 47 f4 24 ..i..~.. i..~.G.$
0010 00 40 f4 a1 01 40 45 00 00 54 ff 06 00 00 40 01 .@...@E. .T....@.
0020 fa 4d c0 a8 00 01 c0 a8 00 03 08 00 77 95 c7 1d .M...... ....w...
0030 00 00 40 7d 97 4d ef 7e 07 00 08 09 0a 0b 0c 0d ..@}.M.~ ....
0040 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d ....
0050 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d .. !"#$% &'()*+,-
0060 2e 2f 30 31 32 33 34 35 36 37 ./012345 67


eth2: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
1 0.000000 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
2 1.001957 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
3 2.004961 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
4 3.006967 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
5 4.008974 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
6 4.015962 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
7 5.017969 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
8 6.019925 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
9 7.022930 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
10 8.025934 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
Frame 1 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: JuniperN_bc:14:7e (00:90:69:bc:14:7e), Dst: Sony_27:8b:d2 (00:13:a9:27:8b:d2)
MultiProtocol Label Switching Header, Label: 1002000, Exp: 0, S: 1, TTL: 63
Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.3 (192.168.0.3)
Internet Control Message Protocol
0000 00 13 a9 27 8b d2 00 90 69 bc 14 7e 88 47 f4 a1 ...'.... i..~.G..
0010 01 3f 45 00 00 54 ff 06 00 00 40 01 fa 4d c0 a8 .?E..T.. ..@..M..
0020 00 01 c0 a8 00 03 08 00 77 95 c7 1d 00 00 40 7d .... w.....@}
0030 97 4d ef 7e 07 00 08 09 0a 0b 0c 0d 0e 0f 10 11 .M.~.... ....
0040 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 .... !
0050 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 "#$%&'() *+,-./01
0060 32 33 34 35 36 37 234567

Swap+push operation

The following commands are used for setup

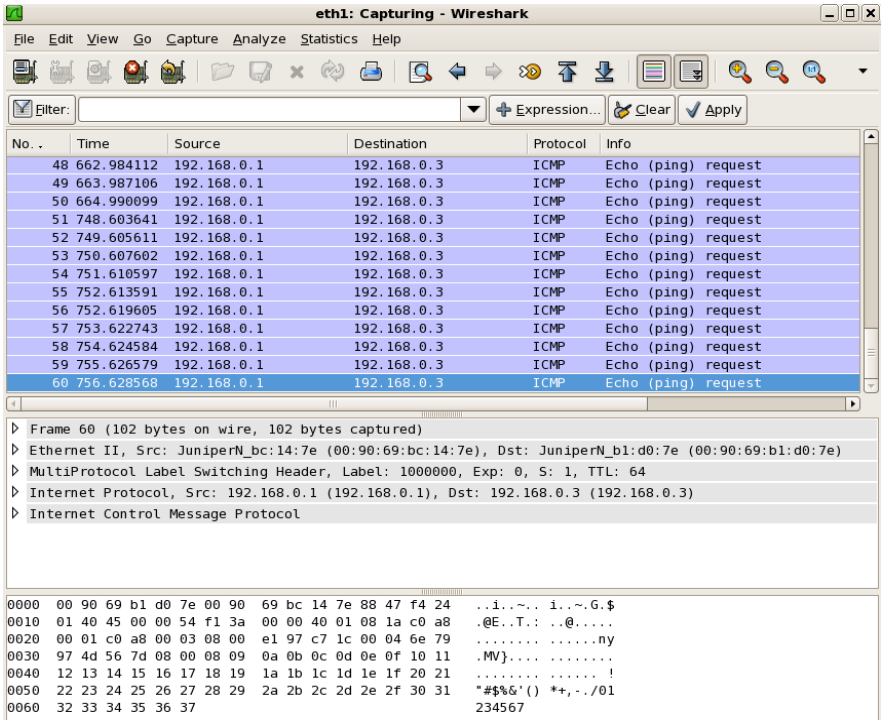
./lsr lsr_init

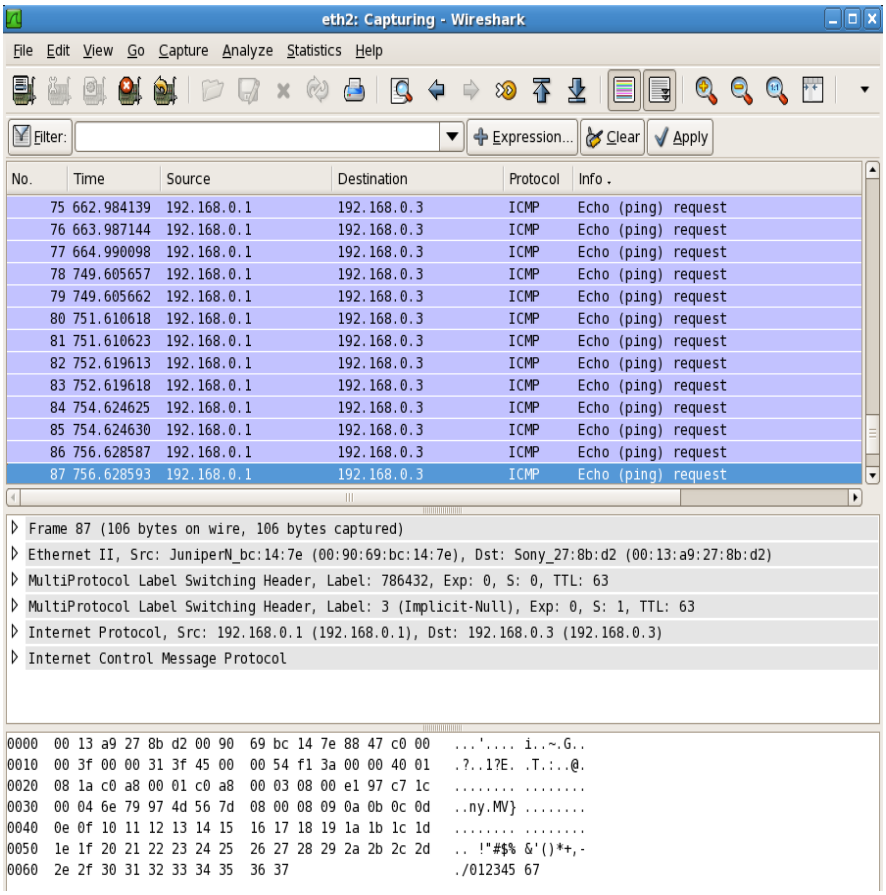
./lsr spush 2 0 3 3 786432

./lsr mac_out 13a9278bd2 3

Then using the tcpreplay utility, run the following command to transmit the captured packets:

tcpreplay intf1=eth1 preswap.cap





Pop+swap operation

The following commands are used for setup

./lsr lsr_init

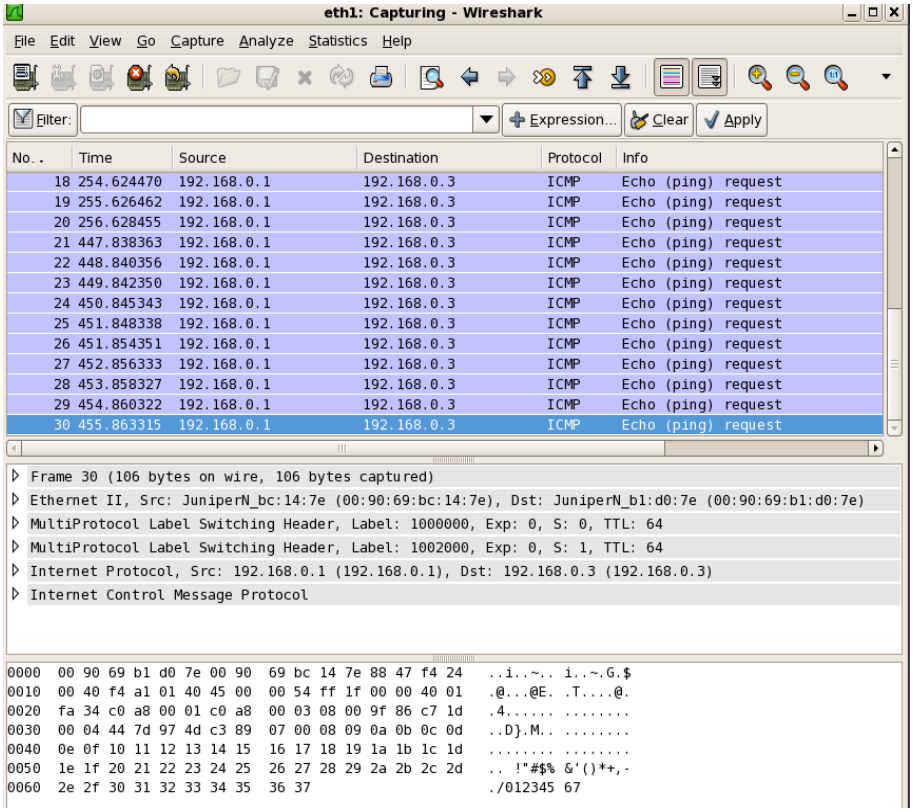
./lsr pswap 0

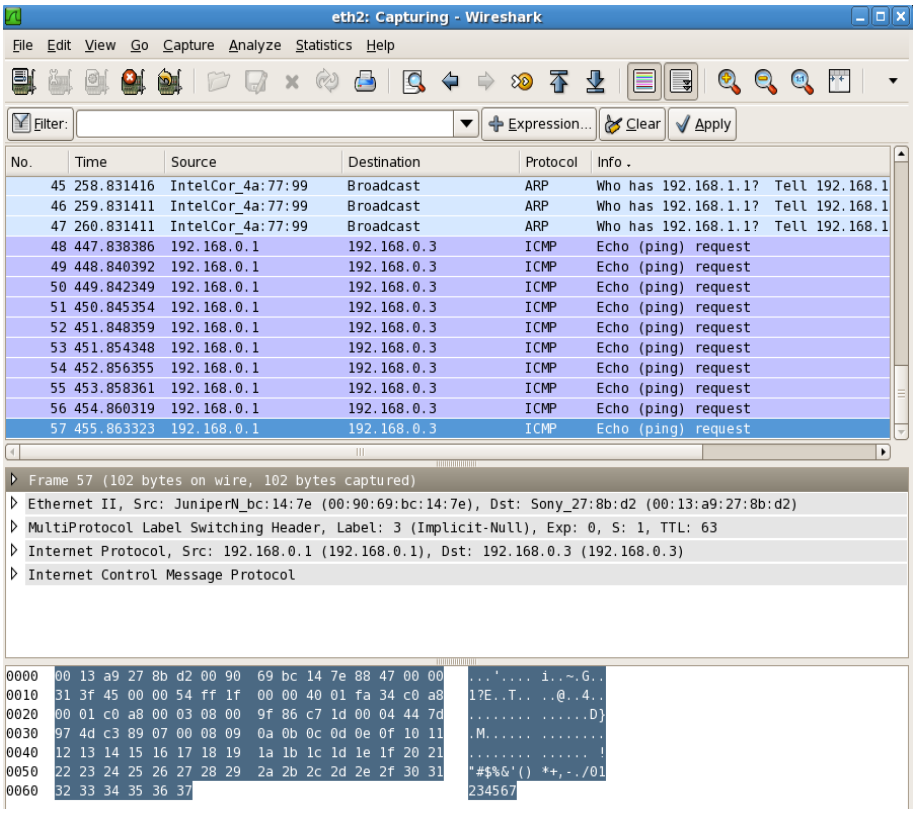
./lsr swap 2 2000 3 9

./lsr mac_out 13a9278bd2 3

Then using the tcpreplay utility, run the following command to transmit the captured packets:

tcpreplay intf1=eth1 predoublepop.cap


eth1: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
18 254.624470 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
19 255.626462 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
20 256.628455 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
21 447.838363 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
22 448.840356 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
23 449.842350 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
24 450.845343 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
25 451.848338 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
26 451.854351 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
27 452.856333 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
28 453.858327 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
29 454.860322 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
30 455.863315 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
Frame 30 (106 bytes on wire, 106 bytes captured)
Ethernet II, Src: JuniperN_bc:14:7e (00:90:69:bc:14:7e), Dst: JuniperN_b1:d0:7e (00:90:69:b1:d0:7e)
MultiProtocol Label Switching Header, Label: 1000000, Exp: 0, S: 0, TTL: 64
MultiProtocol Label Switching Header, Label: 1002000, Exp: 0, S: 1, TTL: 64
Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.3 (192.168.0.3)
Internet Control Message Protocol
0000 00 90 69 b1 d0 7e 00 90 69 bc 14 7e 88 47 f4 24 ..i..~.. i..~.G.$
0010 00 40 f4 a1 01 40 45 00 00 54 ff 1f 00 00 40 01 .@...@E. .T....@.
0020 fa 34 c0 a8 00 01 c0 a8 00 03 08 00 9f 86 c7 1d .4...... ....
0030 00 04 44 7d 97 4d c3 89 07 00 08 09 0a 0b 0c 0d ..D}.M.. ....
0040 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d ....
0050 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d .. !"#$% &'()*+,-
0060 2e 2f 30 31 32 33 34 35 36 37 ./012345 67


eth2: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
No. Time Source Destination Protocol Info
45 258.831416 IntelCor_4a:77:99 Broadcast ARP Who has 192.168.1.1? Tell 192.168.1
46 259.831411 IntelCor_4a:77:99 Broadcast ARP Who has 192.168.1.1? Tell 192.168.1
47 260.831411 IntelCor_4a:77:99 Broadcast ARP Who has 192.168.1.1? Tell 192.168.1
48 447.838386 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
49 448.840392 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
50 449.842349 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
51 450.845354 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
52 451.848359 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
53 451.854348 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
54 452.856355 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
55 453.858361 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
56 454.860319 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
57 455.863323 192.168.0.1 192.168.0.3 ICMP Echo (ping) request
Frame 57 (102 bytes on wire, 102 bytes captured)
Ethernet II, Src: JuniperN_bc:14:7e (00:90:69:bc:14:7e), Dst: Sony_27:8b:d2 (00:13:a9:27:8b:d2)
MultiProtocol Label Switching Header, Label: 3 (Implicit-Null), Exp: 0, S: 1, TTL: 63
Internet Protocol, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.3 (192.168.0.3)
Internet Control Message Protocol
0000 00 13 a9 27 8b d2 00 90 69 bc 14 7e 88 47 00 00 ...'.... i..~.G..
0010 31 3f 45 00 00 54 ff 1f 00 00 40 01 fa 34 c0 a8 1?E..T.. ..@..4..
0020 00 01 c0 a8 00 03 08 00 9f 86 c7 1d 00 04 44 7d ....D}
0030 97 4d c3 89 07 00 08 09 0a 0b 0c 0d 0e 0f 10 11 .M...... ....
0040 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 .... !
0050 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 "#$%&'() *+,-./01
0060 32 33 34 35 36 37 234567

11. Errata

Please note the following issues have been detecting in the existing bitfile, and must be worked around in software:

- The byte count is being done on frame ingress, however the byte counter is off by 4 (too low). For example a frame of 1000 bytes will be counted at 996 bytes.
- The TTL error count is off by a factor of 8. For example a single TTL error will be counted as 8.

Additionally, the following functionality issues should be noted:

- Decrementing MPLS TTL is not supported with PHP (encapsulated IP TTL will remain unchanged after the pop). Esstentially any time PHP is performed by the NetFPGA the packet's TTL will be restored to what it was before MPLS encapsulation took place.
- LD (Load Distribution) is not supported with the pop-swap opcode due to memory bandwidth constraints. It is supported with all other opcodes.

12. For Additional help and assistance:

Please contact: lsr_support@algo-logic.com