CS344

Build an Internet Router

Building an Internet Router

One of the main goals of this course is to design and build a fully functioning internet router. This document is meant to serve as a high level overview of approach that we will take to do this.

Tools

We will use the P4->NetFPGA Workflow to implement the data-plane portion of the router. This workflow attempts to abstract away many of the low level details required for hardware development. One of the goals is for it to be usable by P4 developers with no prior HDL design experience.

The control-plane will be written in Python on top of the Scapy packet processing library. Scapy is a very flexible packet generator, sniffer, parser, and editor; and it is quite easy to use, making developers more productive.

Approach

- Complete Getting Started deliverable to get everything that you will need set up.
- Read and understand the provided starter code, here is a brief overview.
- Review the protocols. Consider rewatching some of the CS144 lectures if you need a refresher on the protocols listed below.
  - IP - understand how a router makes a decision about forwarding an IP packet based on its routing table
  - ARP
  - ICMP * PWOSPF
- Learn how to use P4 and P4->NetFPGA:
  - P4 Mininet exercises

    - P4->NetFPGA exercises
- Review the basic requirements for the data-plane and control-plane (see below).
- Develop an interoperability plan early! As a class, you will want to decide on a concrete plan to make sure that everyone's router will end up interoperable. We strongly suggest that the teams work together to do incremental tests of their implementations against each other. For example, after implementing the PWOSPF HELLO protocol, ensure that inter-team routers can successfully perform neighbor discovery before going on to develop the link state updates. Testing components individually provides a much saner debugging environment than a fully built system.
- Divide and conquer. One member from each team will be responsible for the P4 data-plane implementation and the other will be responsible the control-plane. We encourage close collaboration and communication, but team members will need to work in parallel in order to complete the deliverables on time. So make sure to clearly define the interface between the data-plane and control-plane to ensure that integration will proceed smoothly. Clearly document your design decisions in your project's README file. The deliverables listed below are meant to provide a set of initial baseline tests to make sure your data-plane and control-plane implementations are on the right track, but you will still need to define more tests and integrate the two together. * Data-Plane baseline tests
    - Control-Plane baseline tests
- Integrate your data-plane and control-plane implementations. Make sure to run some tests on your own design before attempting to build a multi-router topology. See the hints below for some example tests.
- Interoperate with other teams! As a class, you will need to prove to the instructors that all of your routers are in fact interoperable.
- Celebrate!

Data-Plane Basic Requirements

- Provide a routing table that can store IP address/prefix pairs with their associated port and next-hop IP address.
- Use the routing table to perform a longest prefix match on destination IP addresses and return the appropriate egress port and next-hop address (or 0.0.0.0 for a directly attached destination).
    - NOTE: We will use a ternary match table for the routing table because LPM tables are not fully supported by SDNet yet.
- Provide an ARP table that can store at least 64 entries. This will accept an IP address as a search key and will return the associated MAC address (if found). This table is modified by the software, which runs its own ARP protocol.

- Provide a “local IP address table”. This will accept an IP address as a search key and will return a signal that indicates whether the correspond address was found. This table is used to identify IP addresses that should be forwarded to the CPU.
- Decode incoming IP packets and perform the operations required by a router. These include (but are not limited to):
  - verify that the existing checksum and TTL are valid
  - look up the next-hop port and IP address in the route table
  - look up the MAC address of the next-hop in the ARP table
  - set the src MAC address based on the port the packet is departing from
  - decrement TTL
  - calculate a new IP checksum
  - transmit the new packet via the appropriate egress port
  - local IP packets (destined for the router) should be sent to the software
  - PWOSPF packets should be sent to the software
  - packets for which no matching entry is found in the routing table should be sent to the software
  - any packets that the hardware cannot deal with should be forwarded to the CPU. (e.g. not Version 4 IP)
- Provide counters for the following:
  - IP packets
  - ARP packets
  - Packets forwarded to the control-plane

Control-Plane Basic Requirements

- Sending ARP requests
- Updating entries in the hardware ARP cache
- Timing out entries in the hardware ARP cache
- Queuing packets pending ARP replies
- Responding to ICMP echo requests
- Generating ICMP host unreachable packets
- Handling corrupted or otherwise incorrect IP packets
- Building the forwarding table via a dynamic routing protocol (PWOSPF)
- Support static routing table entries in addition to the routes computed by PWOSPF
- Handling all packets addressed directly to the router

You Decide

- Responding to ARP requests is actually fairly straight forward to express in P4. You can decide whether you want to implement ARP responding in the control-plane or the data-plane.

Hints and Tips

- Are you handling PWOSPF HELLO packets correctly in the data-plane? What IP address are they sent to?
- Be sure to make use of the CLI tool to add/remove/inspect table entries and read/write counters
- Here is useful command for configuring the tables in your P4 switch:
  - `cat ${P4_PROJECT_DIR}/src/commands.txt | ${P4_PROJECT_DIR}/sw/CLI/P4_SWITCH_CLI.py`
- Possible initial tests:
  - Is your router forwarding correctly with statically configured table entries?
  - Can you ping each of the routers interfaces?
  - Is the router responding to ARP requests?
  - Be careful if you are trying to ping one interface from the other. Unless you are careful, linux will force the traffic to use the loopback interface rather than sending packets out onto the wire. It is possible to do this, but it'll be easier (and less confusing) if you can arrange a time with a neighboring group to use their NIC. Then you can do small tests like sending pings through the router, traceroute to and through the router, send iperf flows through the router, and so on.
- Occassionally the `nf0` - `nf3` network interfaces do not come up after programming the FPGA. Try running the following command to bring it back up:
  - `# ifdown nf0 && ifup nf0`
- Configure interface with IP address:
  - `# ifconfig eth1 1.2.3.4 netmask 255.255.255.0`
- Configure interface with MAC address:
  - `# ifconfig eth1 hw ether 00:11:22:33:44:55`
- Adding routing table entries on Ubuntu:
  - `# route add -net 1.1.1.0 netmask 255.255.255.0 gw 12.12.12.13 dev eth2`
- Show routing table entries:
  - `# route -n`
- Show arp table entries:
  - `# arp -i eth1`
- If you try to program the FPGA and you see something like the following message: `Check programming FPGA or Reboot machine !`, that probably means that the machine has been shut off since the last time the FPGA was programmed. If the

links of the SUME board do not come up when the BIOS enumerates the PCIe endpoints then the SUME board will not be detected. The easiest solution to this problem is simply to do a warm reboot after programming the FPGA: `$ sudo reboot now`. Then try programming the FPGA again after the machine comes back up.

- If the `eth1` or `eth2` interfaces are down then you probably just need to configure them with an IP address using the `ifconfig` command as shown above.
- You can safely ignore the following error that you get when programming the FPGA: `rmmod: ERROR: Module sume_riffa is not currently loaded` because the programming script simply always attempts to unload and reload the `sume_riffa` drivers (even if they are not currently loaded).
- We recommend using VNC Viewer if you'd like a graphical desktop:
  - Install VNC Viewer if you don't already have it installed.
  - Start a VNC server on your development machine: `$ vncserver`. This command indicates the port on which the VNC server is running.
  - You can then use VNC Viewer to connect to your development machine on the appropriate display port. To connect to a VNC server running on port 1 of packet-3 connect to `packet-3:1` from within VNC Viewer.
  - You can start multiple vnc servers, in which case the port number would just increment.
  - To kill the VNC server runing on port 1: `$ vncserver -kill :1`
