CS 538: Advanced Computer Networks Spring 2017

Assignment 1: Experimental Data and Tools

Assignment 1 Due: 11:00 AM CT, Friday February 10, 2017

1 Introduction

The main goal of this assignment is to get hands-on experience with tools for experimental networking research. In addition, the assignment will hopefully remind you about some core networking material. Specifically we will get our hands dirty in two parts:

- Analysis of public Internet BGP routing traces (§2)
- OpenFlow functionality with Mininet, a network emulator (§3)

You may **choose one** of the above two parts to do for this assignment, according to your interests. We encourage you, however, to check them both out. We're providing them because they should be useful for research projects in this class, and checking out what experiments you *can* do may also give you ideas about what research you might *want* to do.

Data. The package is available for download at `https://courses.engr.illinois.edu/cs538/sp2017/assignments/a1.zip`. Inside, you will find two folders: `a1/bgp` for the BGP part (§2) and `a1/openflow` for the OpenFlow part (§3).

Submission instructions. This assignment is due at the time listed above. To submit your solutions, please email the TA (Mo Dong, modong2@illinois.edu) with a tar or zip file containing all relevant materials. **Please name the submission email and attachment file as "CS538-HW1-NetID"(Replace `<NetID>` with your NetID.) .** Acceptable formats are PDF (preferred), or plain text, with attached figures and code files. (Word .doc is not preferred. Please export to PDF.)

Collaboration policy. You're encouraged to discuss the assignment, solution strategies, and coding strategies with your classmates. However, your solution and submission must be coded and written yourself. Please see the policy on academic honesty and cheating stated in the course syllabus.

2 The Global Internet

On January 25, 2011, a popular uprising began in Egypt that would ultimately bring an end to the 29-year regime of Hosni Mubarak. On January 27, 2011, attempting to inhibit the Facebook- and Twitter-organized protests, the Egyptian government shut off essentially all Internet service to the country of 82 million people — a unique event in the history of the Internet.

Your mission is to answer the question, *How long did it take to sever all global networked communications of the 15th largest country in the world?* Because of the open and decentralized nature of the Internet, you can answer this question using publicly-available Internet routing information.

This document guides you through the process, including a sample parser for the BGP data. However, the main point of this section is to get a feel for the BGP data by visualizing an interesting event in the data set. If you would like to write your own parser rather than using ours, or explore an interesting aspect of the data other than what we suggest, or even explore a completely different event (there are many besides the Egyptian disconnection), you are welcome to do so. Check with us first if you have any doubts about appropriateness.

2.1 Getting started

The Route Views Project maintains data of routing behavior on the live Internet, and stores these traces for later analysis. Multiple years of data sets are available. To produce the data, Route Views maintains a number of *collectors*. Each collector has BGP connections to several ISPs' routers. The collectors log two types of data:

- occasional snapshots of the collector's entire routing table (Routing Information Base, or RIB); and
- continuous logs of the BGP update messages received from the neighboring routers.

You can download Route Views data from: `http://archive.routeviews.org/`

You will find a tool called `libbgpdump` under the folder `a1/bgp`. This tool processes both kinds of Route Views data; they both use the "MRT" format. Compile this by running `./configure` and then `make` in the `libbgpdump-1.4.99.11` directory. This builds a program called `bgpdump`, which, when given some input file (`./bgpdump routeviews_data_file`) will produce a human-readable text version of the data. Note the input file can be either a bzipped (`.bz2`) MRT file like you will get directly from the Route Views repository, or an un-bzipped MRT file. Take `bgpdump` and the Route Views data for a spin as follows:

1. From Route Views, download the first RIB snapshot from January 27, 2011, from the London Internet Exchange (LINX) collector.
2. Run `bgpdump` on the RIB snapshot. What does each field mean?
3. Search the `bgpdump` output to find any[1] RIB entry associated with your computer's public IP address. (Hint: you could do this by finding associated IP prefixes or AS numbers. You can use a whois database (e.g., `http://whois.arin.net/ui/`) to find the IP prefix and origin AS number associated with an IP address.)
4. Can you use the above entry to determine the sequence of ISPs through which packets will flow when traveling from one of the LINX routers in London to your personal computer?

What to submit:

1. Brief explanation of the "ORIGIN" attribute in BGP entry and how preference is decided among different origin types.

2. The sequence of ISPs (AS numbers and/or business names) from the last step.

[1] There may be multiple RIB entries for a single IP prefix, because each Route Views collector reports prefix advertisements received from multiple routers.

2.2 Measuring Egyptian route withdrawals

Next, we want to use the Route Views data to figure out how long it took Egypt to leave the Internet. To do this, we will use an imperfect but simple approach: We will count how many Egyptian-related prefix withdrawals we have seen over time. Due to the dynamic nature of the Internet, there are continually announcements and withdrawals even under normal conditions. But we'll see *one period of time with a very high rate of Egyptian withdrawals*, and that period will correspond to Egypt's disconnection from the Internet.

Although you're welcome to write your own parser if you prefer (or use the `libbgpdump` library), we have written an incomplete parser for you. In the folder `a1/bgp`, you'll find a program called `simple_bgp_parse.c`. This program expects to receive, on its standard input, the output of `bgpdump`. It scans through the BGP RIB entries or updates, and does two things. First, when it sees "interesting" entries, it remembers that the associated IP prefixes are interesting. Second, it keeps track of how many withdrawals of "interesting" prefixes it has seen, and prints out a running total.

What is "interesting" is a matter of opinion. The default version of the program is quite dull and thinks nothing is interesting. As described below, you will need to decide how to pick out the "interesting" entries, in order to learn which prefixes are associated with Egypt.

1. From Route Views, download the London Internet Exchange (LINX) collector's Updates data, from near the time of Egypt's departure from the net. That happened sometime between 21:00 and 23:00 UTC on January 27, 2011, so you'll want to grab all the LINX Updates data at least in that interval. Process these files with `bgpdump`, and send the output of all that to the (unmodified) `simple_bgp_parse` program. It should output the total number of updates seen. (For your own interest: In the data you downloaded, what time did the first and last update messages occur, and what was the average rate of updates per second during this period?)

   Tip: You might find the command `bzcat` useful. For new Unix users: to send multiple files to a program's standard input, you can run a command like: `cat file1 file2 file3 | my_program`.

2. Modify `simple_bgp_parse` so that it thinks BGP RIB entries for advertisements that *originated* at Egyptian ASes are "interesting". (Hint 1: this only takes a few lines of code. Hint 2: the following Autonomous System (AS) numbers belong to Egyptian ISPs: 5536, 8452, 24835, 24863, and 36992. Hint 3: Think about what part of the BGP RIB entry information you can use to figure out which advertisements were originated by Egyptian ASes.)

3. Now, run your modified `simple_bgp_parse`. This time, on standard input, feed it the output of `bgpdump` from the RIB file that you downloaded, concatenated with the output of `bgpdump` on the Update files. (The RIB output lets us learn which prefixes are interesting, and then we can count the occurrences of interesting withdrawals in the updates.) Note that the output of `bgpdump` from the RIB file is large (about 2 GB), so if you are running on university machines, you might want to put this in `/tmp` rather than in your home directory. And clean up when you're done.

   The output should now be a list of pairs of numbers; read the comment near the end of `simple_bgp_parse.c` for a description. Using that data, draw a plot showing *time* on the x axis, and *total number of Egyptian prefix withdrawals seen so far* on the y axis.

   You can use any plotting tool you want, but we have included an example of how to use Gnuplot in the package of code included with this problem set. Gnuplot is very useful and easy to get started with. In the `a1/gnuplot_example` directory, run the command `gnuplot example.gpl` which will produce `example.pdf`. You should be able to inspect `example.dat` and `example.gpl` and modify them to suit your purposes.

What to submit: Your plot from the last step. If everything worked, it should show a slowly increasing number of withdrawals seen, and then it should increase quickly for a period — a "withdrawal storm", corresponding to the disconnection of Egypt from the Internet – and then the rate of withdrawals should slow down again. Based on your plot, how long did that high-rate "withdrawal storm" last?

3 OpenFlow functionality with Mininet

Experimentation is an important part of networking research. However, large-scale experiments can sometimes be hard to achieve, e.g., due to lack of machines. In this section, you will learn how to use Mininet[2], a relatively new experimental platform that can scale to hundreds or more emulated "nodes" running on a single machine. Mininet takes advantage of Linux support for *network namespaces*[3] to virtualize the network on a single machine, so that different processes on the same machine can see their own network environments (like network interfaces, ARP tables, routing tables, etc.), distinct from other processes. Combined with the Mininet software, this enables a single machine to emulate a network of switches and hosts. The emulated processes, however, do see the same real/physical file system.

Mininet is designed with OpenFlow[4] in mind. In this exercise, you will gain a basic understanding of OpenFlow and create a custom OpenFlow controller to control your switches. Quite simply, OpenFlow allows for "programmable" network devices, e.g., switches. With Mininet, each switch will connect to the controller specified when the switch is launched. When the switch receives an Ethernet frame, it consults its forwarding table for what to do with the frame. If it cannot determine what to do with the frame, the switch sends the frame (and some extra information such as the input switch port) to the controller, which will then instruct the switch on what to do with the frame. To avoid this extra work on every such frame, the controller can install a new rule/match in the switch's forwarding table, so that the switch can forward future similar frames without having to contact the controller.

3.1 Prepare the Mininet VM and OpenFlow Controller

1. Install VirtualBox from `https://www.virtualbox.org/wiki/Downloads`. VMware should also work; adjust your VM configurations accordingly. (It is possible to install Mininet directly on your Linux system, but for simplicity we'll use the virtual machine here.)
2. Download and unzip the VM with Mininet already installed from `http://onlab.vicci.org/mininet-vm/mininet-2.2.1-150420-ubuntu-14.04-server-amd64.zip`
3. In VirtualBox, import the "ovf" template just unzipped. For the newly imported machine, go to "Settings" → "Network" and make "Adapter 1" a "NAT" (Network Address Translation). If your VM is allowed to obtain an IP address from your local network, you can alternatively use "Bridge Adapter." For more information on networking with VirtualBox, see `http://www.virtualbox.org/manual/ch06.html`
4. Start the VM
5. Log in with **mininet** for both username and password
6. Make sure `eth0` is up:
   (a) run the command:
   `ifconfig eth0`
   (b) check the `inet addr` field. If it does not have an IP address, then run the command:
   `sudo dhclient eth0`
   and repeat step (a).
7. The downloaded image should have POX preinstalled. POX is a platform that allows you to write your own OpenFlow controller using Python. Please check home folder and see if there is a folder called "pox". If not, please do:
   `git clone https://github.com/noxrepo/pox`
   For more information on POX, see `https://openflow.stanford.edu/display/ONL/POX+Wiki`

[2] `http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet`
[3] `http://lwn.net/Articles/219794/`
[4] `http://www.openflow.org/`

8. Install a GUI in the VM:
   (a) Install the GUI
   `sudo apt-get update`
   `sudo apt-get install openbox xinit -y`
   (b) Start it
   `startx`
   (c) To create a new terminal, right-click on the desktop and select "Terminal emulator"

   Alternately you may use SSH to log in to the VM remotely, with GUI (X11) forwarding. With SSH, you will need to enable X-forwarding (e.g., `ssh -X` on *NIX hosts) when you ssh into the VM. NOTE: this requires you have an X server running on the host. See a description of how to do this on various platforms at `http://www.openflow.org/wk/index.php/OpenFlow_Tutorial#Download_Files`. Alternative for some versions of Mac OS X: install the Developer Tools (a free download from the App Store) and open `/Applications/Utilities/X11`.

3.2 Create a hub in Mininet using POX

In this exercise you will create a Mininet network with 3 hosts connecting via a switch. Using POX, you will program the switch to behave like a hub, which simply forwards incoming packets to every port except the one on which it entered.

First, you can familiarize yourself with Mininet by following `http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/MininetWalkthrough`. To start Mininet with the topology we want:

- First clean up the network:
  `sudo mn -c`

- Then create a network with the topology we want:
  `sudo mn --topo single,3 --mac --switch ovsk --controller remote`

This will create a network with the following topology:

```
host h1 ------switch s1 ---- controller c0
host h2 -------/ /
host h3 --------/
```

After you create this network, you will be entering the Mininet console. You can type `help` in the console to see a list of commands provided by Mininet. We will later use some of these commands.

Now let's run POX controller. Create another terminal (right-click on the desktop and select "Terminal emulator"). Go to the directory you installed POX in this new terminal, and then start POX with basic hub function:

```
pox/pox.py log.level --DEBUG forwarding.hub
```

The argument `log.level --DEBUG` enables verbose logging and `forwarding.hub` asks POX to start the hub component. It takes up to 15 seconds for switches to connect to the controller. When a OpenFlow switch has connected, POX will print something like:

```
   INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:forwarding.hub:Hubifying 00-00-00-00-00-01
```

To verify the hub behavior, we use `tcpdump`, a common packet analyzer that intercepts and prints packet information. To do this, we first create an xterm (terminal emulator in X Window System) for each host in Mininet and view packets in each. To start an xterm for each host, type the following command in the Mininet console:

```
xterm h1 h2 h3
```

You may want to arrange xterms properly so that you can see them on the screen at once. You may need to reduce the terminal height to fit a laptop screen. In the xterms for h1 and h2, run `tcpdump` to capture and print all the packets:

```
tcpdump -XX -n -i h1-eth0
```

and

```
tcpdump -XX -n -i h2-eth0
```

In the xterm for h3, send a ping to h1:

```
ping -c1 10.0.0.1
```

The ping packets are going to the controller, which floods the packet out all interfaces but the received one. Because of this hub behavior, you should see identical ARP and ICMP packets in both xterms running tcpdump.

Question #1: What will happen if you ping a non-existent host that doesn't reply ICMP requests? For example, do the following command in the xterm for h3:

```
ping -c1 10.0.0.9
```

Submit and explain the results.

Now let's take a look at the hub code at `pox/pox/forwarding/hub.py`. Make sure to get familiar with the code because many POX API functions used here will help you answer the later questions. We describe several important API functions here, and you can find more information about POX APIs at `https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-POXAPIs`.

- `connection.send()` function sends an OpenFlow message to a switch.
  When the connection between a switch and the controller established, the code will invoke `_handle_ConnectionUp()` function that implements the hub logic.
- `ofp_flow_mod` OpenFlow message
  This tells a switch to install a flow entry, which matches some fields of incoming packet headers and executes some actions on matching packets. Important fields include:
  - `actions`: A list of actions that apply to matching packets (e.g., `ofp_action_output` described below).
  - `match`: An `ofp_match` object (described below).
  - `priority`: When a packet matches on more than one non-exact flow entry, only the highest priority entry will be used. Here, higher values are higher priority.

- `ofp_action_output` class
  This is an action for use with `of.ofp_flow_mod`. You can use it to assign a switch port that you want to send the packet out of. It can also take "special" port numbers, e.g., we use `OFPP_FLOOD` to send the packet out all ports but the received one.
- `ofp_match` class (not used in the hub code but is useful in the assignment) This is an object that specifies packet header fields and input port to match on. All fields here are optional, i.e., if you do not specify a field, it becomes a "wildcard" field and will match on anything. Some important objects in this class:
  - `dl_src`: The data link layer (MAC) source address
  - `dl_dst`: The data link layer (MAC) destination address
  - `in_port`: The packet input switch port

  Example to match packets with source MAC address 00:00:00:00:00:01 in a OpenFlow message `msg`:
  `msg.match.dl_src = EthAddr("00:00:00:00:00:01")`

3.3 Create a firewall

A firewall is used as a barrier to protect networked computers by blocking the malicious network traffic generated by viruses and worms. In this assignment, you are asked to implement a data link layer firewall to block certain traffic.

To start this, you will find a skeleton class file at `a1/openflow/firewall.py`. This skeleton class is currently not blocking any traffic and you will need to modify this skeleton code to add your own logic later. To test the firewall, put the `firewall.py` in the `pox/pox/misc` directory and run the POX controller:

```
./pox.py log.level --DEBUG forwarding.hub misc.firewall
```

After the connection between the controller and the switch is established, we can verify the connectivity between all pairs of hosts by typing `pingall` in the Mininet console. Note that when ping cannot get through a pair of hosts, you need to wait for the timeout, which takes about 10 seconds.

Question #2: Modify the firewall (`firewall.py`) to block traffic with source MAC address 00:00:00:00:00:02 and destination MAC address 00:00:00:00:00:03. To show the result, you can use the command `pingall` and copy the output to your report. (Hint 1: this only takes a few lines of code. Hint 2: if you did not specify any action in a OpenFlow message, then matching packets will be dropped.)

What to submit:

1. Your completed `firewall.py`
2. Results and your interpretation/explanation for both questions.

Note:

1. To get your files off the VM, you can `scp` or `ftp` them to some other machine. Or you can install the GUI (instructions in PDF) and then "sudo apt-get install firefox" and then launch the GUI and use firefox to upload/email the files off the machine.