EN.601.419/EN.601.619: Cloud Computing Spring 2020

EN.601.419/EN.601.619 Assignment 1

Assignment 1 Due: 5pm ET, February 24, 2020

Goal. The main goal of this assignment is to get hands-on experience with a few tools for experimental research in cloud networking and the MapReduce programming paradigm.

Submission instructions. This assignment is due at the time listed above. Email your assignment to soudeh@cs.jhu.edu:

- Subject: [Cloud, Spring 2020] Assignment 1 Your Name
- Attachment format: (a) a PDF file for your answers to the questions and (b) your completed code as a **.zip** file.
- Attachment filenames: (a) $a1\_X$.pdf and (b) $a1\_X$.zip, where *X* is your name.

Collaboration policy. You're encouraged to discuss the assignment and solution strategies with your classmates. Your solution and submission, however, must be written by yourself, in your own words. *Your submission must clearly mention the names of **all** the students that you have discussed/worked with, as well as all the resources you have used!* Please see the policy on academic honesty and cheating stated in the course syllabus.

1 OpenFlow functionality with Mininet

Experimentation is an important part of networking research. However, large-scale cloud experiments can sometimes be hard to achieve, e.g., due to lack of machines. In this section, you will learn how to use Mininet[1], 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*[2] 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[3] 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.

[1] `http://mininet.org/`
[2] `http://lwn.net/Articles/219794/`
[3] `http://www.openflow.org/`

1.1 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`
   You can also download VM images with Minine from `https://github.com/mininet/mininet/releases`
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`
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 `https://github.com/mininet/openflow-tutorial/wiki/Installing-Required-Software`. Alternative for some versions of Mac OS X: install the Developer Tools (a free download from the App Store) and open `/Applications/Utilities/X11`.

1.2 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://mininet.org/walkthrough/`. 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.

- **[1 points] A.1.1** 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")`

1.3 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 `http://soudeh.net/teaching/cloud/spring_2020/files/a1/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
```

Note: You may need to use the layer 2 MAC learning instead of the hub, i.e., you can replace the command above with:

```
./pox.py log.level --DEBUG forwarding.l2_learning 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.

- **[1 points] A.1.2** 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.)

1.4 What to turn in

Your submission should comprise two parts: 1) a part in your PDF document that answers the aforementioned questions (2 points) and 2) a **.zip** file that contains your source code (3 points).

Note: 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.

2 Playing with MapReduce

When discussing data analytic systems, we will see MapReduce as a paradigm for large-scale data processing. This part of the assignment provides practical experience writing MapReduce jobs.

2.1 Overview

The National Do Not Call Registry[4] is a national database of telephone numbers of individuals who do not want to be contacted by telemarketers. Unfortunately, robocalls and spoofing are on the rise, leading to a record number of complaints in recent years.

In an effort to stop unwanted calls, law enforcement recently seized evidence from a cloud provider that assists businesses in contacting their customers.[5] In the excitement of the raid, however, service metadata about the records and cloud consumers was damaged, leaving only portions of logs recording phone calls. Your goal is to uncover which cloud consumers are violating the law!

Fragments of the original call logs have been partially pieced together. Each log entry is a single line that provides the following information:

1. the date and time of a call,
2. the company responsible for the call,
3. the originating phone number for the call,
4. the recipient's phone number, and
5. the duration of the call (in seconds).

[4] `https://www.donotcall.gov/`

[5] The events that follow are fictitious. Any similarity to real life is purely incidental.

For example, the log entry indicates that the Acme Corporation used (429) 785-4094 to place a 9-second call to (429) 826-1640 slightly before 6 p.m. on 9 April 2017. Of course, this information alone is insufficient to determine if this phone call is legitimate: one must know that 1) the first telephone number has been reported for spam calls or 2) the second telephone number is part of the Do Not Call Registry and reported this specific call as unwanted.

Law enforcement has identified that the following numbers reported unwanted calls during the time frame captured in the call log: (216) 684-9356, (404) 934-5110, (589) 371-5037, and (945) 792-0329. You may assume that *all* calls that appear in the log and were placed to these numbers are violations of the Do Not Call Registry. That is, no business contacted these numbers for legitimate reasons such as an order confirmation.

As part of your forensics investigation, you must answer the following questions:

- **A.2.1** On how many occasions did companies violate the Do Not Call Registry?
- **A.2.2** How many numbers should be blocked / marked as spam to reduce the number of unwanted calls?
- **A.2.3** Which telephone numbers received the most spam calls?
- **A.2.4** Which telephone numbers are responsible for the most spam calls?
- **A.2.5** Which hours of the day are spam calls most likely?

Although you technically need not use MapReduce to answer these questions, large-scale data analysis practically necessitates a parallel processing framework.

2.2 What to turn in

You may implement the MapReduce jobs using any programming language. For simplicity, consider using a scripting language (e.g., Python) and Hadoop's streaming to facilitate testing your jobs.

For testing your implementation and answering questions, you can use the data file at `http://soudeh.net/teaching/cloud/spring_2020/files/a1/mr.data`.

Your submission should comprise two parts: 1) a part in your PDF document that answers the aforementioned questions and uses the guidance that follows for forming your responses to them (5 points) and 2) a **.zip** file that contains your source code (10 points). The source code archive should include all code used to answer each question, with the source code for each question in a separate directory named (01, 02, …). That is, the root directory of the archive should contain a subdirectory for each question and each subdirectory could include all source code (i.e., implementation of the MapReduce job) used to answer that question. This archive must also include a script called "runall.sh" which will run all of your mapping and reducing jobs.

Please note that answering some questions may require post-processing of the MapReduce results (e.g., extracting only the top-3 hours that had the most spam calls). You are not required to submit any code used for such post-processing, as it is assumed that you can perform this step manually.

Specific guidance for answering each question follows.

A.2.1. On how many occasions did companies violate the Do Not Call Registry? For each company that violated the Do Not Call Registry list the number of known unwanted calls placed by that company. That is, how many times did each company contact one of the numbers that reported unwanted calls? Order your results lexicographically by company (i.e., alphabetically by company name).

Your answer should resemble the following:

Acme Corporation 4
...

A.2.2. How many numbers should be blocked / marked as spam to reduce the number of unwanted calls? What is the total number of telephone numbers that should be blocked for each company? These telephone numbers should be *all* numbers used by the companies guilty of violating the Do Not Call Registry. That is, if the company violated the Do Not Call Registry once, then assume that all its calls should be marked as spam. Order your results lexicographically by company (i.e., alphabetically by company name).

Your answer should resemble the following:

Acme Corporation 6411
...

A.2.3. Which telephone numbers received the most spam calls? List the top-3 telephone numbers receiving spam calls and how many spam calls each received. Order your results in decreasing order of the telephone number receiving the most calls.

Your answer should resemble the following:

(847) 580-3060 18
...

A.2.4. Which telephone numbers are responsible for the most spam calls? List the top-3 telephone numbers placing spam calls and how many calls originated from each. Order your results in decreasing order of the telephone number responsible for the most spam calls.

Your answer should resemble the following:

(202) 221-4130 77
...

A.2.5. Which hours of the day are spam calls most likely? List the top-3 hours that had the most spam calls and how many spam calls were placed in each hour. Order the results in decreasing order of the number of calls.

Your answer should resemble the following:

11 a.m. 3157
...

A Hadoop

Apache Hadoop[1] is an open source software framework for big data processing. It has several components, but the two most critical to this assignment are its implementation of MapReduce and the Hadoop Distributed File System (HDFS), which are based on MapReduce [1] and the Google File System (GFS) [2] respectively.

For simplicity in this assignment, we'll use Hadoop's streaming application programming interface (API) which allows the use of any executable script to define the `map` and `reduce` operations. The streaming API uses the standard input and output streams to pass information among jobs. More specifically, the `map` operation coverts lines of input (text-based and terminated by a line break) into a series of key-value pairs, one per line of output. After these key-value pairs are sorted (automatically by Hadoop), the `reduce` operation aggregates them to produce a final value for each unique key. By convention, the streaming API uses the first tab character on a line to delimit the key and value.

An advantage of the streaming API is that you can use command line utilities to test your `map` and `reduce` operations. For example, the following shell command executes two Python scripts using a (small) local data file:

```
cat /path/to/data | python map.py | sort | python reduce.py
```

where `cat` prints the specified data files on standard out, `map.py` defines the `map` operation, `sort` sorts the script's output in ascending order, and `reduce.py` defines the `reduce` operation. Of course, none of these steps are parallelized in this case, but Hadoop will perform the various operations in parallel when processing multiple data files.

B MapReduce

Writing a MapReduce job using the streaming API is straightforward. The canonical MapReduce example is counting words so we'll use it to illustrate the process.

[1] https://hadoop.apache.org/

As previously mentioned, the `map` operation reads input from standard in and outputs a series of key-value pairs, one per line. The following Python code implements this operation for counting words:

```
#!/usr/bin/env python

import sys


for line in sys.stdin:
    line = line.strip()  # remove leading and trailing whitespace

    # split line using whitespace as delimiters
    tokens = line.split()
    # iterate over tokens
    for token in tokens:
        print("{token}\t{count}".format(token=token, count=1))
```

That's it! This Python code outputs a stream of tokens with a '1' to indicate that each token was encountered once in the line of text. (If the same token appears multiple times, then it will be listed multiple times.) For example, the input

```
a man a plan a canal panama
```

becomes

```
a       ⇥1
man       ⇥1
a       ⇥1
plan        ⇥1
a       ⇥1
canal         ⇥1
panama        ⇥1
```

where ⇥ indicates a tab character (i.e., `\t`).

The `reduce` operation reads the key-value pairs and aggregates them to produce a final value for each key. Of course, its input must be sorted to produce the correct results. The following Python code implements this operation for counting words:

```
#!/usr/bin/env python

import sys


```

```
def emit(token, count):
    print('{token}\t{count}'.format(token=token, count=count))


previous = None

for line in sys.stdin:
    line = line.strip()  # remove leading and trailing whitespace

    token, count = line.split('\t', 1)  # split key-value pair
    try:
        count = int(count)
    except ValueError:
        continue

    if previous == token:
        total = total + count
    else:
        if previous:
            emit(previous, total)

        previous = token
        total = count

emit(token, total)
```

In essence, this script simply checks to see if the prior token is the same as the current token, incrementing the total count when they match and outputting the total when they differ.

Try writing these scripts and testing them as follows:

```
echo "a man a plan a canal panama" | map.py | sort | reduce.py
```

You should see the following result:

```
a       3
canal       1
man       1
panama       1
plan       1
```

(Note: Both scripts must be executable to invoke them in this fashion.)

References

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation*, volume 6 of *OSDI '04*, pages 10–10, 2004.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 29–43, 2003.