# Final Project: Simple Twitter

Due date: Last lab session of the quarter or my office hours during finals week. (You will work on it during weeks 8, 9, and 10 of the quarter.)

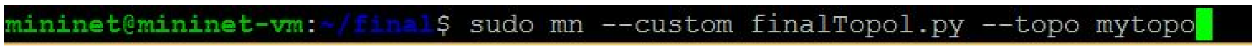
In this project you are going to implement a simple Twitter application. This application should have two parts, client and server. Users use clients to share tweets and see the tweets of the other users they subscribe to. The server is responsible for authenticating the users, getting their tweets from them, and sending those tweets to other users. To demo your work, you will run three clients and a server on an emulated virtual network in mininet.

## Setup the underlying network:

We will be using a virtual network to make the connection between a server and a total of three clients. For this purpose we are going to use Mininet. In Mininet you can define a set of hosts and use switches to connect them together. The whole network will be emulated by Mininet with Mininet allowing you to run different applications or commands on different hosts in your network. There are tools as well allowing you to investigate the network. For example, you can check the connections and available hosts in the network. To get familiar with the Mininet you can go to the following link which has a very good walkthrough for Mininet. (Reading this walkthrough is not mandatory but it can help you if you face problems in the next steps.)

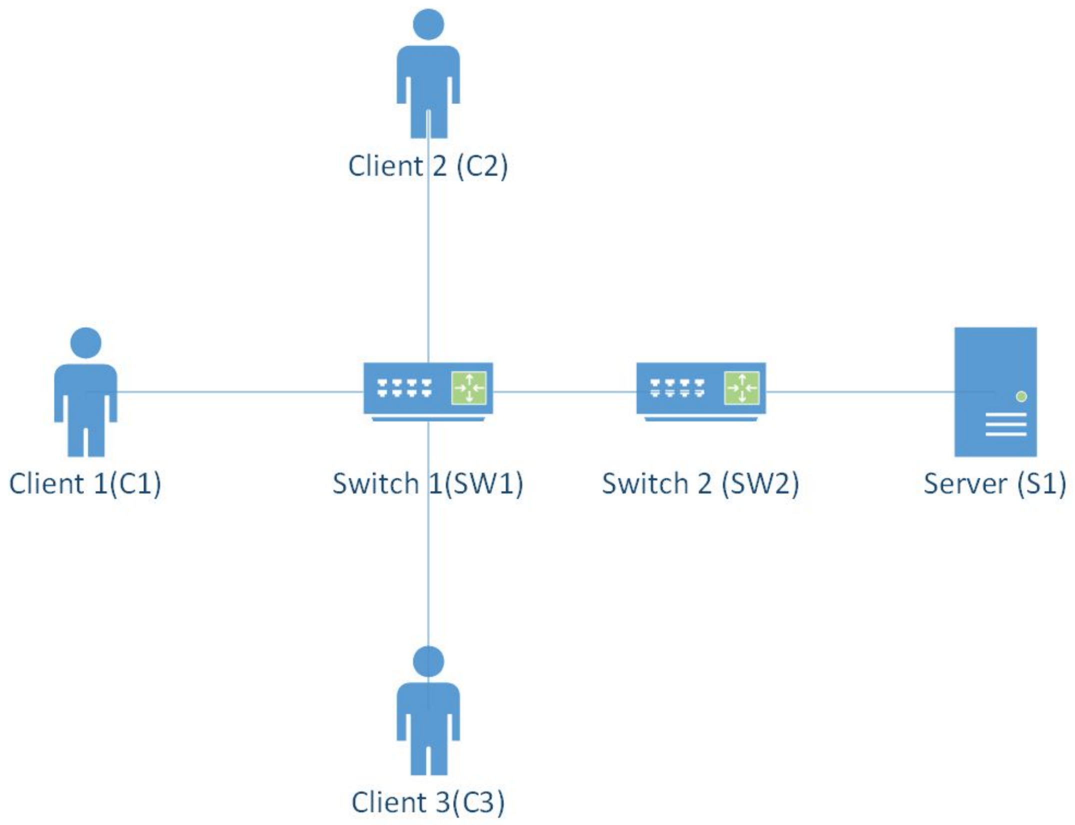
http://mininet.org/walkthrough/

One of the ways to define a custom topology in Mininet is to define the topology in Python and use the following command to run this topology

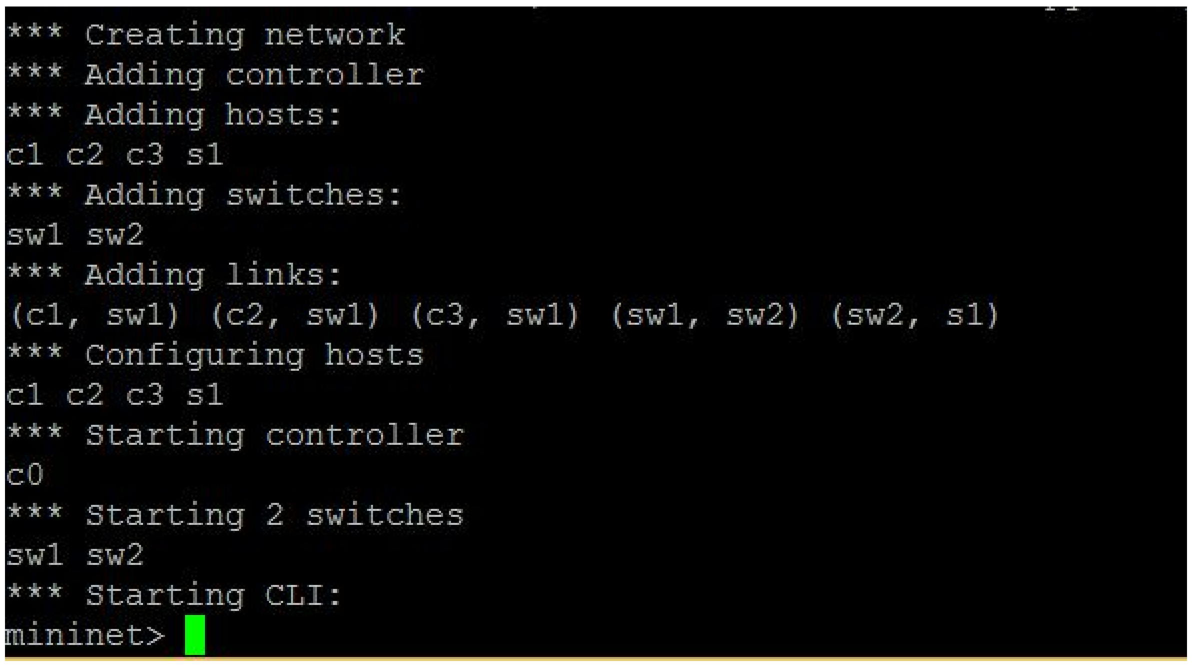


```
mininet@mininet-vm:~/final$ sudo mn --custom finalTopol.py --topo mytopo
```

In this command the finalTopol.py is your input custom topology file. To make it easier, we have provided the topology file for you. If you haven't already, you can download it from the same place you downloaded this document. (Please notice that this command is executed in the directory of finalTopol.py and that you should use the path of this file if you are in another directory). You can see the required underlying topology in the next figure.

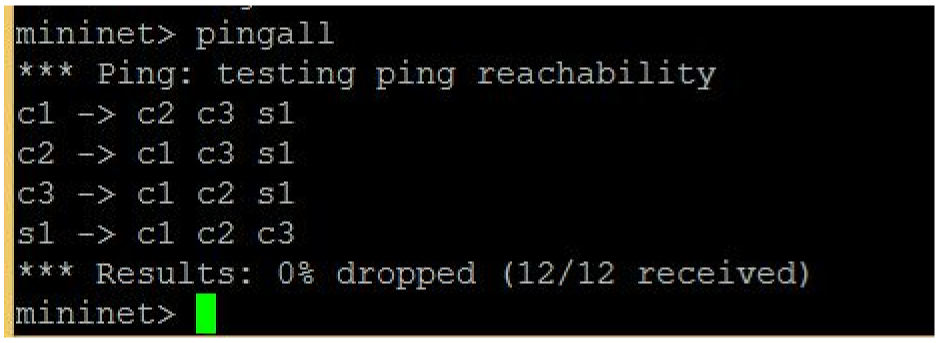


Before running this topology, please open finalTopol.py to familiarize yourself with how a topology can be defined simply in Mininet. (Note: there is a minor syntax error in the file that you will have to fix before you can use it. This is to make sure that you have at least opened the file.) Now run the aforementioned command to create your topology. After running this command you should see the following output from the Mininet interface:



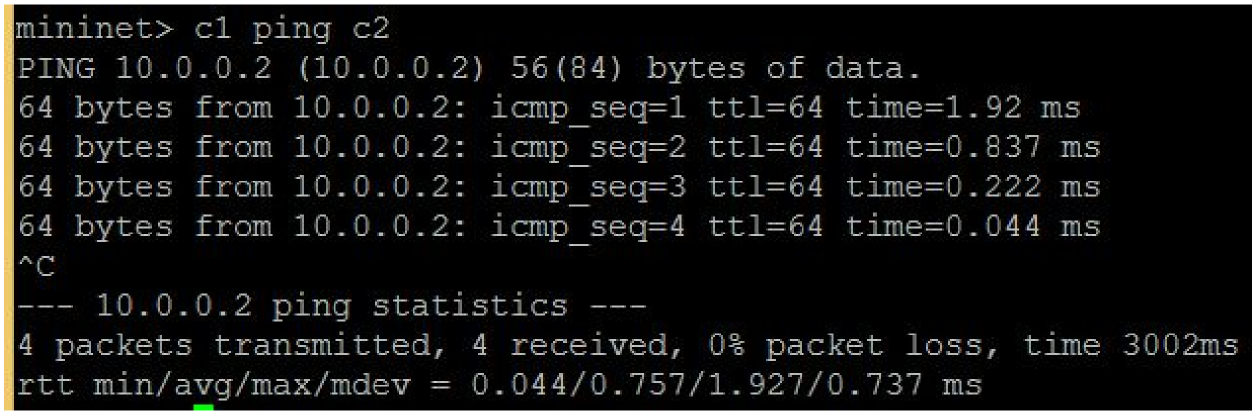
```
*** Creating network
*** Adding controller
*** Adding hosts:
c1 c2 c3 s1
*** Adding switches:
sw1 sw2
*** Adding links:
(c1, sw1) (c2, sw1) (c3, sw1) (sw1, sw2) (sw2, s1)
*** Configuring hosts
c1 c2 c3 s1
*** Starting controller
c0
*** Starting 2 switches
sw1 sw2
*** Starting CLI:
mininet>
```

To make sure everything is working, you can use the `pingall` command. The `pingall` command pings all the hosts in the network from all the other hosts. You should see output similar to the following:

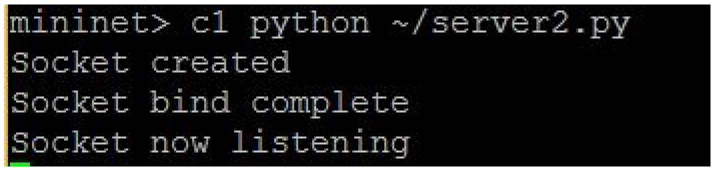


```
mininet> pingall
*** Ping: testing ping reachability
c1 -> c2 c3 s1
c2 -> c1 c3 s1
c3 -> c1 c2 s1
s1 -> c1 c2 c3
*** Results: 0% dropped (12/12 received)
mininet>
```

Executing a command on one of the hosts is very easy. You just simply use the name of that host at the beginning of the command to show that this command should be executed on that virtual host. For example, to ping client 2 from client 1, we need to use the following command:


```
mininet> c1 ping c2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.92 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.837 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.222 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.044 ms
^C
--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.044/0.757/1.927/0.737 ms
```

Not only can you use a command like `ping` on one of the virtual hosts, but you can also execute any applications on a host in a similar way. For example if you want to run the Python server code from previous labs, you can use the following command:


```
mininet> c1 python ~/server2.py
Socket created
Socket bind complete
Socket now listening
```

So you should be able to easily run your client and server code on the virtual hosts of Mininet. Starting out, however, we suggest that you develop your implementation on a single machine (just as we have done for the labs up until now). Later, try your implementation on the Mininet virtual network. This way, your coding and debugging will be simplified until your code is ready for more thorough testing.

Just like in our previous labs, your client and server will be command-line applications only. We recommend that you implement them using Python. While Python made socket programming easier than it was in C, if you would like to use a different method for establishing and maintaining your network connections that is acceptable. There are higher-level modules that will allow you to do this if you want to research and learn how to use them.

What follows are the specifications for the client and server programs you will be writing.

## Client Side:

The user experience should look like the following:

1) Prompt the user for their username and password. (10 points) **Note:** The password does not have to be obscured when the user enters it at the prompt and can be sent to the server as cleartext.

2) Provide a welcome message that displays the number of new messages generated by the current user's subscriptions, e.g., "You have 6 unread messages." (10 points)
3) Provide a menu that allows the user to select from their valid options. For each option there should be a way for the user to cancel their current option and go back to the menu at any time. (5 points)
4) Menu option: See Offline Messages (15 points)
    a) If the user selects this option, the client should ask if the user wants to see all messages or if they want to select messages from a particular subscription. If the user picks the latter, then you should provide a list of current subscriptions that the user can choose from. Once the user has made all their choices, the appropriate messages should be displayed. **Note:** This option is for viewing messages generated while the user was offline only. Once the user logs out, the current list of offline messages should be removed and replaced by any new messages generated while the user is offline.
5) Menu option: Edit Subscriptions (15 points)
    a) If the user selects this option, they should be given the choice to add a subscription or drop a subscription. If the user wants to add a subscription, prompt the user for the name and display a message indicating whether the name is valid or not. If the user wants to delete a subscription, give them a list of their current subscriptions to choose from.
6) Menu option: Post a Message (5 points)
    a) Allow the user to post a tweet-like message of 140 characters or less. Allow the user to enter in hashtags but do it by prompting the user separately for the hashtags they want to include, i.e, store the hashtags in a separate field so you do not have to parse a message for the hashtags. Verify the message to make sure it meets the length requirement. If it doesn't, notify the user and allow them to re-enter a different message (or cancel).
7) Menu option: Logout (5 points)
8) Menu option: Hashtag Search (15 points)
    a) A user can search for a hashtag and he will get last 10 tweets containing that hashtag.
9) While the user is logged on, make sure to display in real time any message sent by any of the user's subscriptions. (20 points)
10) (Optional) Menu option: See Followers (5 points)
    a) Display the users following this user.
11) (Optional) Improve the "Edit Subscriptions" option by allowing the user to subscribe to hashtags in addition to being able to subscribe to users. **Note:** The client does not have to validate a hashtag subscription. (10 points)
12) (Optional) Currently, all the tweets are public. As an optional feature, you can add the ability for users to decide whether a tweet should be public (broadcast to all subscribers) or private (sent to only certain specified users). (10 points)

13) (Optional) Make password entry more secure by hiding the characters typed at the prompt and/or encrypting the packets sent to the server that contain the password. (5 points)

Please notice that nothing is stored locally in the clients since all information is received from the server.

## Server Side:

The server should do everything necessary to support the client's functionality outlined above. It should also support some rudimentary server administration functionality. If there is no point value listed for a feature, it is because those points have already been accounted for in the client section above.

1) Validate a user login from a list of 3-5 hard-coded user accounts.
2) Maintain a list of each user's subscriptions (the users to whom they subscribe). Allow the client to change their subscriptions as they see fit, but make sure that only subscriptions to valid users are allowed.
3) Receive messages and redistribute them to the appropriate subscribers in real time.
4) Store a list of messages sent out but not delivered because the subscribers were offline. This is how we know how many new messages a user has when they log in, etc.
5) Allow an administrator to type "messagecount" in order to display the number of messages received since the server was activated. (5 points)
6) (Optional) Allow an administrator to type "usercount" in order to display the current number of users logged in. (5 points)
7) (Optional) Allow an administrator to type "storedcount" in order to display the number of messages that have been received but not yet delivered because the user is offline. (5 points)
8) (Optional) Allow an administrator to type "newuser" in order to permanently add a new username and password to the current list of user accounts. (10 points)
9) (Optional) Add support for subscribing to hashtags.
10) (Optional) Add support for decrypting any packets sent securely from the client (like the password).

## Extra Credit:

Extra credit will be given for completing optional features only if all the mandatory features are completed. Any extra credit you receive on this project can only make up for points lost in earlier labs. So, for instance, if you were getting 96% on your lab grade before the project and you do some of the extra credit features, your lab grade could go up to but not over 100%. As a reminder, you lab grade is 20% of your total grade for this class.

## Demo and Submission:

In order to demo your project, you will have to meet weekly milestones. The milestone for each week is to complete three of the nine required features for the client (and make sure the server is able to support those features). They could be any three you choose from week to week, but you must progress from completing a total of 3 in week 8 to completing a total of 6 in week 9, etc. There are some difficult features and some easy features, so feel free to be as strategic as you like with your choices. Each week you will demo your three features and then submit your code to iLearn. As with the previous labs, you have until Tuesday of the following week at 1:30 PM (my office hours are from 11:30 AM to 1:30 PM in WCH 110 on Tuesdays) to demo and submit code.

## Grading:

The lab is scored out of 150 points (see point values above). There's a total of 50 points of extra credit available. Attendance for each lab period is worth 10 points. Demoing your finished code on the mininet virtual network is worth 10 points. There are 5 points for scoring the quality of your code (whether you have bugs during your demo or not). If you fail to meet a milestone, you will be docked 20 points.

The project is worth 30% of your lab grade (so 6% of your overall grade).

Good luck!