CS 3251 - Fall 2018

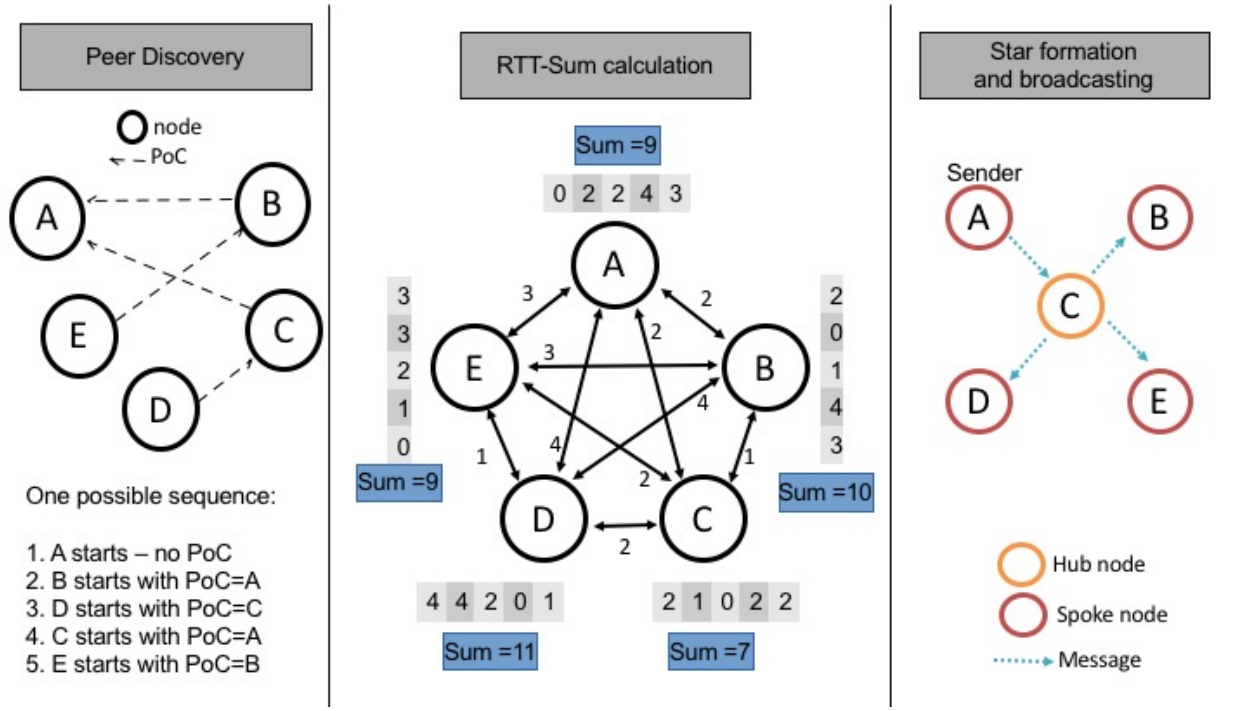
2nd programming assignment (a.k.a., "the project")

Reliable message broadcasting over a self-organized dynamic star network

Document version 1.0 (released Sept 28)

You can work on this assignment in groups of two students or individually. But you need to finalize this decision by the due date of the first milestone.

You can use C/C++/Python/Java. Please note that we will test your code on the CoC NetLab machines that run a special network emulator -- it is not sufficient if your code only works on your laptop.



1. Functional description and requirements

You are asked to design and implement a simplified peer-to-peer communication protocol that will be able to dynamically form a hub-and-spoke star network, and to perform reliable message broadcasts over that network. Let us refer to this as the ***"Star-Net"*** protocol.

Peer Discovery

Think of each peer (referred to as "star-node") as a process that will be running at one of our NetLab machines. One first task is to perform "Peer Discovery" – this term means that each active star-node should try to discover all other active star-nodes. This can be performed, in principle, if the peers exchange with each other the identity of all other active star-nodes they know (the "identity" of a star-node includes a name (an ASCII string) as well as the IP address and UDP port number of that process). When you run star-node X, it will initially know how to reach at most one other running star-node Y – we refer to Y as the "Point-of-Contact" (PoC)" of X. Y will be provided to X as a command-line argument. You will need to design the details of this peer discovery mechanism yourselves.

It is possible that when you run a star-node X, it does not have any PoC. This means that X can only be discovered by other star-nodes that include X as PoC – it cannot discover any star-nodes itself. For example, X may not have any PoC but Y and Z may have X as PoC.

It is also possible that even if X has a PoC (e.g., star-node Y), Y may not be running yet when X starts. This means that X will need to contact its PoC periodically (say every 5 seconds), until Y goes online. Of course, to avoid getting stuck in an infinite loop, this periodic probing process should terminate with an error message at X if Y does not respond within say 1-2 minutes after you start X.

Note that it may be impossible to discover all active star-nodes, depending on the given initial PoC of each star-node. You need to figure out the condition that the initial (peer, PoC) pairs should satisfy so that every star-node can discover all other star-nodes. You can assume that when we test your code, we will make sure that this condition is satisfied.

Churn and Keep-Alive mechanism

For simplicity, the *maximum* number of star-nodes (denoted by N) will be given as a command-line argument to every star-node. However, the number of active star-nodes at any point in time may be less than N because some of them may not run or they may be temporarily down.

Each star-node will need to periodically check if all other star-nodes it has discovered are still active. It is possible that a star-node may go offline for a randomly long period of time, and then come back online. This is referred to as "Churn" and it will be one of the most interesting aspects of this project.

You will need to design a "Keep Alive" (also known as "Heartbeat") mechanism, based on the exchange of periodic short messages between the active star-nodes, so that they can

automatically and quickly detect when one of them is offline. For example, this mechanism can be implemented with the exchange of periodic (say every 3 seconds) request-response messages between every pair of active star-nodes.

You can assume that when a star-node goes offline it stays in that state for at least 15 seconds, and when it comes back online it stays in that state for at least 1 minute.

When we bring a star-node down/offline, the corresponding process will be terminated with Cntr-C. This means that if a star-node comes back online it will not remember any of its previous state.

Round-Trip Time (RTT) measurements

Each star-node X will need to periodically measure its RTT with every other active star-node (say every 5 seconds). This (N-1)-dimensional vector is referred to as the RTT vector of star-node X.

The RTT between two star-nodes X and Y is the delay between sending a request from X to Y, and receiving a response from Y to X. In case the request or response is lost, you should repeat the RTT measurement.

Each star-node will be updating the sum of its RTT vector elements, say S(X), and send that sum to all other star-nodes. For example, star-node X will send the sum of its RTTs to star-nodes W, Y and Z, but not the individual RTTs.

Optimal network formation

The active star-nodes should compute the optimal star network that interconnects all of them in the following sense: the hub network (i.e., the center of the star) should be the star-node that has the minimum RTT sum value (the star-node X with minimum S(X)).

You should design a mechanism that avoids instability in the hub selection. For example, if there are two star-nodes X and Y that have equal (or almost equal) RTT sums, you should have a way to pick one of them as the hub, and to keep that hub selection despite small fluctuations in the RTT sums.

Reliable message broadcasting

A star-node should be able to reliably broadcast a message to all other active star-nodes through the hub.

For example, if X is the hub and Y is another star-node, Y can send a message to X, and then X will need to send the same message to all other active star-nodes. These message transmissions will need to be reliable however: if a message is lost by the network, the sender of the message will need to retransmit it until it gets successfully acknowledged.

We will be using a Network Emulator that will be introducing random packet losses and packet delays between the NetLab machines.

The Emulator can also duplicate packets.

If an ACK is not received within a certain Retransmission Timeout (that you will need to pick wisely), the packet is retransmitted by the Sender.

Some requirements about the Star protocol:

- You should not use TCP for the data transfer or for any other task in this project. You should only use UDP sockets.
- Your transport should be efficient. For example, a design that tries to deal with packet losses by transmitting each packet ten times would be a very bad design.
- The transferred messages may be anything, e.g., ASCII text, or any binary format such as JPG images.

A simplifying assumption about the transport protocol:

- The maximum size of the transferred message will be 64KB. This means that any message can fit in a single UDP datagram (i.e., you do not need to worry about breaking the message into different UDP packets).

2. Star interface

The star-node command-line should be as follows:

- `Command line:` **`star-node <name> <local-port> <PoC-address> <PoC-port> <N>`**

<name>: an ASCII string (Min: 1 character, Max: 16 characters) that names that star-node
<local-port>: the UDP port number that this star-node should use (for peer discovery)
<PoC-address>: the host-name of the PoC for this star-node. Set to 0 if this star-node does not have a PoC.
<PoC-port>: the UDP port number of the PoC for this star-node. Set to 0 if this star-node does not have a PoC.
<N>: the maximum number of star-nodes.

Example: star-node Alexa 23222 networklab3.cc.gatech.edu 13445 5

Star Commands
After a star-node starts running, it should interact with the user through a basic text-based interface. The interface should be able to recognize the following commands:

1. `Star-node command: send <message>`

This command will be executed by any star-node that wants to broadcast a message to others. The <message> may be an ASCII string in quotes or a file (see examples below). In either case, the message or file size should not be more than 64KB.

Example-1: send “Hey there. This is Alexa.”

Example-2: send image.jpg

When a message is received, that star-node should show the original Sender of that message (may not be the Hub). It should also show the message at the screen (if it is an ASCII message) or notify the user that a file has been received and report the corresponding filename.

2. Star-node command: show-status

This command should print (please use an easy-to-understand format!):
- the list of names of all other active star-nodes that this star-node currently knows, together with the latest RTT measurement to each of them
- the name of the currently selected hub star-node

3. Star-node command: disconnect

This command should gracefully terminate that star-node. It is necessary that that node contacts all other star-nodes, letting them know that it is going offline. If that star-node is the hub, a new hub will need to be chosen.

4. Star-node command: show-log

This command should print (please use an easy-to-understand format!) a timed log of all major events since the start of that star-node. This log will also be very useful for debugging your code. We suggest that you create a text file in which you append a short description every time one of the following events take place:
- when you first discover another star-node
- when you first discover that a previously active star-node is now down/offline
- when you send or receive an RTT request/response
- when you compute a new value of the RTT sum
- when you receive a new RTT sum from another star-node
- when you update the hub selection
- when you send a message
- when you forward a message (if you are the hub)
- when you receive a message
- when you decide to retransmit a message

- when you learn that another star-node disconnects
- when you disconnect

3. Milestone-1: Design Report (**due on October 11**)

The design report will need to describe at least the following:

- A description of your Star protocol.
- A description of the header structure for each type of packet that the Star protocol uses (e.g., the Keep-Alive request-response packets, RTT-sum packets, data messages, ACK packets, etc).
- Any relevant timing diagrams that would help illustrate the behavior of your protocol.
- Algorithms/pseudocode for any non-trivial Star protocol functions (e.g., how you decide whether a star-node is offline or how you decide how to select the hub node).
- Description of the key data structures you plan to use.
- Thread architecture: we highly recommend that you use thread programming for this project. If you do so, make sure that you design report identifies the threads that you will be using for each star-node process. For example, you may need to have a separate thread for exchanging and processing Keep-Alive packets.

Please note: we expect that your design will evolve over the course of this project – this is fine. However, it is necessary that you submit a complete design report by the October 11 due date. Your design report will be graded as “Successful” (100% credit) if it is submitted on time and it includes at least the previously mentioned sections. We will not grade it based on the details of your design.

At the end of the project, after you submit the final version of the code, we will ask you to resubmit your Design Report. The expectation is that that Report will describe the final version of your design.

4. The two programming milestones (**due November 1 and November 27**)

By milestone-2 (due November 1), you will need to submit a working version of your code. We will only test that version of your code however WITHOUT the following two sources of complexity:

a. No churn. Star-nodes never go offline.

b. The network emulator does not cause packet losses.

By milestone-3 (due November 27), you will submit the final version of your code, together with the final Design Report. We will test again your code at that point but including the previous sources of complexity.

5.Testing on an Unreliable Network

We have set up several physical machines to test your code. These machines are configured to delay packets, duplicate packets and to reduce the capacity of the network. This will allow you to test your implementation under adverse conditions. To access these machines, you need to be either on the Georgia Tech network (i.e., using GT machines or connected through the GT WiFi network) or using a Georgia Tech VPN client. Steps to install the VPN client are given by OIT (see http://anyc.vpn.gatech.edu). The process is lengthy and sometimes problematic so make sure you start early. Make sure to start and login to the VPN client every time you plan to use the remote machines.

In order to access these special machines, you need to *ssh* as follows:

```
ssh <gt_username>@networklabX.cc.gatech.edu
```

where X is an integer between 1 and 8.

Remember to use 130.207.107.*/127.0.0.1 as destination or source address (to listen on) in your code. For transferring your code to the remote machines, you may use *scp, sftp,* or the more user-friendly *filezilla*, which has a GUI.

We have used *netem* along with *tc* in order to setup adverse network conditions. If you feel more comfortable testing/debugging on your laptop, we show how to setup the network emulator at your laptop in Section-7 of this document.

6. Submission instructions

For the first milestone, you will only need to submit a PDF file – the Design Report. You do not need to submit any code at that point.

For the second and third milestones, please follow these instructions:

Please turn in well-documented source code, a README file, and a sample output file called sample.txt. The README file must contain :

- Your name (or names for group projects), email addresses, date and assignment title
- Names and descriptions of all files submitted
- Detailed instructions for compiling and running your programs
- Design Documentation as described in section 3

- Any known bugs or limitations of your program

You must submit your program files online. Create a ZIP/tar archive of your entire submission. Use T-Square to submit your complete submission package as an attachment.

An example submission may look like as follows -
pa2.zip
| -- **pa2/**
| -- **star-node.py**
| -- README.txt/.pdf
| -- sample.txt

We will use an automated script to test the code which may fail if you use a different naming convention.

Only one member of each group needs to submit the actual code and documentation. The other group members can submit a simple text file in which they mention their partner's name that has submitted the actual assignment.

7. Grading

The following table gives the maximum number of points for each component of this programming assignment.

Task	**Points**
Milestone-1: Design report	25
Milestone-2: Test peer discovery & RTT-sum exchanges	10
Milestone-2: Test star-network formation (no churn, no loss)	10
Milestone-2: Test message transfers (no churn, no loss)	15
Milestone-3: Test star-network formation (with churn & loss)	20
Milestone-3: Test message transfers (with churn & loss)	20

8. How to configure the network emulator at your laptop (only if you choose to do so – not required)

The goal of the following rules is to setup a network with the following network artifacts:
● Lost packets (e.g., 10% -- note that corrupted packets will be dropped at the UDP layer)

● Delayed packet delivery (we specify the mean and standard deviation of a Normal distribution)
● Lower transmission bandwidth (e.g., 1mbps)
Please note that we may modify these parameters when testing your code. Also, we may add packet duplication and re-ordering.

The tc commands that we execute on the remote servers, are as follows

```
#parameters
IF = eth0
SUBNET = 130.207.107.12/30
BW = 1mbit
CORRUPT_PCT = 10%
DELAY_MEAN = 100ms
DELAY_STD = 30ms
# tc commands
sudo tc - s qdisc ls dev $IF
sudo tc -s filter ls dev $IF
sudo tc qdisc del dev $IF root
sudo tc qdisc add dev $IF root handle 1 : htb
sudo tc filter add dev $IF parent 1 : protocol ip prio 1 u32 flowid 1 : 1 match ip dst $SUBNET
sudo tc class add dev $IF parent 1 : classid 1 : 1 htb rate $BW
sudo tc qdisc add dev $IF parent 1 : 1 handle 1 0: netem delay $DELAY_MEAN $DELAY_STD
distribution normal corrupt $CORRUPT_PCT
```

Rules for Local Host

The previous rules only work when the client and server run on different physical hosts. In order to test the program on your own computer, you will need to be running a Linux distribution (ubuntu, fedora, centos, mint) and execute these rules with the following parameters

```
IF = lo
SUBNET = 192.168.56.2/32 # your local IP address
```

Note that the IP that you use here should be used in your code as well. For example, if you decide to use 127.0.0.1 in your code, you should use 127.0.0.1/32 here. Do not use the IP 0.0.0.0 or *localhost* for testing purposes. Feel free to post on piazza if you face any issues.

References

http://www.linuxfoundation.org/collaborate/workgroups/networking/netem
http://onlinestatbook.com/2/calculators/normal_dist.html