Design Report

Andrew Le

Will be using Python 3.6.5 on Windows 10

Partner: Alex Chan

A description of our star protocol:

(this description covers the other parts of the report such as data structures used, algorithms, pthreads, etc with the exception to packet header structures and timing diagrams)

We will have a thread for each separate component of the star protocol. Meaning there will be a separate thread for peer discovery, rtt, rttsum, finding optimal hub, broadcasting messages, and churning. In addition to these threads, we will have 3 general threads: sending, receiving, and processing. How does this work? Each thread will have its own designated task. If that task requires to send a packet, the packet will be put into a send queue. The sending thread will get the head of the queue and send the packet to its destination. The receiving thread will be always activate waiting for incoming packets. The receiving thread waits for incoming packets, it will place the in the receiving queue as soon as the packet as been received. The processing thread will do any processing needed for that designated task. Whether it is placing an ACK into the sending queue or computing other information, the processing thread oversees these tasks.

All packets will be structured as a dictionary to be JSON serializable. The structure consists of:

{packetType, payload, senderIP, senderPort, receiverIP, receiverPort}

The payload will vary depending on the packet type as well as its purpose.

Payload could contain intergers, floats, strings, or anything necessary. When the payload needs to hold more than just one variable, a dictionary with multiple elements will be used.

The program will have a string (String hub) that contains the name of the current hub in the starnet. Each time a new star is added to the starnet or a star goes offline from the starnet, a new hub will be calculated and the string will be renamed. Another string (String timed_log) will contain a description of events as well as the time it takes place and append it to a text file to debug.

There will also be a list of dictionaries data structure. This will contain a list of all the stars in the starnet mapped to the information of the stars such as its star-name, IP address, UDP port number, whether it is offline or online (0 or 1), RTT, if it's the hub or not, its PoC (can be 0 if it has no PoC), etc.

Example of data structure: [star1: (ip, port, poc) , star2: ip,port, poc) ..]

When a star is joined into the starnet, it will send its information (star-name, IP, UDP port, etc) to its PoC and its PoC will send this information to its PoC and so on until PoC is 0 (it has no PoC). This information is stored into the dictionary data structure. For the star that receives this information, if star information being received is a unique star in the dictionary, it will be added. The star with the most stars in its dictionary contains the actual number of stars in the starnet when you +1 (includes itself). The star with

the highest number will send its dictionary to every other star in the starnet so that each star has an updated connection of the whole starnet to send its information to.

When a hub receives a message from the sender, it will go through its dictionary of stars and send this message over to the other stars except for the sender. The stars that receive the message from the hub, will continue to send the message to its PoC and so on. If a message is received, an ACK will also be sent back. If not, then the star that has not received an ACK after a certain amount of time will resend its packet.

Every 3 seconds, each star will also send a message to its peer consisting of its own star-name, ip address, and port number. If the star with the greatest number of stars in its list sees a decrease in the number of messages received, it means that a star has gone offline. The data structure will then find this star in its list and mark it as 0 (or offline). There will be an int variable that keeps track of whether there is a response from this star. If it does not respond 5 times, it will be marked offline.

For the RTT, each star will send a message to its PoC and wait for a response back. It will do this for every other star and there will be a variable that keeps track of the sum of its RTT. This sum will also be set to its PoC. Another data structure is also used to keep a list of all the RTT sums that a star receives. Again, the star with the most RTT sums received will be the one that updates each RTT variable in each star in its starlist data structure. The star with the least RTT will be selected to be the hub and its variable will be marked. If there are multiple hubs with the same least RTT, the hub will be the first one in the list.

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):

The header structure for each packet will contain the sender's IP address and port number, the receiver's IP address and port number, number of packets message is broken into (although not needed because a datagram will be able to fit the whole message), packet length, and the type of packet. Each type of packet will have a different number. For instance, ACK packet can be a 1, RTT-sum packets can be a 2, keep alive request response packet can be a 3, and so on.

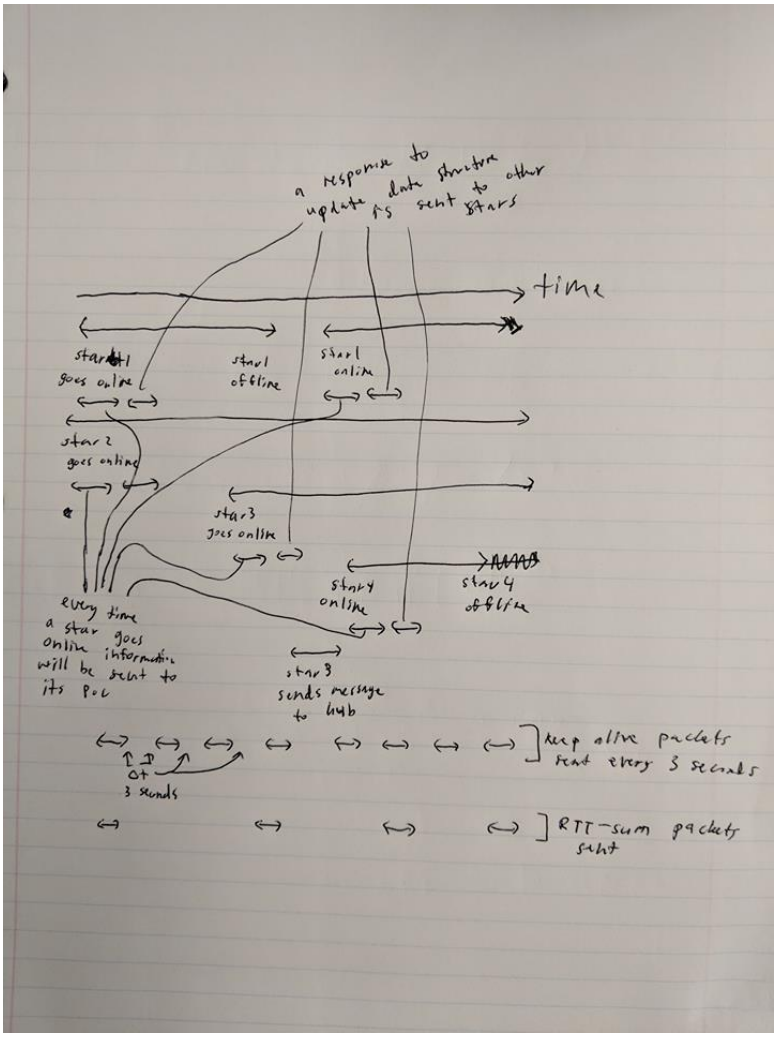
All packets will be structured as a dictionary to be JSON serializable. The structure consists of:

{packetType, payload, senderIP, senderPort, receiverIP, receiverPort}

The payload will vary depending on the packet type as well as its purpose.

Payload could contain intergers, floats, strings, or anything necessary. When the payload needs to hold more than just one variable, a dictionary with multiple elements will be used.

Timing diagram to help illustrate the behavior of the protocol:



Algorithm/pseudocode for any non-trivial star protocol functions (e.g. how you decide whether a star-node is offline or how to select a hub node):

When a star is joined into the starnet, it will send its information (star-name, IP, UDP port, etc) to its PoC and its PoC will send this information to its PoC and so on until PoC is 0 (it has no PoC). This information is stored into the dictionary data structure. For the star that receives this information, if star information being received is a unique star in the dictionary, it will be added. The star with the most stars in its dictionary contains the actual number of stars in the starnet when you +1 (includes itself). The star with the highest number will send its dictionary to every other star in the starnet so that each star has an updated connection of the whole starnet to send its information to.

Every 3 seconds, each star will also send a message to its peer consisting of its own star-name, ip address, and port number. If the star fails to receive an ACK back within 15 seconds, the star it sent to will be marked as offline in the peers list. There will be an int variable that keeps track of whether there is a response from this star. On each offline change, the star will send a packet of its updated peers list to all the other online and known stars.

For the RTT, each star will send a message to its PoC and wait for a response back. It will do this for every other star and there will be a variable that keeps track of the sum of its RTT. This sum will also be set to its PoC. Another data structure is also used to keep a list of all the RTT sums that a star receives. Again, the star with the most RTT sums received will be the one that updates each RTT variable in each star in its starlist data structure. The star with the least RTT will be selected to be the hub and its variable will be marked 1. If there are multiple hubs with the same least RTT, the hub will be the first one in the list.

Description of key data structures we plan to use:

There will also be a list of dictionaries data structure. This will contain a list of all the stars in the starnet mapped to the information of the stars such as its star-name, IP address, UDP port number, whether it is offline or online (0 or 1), RTT, if it's the hub or not, its PoC (can be 0 if it has no PoC), etc.

Example of data structure: [star1: (ip, port, poc) , star2: ip,port, poc) ..]

Another data structure is also used (arraylist of arrays) to keep a list of all the RTT sums that a star receives.

Thread architecture (threads we will be using for each star-node process (E.g. you may need a separate thread for exchanging and processing Keep-Alive packets):

We will have a thread for each separate component of the star protocol. Meaning there will be a separate thread for peer discovery, rtt, rttsum, finding optimal hub, broadcasting messages, and churning. In addition to these threads, we will have 3 general threads: sending, receiving, and processing. How does this work? Each thread will have its own designated task. If that task requires to send a packet, the packet will be put into a send queue. The sending thread will get the head of the queue and send the packet to its destination. The receiving thread will be always activate waiting for incoming packets. The receiving thread waits for incoming packets, it will place the in the receiving queue as soon as the packet as been received. The processing thread will do any processing needed for that designated task. Whether it is placing an ACK into the sending queue or computing other information, the processing thread oversees these tasks.