Network Virtualization Project 1 CD Report

Cheng Yuxin

Introduction of the Application:

This application has three main modules: Monitor Module; Member Module; Graph Module. By using different parameters in command line, the application will go into different mode, either monitor mode (-M), or member mode (-m monitor_name –v VID). The monitor node need to be pre-started before member node.

The monitor node will log all the event of the network, node join or node down and current vid-hostname mapping table. After all the member nodes are activated, the monitor will log the reaction time of the coming event.

The member node will first go in to bootstrapping phase. After all the neighbors are alive, the user can type “route” to check current routing table, or “route dst_vid” to check the route to the destination node. Also, the use can type “send size dst_vid” to send a packet to destination node. The size must be either 1 (1KB) or 10 (10 KB), the ideal latency time from routing table and the real latency time will return after the node receives the ACK from the destination node. The member will also log when there is any change in the routing table.

Modules:

Monitor Module:

The monitor module will first read from pre-defined topology file and store it in a graph class. Then it will create a TCP server socket to receive connections from the member node. When a connection is established, the monitor will check applied VID from the member to see if it is used. If not used, it will store the host name and vid in the mapping table. Note this connection is always established. If the connection is down, the monitor will consider the member down and begin the timer of the reaction time. This is one of the bottlenecks of this application, which will be mentioned in the later chapter. The monitor will only activate the reaction timer only after all the members finish the bootstrapping phase.

When there is a new node join in the overlay network, the monitor will first return all his neighbors hostname, and then notifies all his neighbors about his name.

Member Module:

The member module will first read from pre-defined topology file and store it in a graph class. Then it will create TCP client socket, connecting to the monitor. After the monitor return his neighbors name, it will store it in the mapping table, then start to broadcast hello packet periodically (2s) to all his neighbors using UDP and start a receiving TCP server socket that used for metric transmitting.

The hello packet contains a hello-sequence number. The original hello-request-sequence numbers are all even number starting from 0. After a node receives a hello-request packet, it will reset the sequence number to an odd number that is 1 larger than the request. Thus, the sender sends a hello packet with seq 0, the receiver will reply with seq 1. So the original sender will know the RTT time of every request hello packet. In this way, the hello packets are used as failure detection as well as latency counter in application layer. If there is no received udp packet from neighbor after 5s, the neighbor is considered down. This is another bottleneck of this application.

After all the neighbors are alive (that means receives the hello packet from all the neighbors), the node will broadcast the metric packet periodically (10s) to all his neighbors by TCP. The format of metric packet is “Metric seq start_node-end_node metric”. The sequence number is the identifier of the packet. The member node will only broadcast a receiving metric packet will a new seq number to his neighbors. The broadcast storm is avoided in this way. Also, since the metric format specifies the start node and end node of the metric, which might by slightly different in reverse, the graph is considered as a directed graph.

When a neighbor is detected as a down node, the member node will set the metric to 999 as unreachable, and broadcast to in the overlay network. When receiving a packet with metric 999, the end node in the packet will be considered as down, and the member node will tell the monitor about this event in order to calculate the reaction time.

When a new node joins, the other node will check to see if the previous metric is 999. If the previous metric is 999 and a new metric is not 999, the end node will be considered as up and the member node will tell the monitor to calculate the reaction time.

After received a metric packet, the member node will store the metric in his graph class, and then call the graph class function ‘dijkstra’ to calculate the open shortest route and metric. The user can type ‘route’ to see the current routing table.

For sending a packet to destination, the application uses source routing on UDP. The sender will create a header with “Data route” and add it to the data. The route specifies the best route of the sender. The intermedium node will follow the path to the next hop. After the receiver receives the data, it will create header with “Ack route”. The route in this header is same with the sender, not the receiver. The intermedium node will follow the Ack route and return it to the original sender. The sender then can calculate the real latency time.

The reason of using source routing is that since the application considered is as a directed graph and the user want to compare the real and ideal latency reaction time, only following the router from the sender is meaningful to compare to the ideal latency, since this ideal latency is calculated by the sender. If intermedium node uses his own best route, which might be different from the sender route, it will make no sense if the data or ack packet goes in another way.

Graph Module

The graph class will read the predefined topology.

The class uses two matrixes, one is metric matrix and one is sequence matrix. The insert function will set the metric in the metric matrix, only if the sequence is larger than the corresponding seq metric sequence, meaning it is a new metric. If anther node is considered unreachable, the metric will set to 999. If a node is consider as down, the sequence will reset to -1, and all the metric containing the node itself will be set to 999

The dijkstra function will calculate the best route of the given source id.

Real & Ideal Latency and Reaction Time

If the sender and receiver are neighbors, the real reaction time will be larger or smaller than ideal reaction latency, since the data packet can be consider as a type of “hello” packet. The performance is hard to tell. If the sender and receiver are not direct neighbors, the real reaction time is larger than ideal latency. This is due to the reason that every intermedium node need to parse the data header and follow the give path to the next hope. It will take extra time to parse the data and map the next hop vid to next hop hostname. Moreover, the 10KB packet takes

longer timer than 1KB packet to the destination.

When it comes to the join/down event reaction time from the monitor, the performance differs a lot. Since every time a new node join the overlay network, the monitor will tell it about all his neighbor and tell all his neighbor by TCP connection, and then the metric packets will be broadcast in the whole network, it usually only takes about 1 second that all the neighbors know the new node and specify to the monitor.

But if a node is down, the monitor won't tell the other member anything. It will take few seconds that the neighbor detect the failure and broadcast one by on. Since the failure detection timers are not synchronized, the total reaction time differs from 5 seconds to 10 seconds, depending on the numbers of the nodes in the overlay network.

Problems and bottle neck of the application

Since I did this project alone and it is really a large time consuming project, the application and the whole overlay network is not very robust. There are some known bugs as follows.

Since the monitor uses the TCP connection to test the member alive or not, and the connection will established all the time. Sometimes the member is not down but the connection to the monitor is somehow stopped, it will cause unknown bugs. This happens a lot when I try to write a bash to run 15 member nodes. I can see from the monitor that after first few concurrent TCP connections are established, some previous connection is down before the latter nodes join. So in order to finish the project, I do the test all manually, and only in 6 nodes. Really sorry about that, but I really tried my best on my own.

Another bottleneck is the failure detection. The plantnab nodes are not all reliable. Since the application uses hello packet in a timeout interval to check whether the other node is alive or not, sometimes the nodes will perform very strange in receiving and sending udp packet, and make the overlay network a mess. For example, one Asian node has 4 neighbors all in Europe and US, it make take very long time to receiving udp and the send the reply. Since the application uses a short hello_timeout, it always happen the wrong failure detection. And some plantnab nodes cannot handle to many packets in a short time and result in unpredictable bugs.