MP2 Report - CS425

Carson Sprague (cs104) & Rohan Kumar (rohandk2)

Overall Design:

To begin, we wanted to integrate this MP (failure detection) with the previous MP (log querying). Our log querier handled traffic by using a TCP connection, but (as well-defined in the spec) we decided to use UDP for the PingAck traffic. This was much more bandwidth efficient, as we weren't sending large amounts of data and didn't need the benefits of a TCP connection. We thought of it as a UDP layer underneath the TCP layer that would be responsible for failure detection. As to not get bogged down with traffic, we opened 10 UDP ports per VM (each being serviced by their own thread), which allowed each VM to send PingAcks via its own traffic-free port to one-another.

Swim PingAck Design:

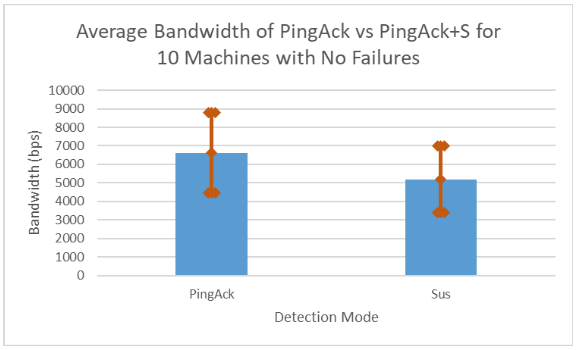
Next comes to the specifics of our SWIM protocol. To handle joins (and rejoins), we designed a dedicated introducer node. If this node is dead, no other nodes can join. This special node would receive all joins requests from new nodes, and send this information out to all the existing nodes in a multicast fashion. Next is our PingAck system. As SWIM guarantees deterministic failure detection, we implemented our ping system with a pseudo-random number generation method that would guarantee a worst case detection time of 2n-1. We then send an ack back to the sender, as described in the protocol. If there's a case in which we never receive an ACK back after a certain timeout, we mark the node as dead. With this approach, there is the chance of false-positives occuring (due to network latency or packet drops), but as given in the protocol, that is fine. To handle updates to the membership list, we would send the node's membership list along with the pings/acks in a piggyback fashion. We follow the priority given in the SWIM paper to update the membership lists. If we receive a newer timestamp/incarnation number, we update our list to match that entry. If the incarnation or timestamp is the same, a 'dead' status triumphs all, and a 'sus' state triumphs alive (read more below in the suspicion section). When it comes to failure detection, there are two ways a node can leave the system: voluntarily leave or crash. If a machine wants to leave, it sends that out to all nodes in its membership list via a multicast send. If a node crashes, a node will eventually ping it (in under 2n-1 time), and will not respond. At that point, the pinging node waits for a timeout, updates its list with the node marked dead, and disseminates this through the piggybacking method as mentioned earlier.

Swim PingAck+Suspicion Design:

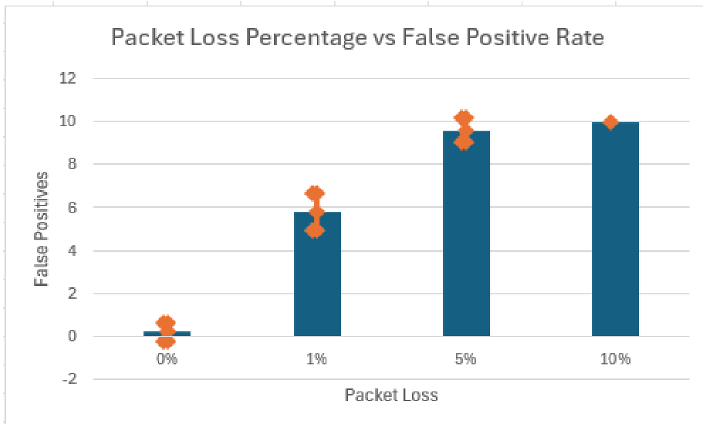
To layer suspicion into our PingAck system, we added a command that would dynamically enable suspicion. Instead of simply marking a node as dead when it didn't respond, we would mark the node as 'sus.' This is a higher priority state than alive, and would be disseminated throughout the nodes. If a 'sus' node received a membership with itself labeled as such, it would increase its timestamp/incarnation number and disseminate that to the other nodes, thereby removing its 'sus' status from the group.

Performance:

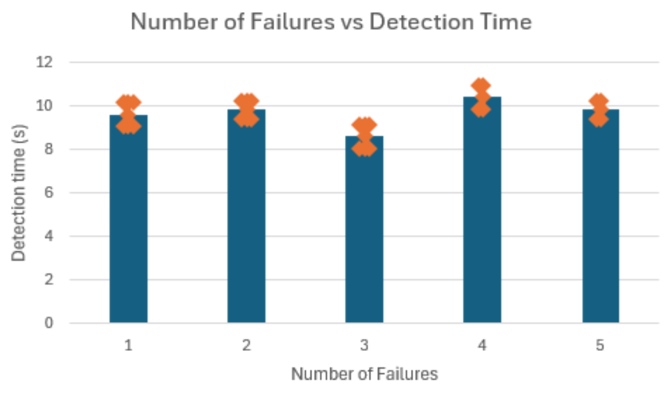
Our first measurement compared the bandwidth usage for Ping versus PingAck+S. Our design sends out the entire list of nodes every time, which is why our bandwidth is much higher than we expected. This made it easier in our design because we didn't have to have a dynamically sized membership list to send out.



Our second measurement compared the packet loss percentage vs the false positive rate. We observed a trend of the higher the packet loss, the higher the false positive rate. The standard deviations for each trial were low for each percentage of packet loss.



The graph below shows the detection time as a function of the number of failures. As you can see in the graph, our system holds the failure detection to around 10 seconds, only going over the worst case scenario once. The reason we are so close to the threshold is because we are sending large amounts of data per message, which can slow down our detection, even with a higher bandwidth. Because our bandwidth was so high, we were running into issues where we were clogging up a port, hence the large timeout time.


Number of Failures vs Detection Time
Detection time (s)
Number of Failures