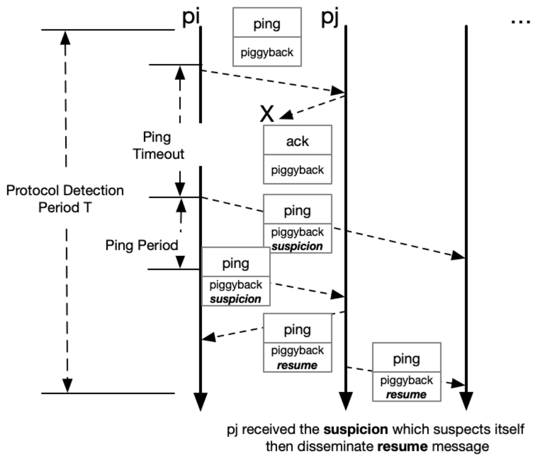
baohez2(Baohe Zhang) and kechenl3(Kechen Lu)

CS425 MP2 Report

Part I System Design:

Our distributed membership system implemented a SWIM-like protocol, which includes infection-style dissemination, piggyback on membership messages, ping-ack failure detection with suspicion mechanism.

For failure detection, in one ping period, each process p_i randomly select one process p_j in membership(other than itself) as the ping target. The target process p_j will be marked suspicious if the process p_i does not receive ack from p_j within the ping-ack timeout limit, then disseminating suspect message of process p_j to others. All other process (including p_i) would mark p_j failure if they do not receive the resume message within the suspect timeout limit. We guarantee the eventual completeness by implementing time-bounded completeness, shuffling the list in one traverse and traversing the list when selecting ping target. In addition, with suspicion mechanism, the false positive rate would reduce a lot. And suspicion mechanism shown as Fig.1.



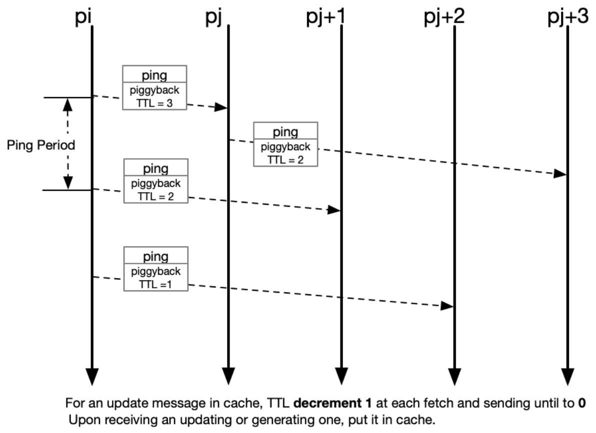

pi
pj
pj+1
pj+2
pj+3
ping
piggyback TTL = 3
Ping Period
ping
piggyback TTL = 2
ping
piggyback TTL = 2
ping
piggyback TTL =1
For an update message in cache, TTL decrement 1 at each fetch and sending until to 0
Upon receiving an updating or generating one, put it in cache.

Fig. 1 Failure Detection with Suspicion Mechanism

Fig. 2 Infection-style message dissemination

For message dissemination, all messages piggyback on the ping or ack packets. All membership update and initial messages generated by self are cached within $\lambda log(N)$ detection periods. Each period or when responding to others' ping, we get one update payload from the cache, and decrement its TTL count, and piggyback on a ping or ack packet. Other process who receives this message, caching and disseminating as well, the example shows as Fig.2. This mechanism acts like infection-style dissemination, which has $O(log(N))$ dissemination time w.h.p. .

For membership service, we have new process joining with "introducer", but process can leave without "introducer". The new process sends a request of joining group to the introducer which then inserts the new process id and state to its membership list, meantime generating a join update message, disseminating to others in the group. Also the introducer would reply with entire memberlist to this new joining process. When a process voluntarily leaves, this process would put this message in cache and then have the epidemic dissemination as well.

Our implementation can work in large N nodes group with fair scalability. We have constant load per ping period (each process target and ping one with one piggyback payload), almost constant $O(log(N))$ dissemination time (infection-style), time-bounded completeness within $O(log(N))$ periods with high probability, constant first failure detection time $\frac{1}{1-e^{-1}} T_{ping\ period} + T_{suspect\ timeout}$.

Each process runs as a daemon that includes packet sender and receiver running concurrently. Message packet encapsulates header and payload serialized in big-endian. Header has 3 fields, type, sequence number, and reserved field. There are 2 kinds of payload, member and update information. Member information is used for member joining, while update information used for updating states of members. Our protocol's marshaled message format list like below Fig.3.

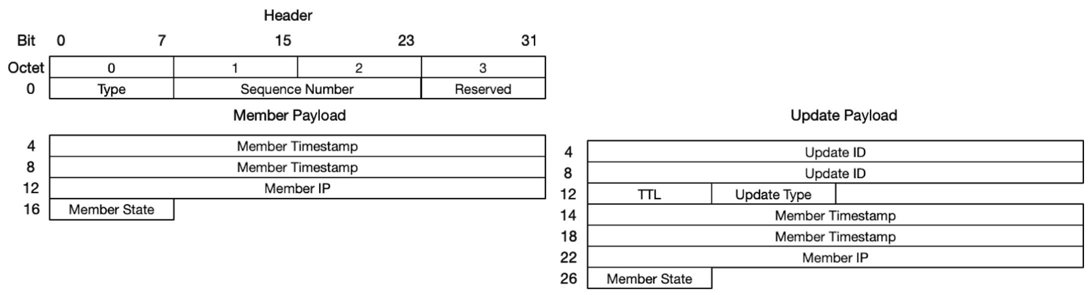


Fig. 3 Header and Update/Member Message Format

We debug the membership system using distributed grep which helps a lot to explore failure detection and correctness of ping-ack timeline sequences for each node, and then optimize our implementation and tune our parameters like ping period.

Part II Evaluation:

(i) To measure the background bandwidth(4 machines), we use wireshark to capture the packets on one machine's network interface. After 20 min capturing, we find that the average packets per second is 11.9, each packet carries 4 bytes udp data. So when we only consider the udp data length, the bandwidth usage of each machine is $11.9 * 4 = 47.6\ Bps$. This is in accordance with our theoretical derivation, where each machine sends and receives $\frac{N-1}{N} * \frac{1}{T} * 4$ packets in total per second, with $N = 4,\ T = 0.25\ second,$ the theoretical bandwidth usage is 12 packets per second and 48 Bps. The reason we only consider the udp data bytes is that the udp payload of the messages(either ping/ack or join/leave) is very small compared to the entire frame, and the frame size is dependent on the deploying environment.

(ii) We measure the bandwidth usage of a node joins, leaves or fails when there are 4 machines already in the system. We define each join/leave/fail procedure starts when the user types command and ends when the member list converges. The join procedure lasts for 1.026 second, and the bandwidth usage on the joining machine is 276 Bps. The leave procedure lasts for 1.519 second, and the bandwidth usage on the leaving machine is 218 Bps. The failure detecting and disseminating procedure lasts for 2.18 second, and the bandwidth usage on one monitor machine is 138 Bps.

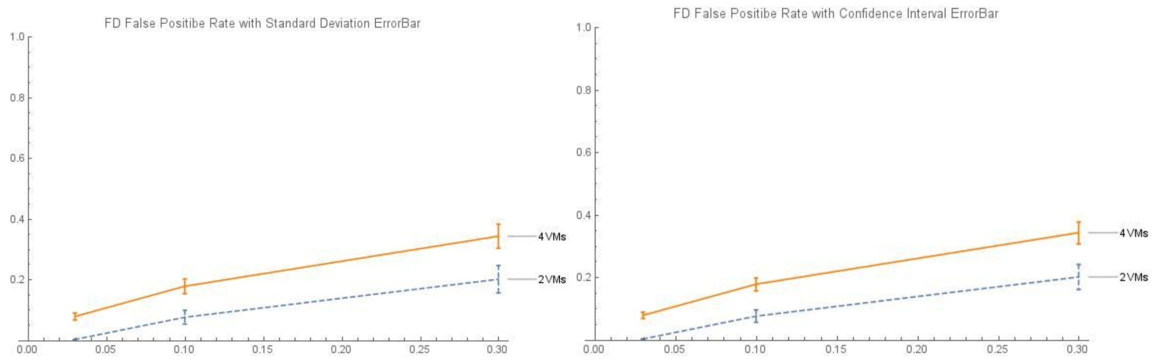


Fig. 4 Failure Detector False Positive Rate with SD/CI in Error Bar

(iii) From the graph above, we can observe that when the packet drop rate is fixed, the false positive rate increases as the number of machines increases; when the number of machines is fixed, the positive rate increases as the packet drop rate increases.