ECE428 MP2 Report

Feize Shi (feizes2) and Dongheng Lin (dl58)
University of Illinois, Urbana Champaign

Design and Overview

Our team has designed the MP structure as a Service-based system. A MembershipManager class is designed to manage the membership list encapsulated in a MemberInfo struct. The MemberInfo stores the corresponding membership list in a map with process name in the format hostname-daemonTimestamp, which is unique to all the members; the value of the map contains corresponding hostname, counter, and status code attributes which are essential for gossip behaviors. The MemberShipManager is registered to the Service Class, which will be initialized and respond to any incoming UDP packets based on specified method names included in the UDP message headers. The Service class then calls the corresponding handlers responsible for accomplishing tasks..

The network package is the infrastructure we have implemented to support the above functionalities. First, network packets provide RPC-like message encoding and decoding capabilities. We define the Meta structure to describe how the UDP packet content is encoded, and the receiver decides how to encode it by changing the fields. For each request, a header is generated describing the type of request and the length of the request (to distinguish different requests from the data stream). This is the basis of piggyback, while the UDP server hands the request to various handlers depending on the type of request.

At the same time, we implemented a UDP server to listen for requests from other members, and the members themselves use a UDP client to send requests. In the client, we have implemented piggyback functionality, i.e., adding multiple requests (such as heartbeat, suspiciousion, join, etc.) to a UDP packet payload (a typical UDP payload structure is shown in the figure below). For Suspicion requests, the call method encodes the request and adds it to the corresponding buffer until the heartbeat request arrives and triggers the actual UDP send. At the same time, we allow some requests (e.g., commands) to allow them to be sent directly.

Capitalizing on the adaptability of our UDP-based service registration, we've incorporated a mechanism to toggle gossip suspicion on or off. Executed via a UDP request to the local where the command is triggered, this change is handled locally and subsequently broadcasted to the entire virtual machine cluster using regular HeartBeats. Any subsequent turn command on/off supersedes its predecessor. In addition, a fixed' early' timestamp will be assigned to the peer when the program starts, so newly joined nodes will comply with either the default setting or the changes made during the cluster's lifetime.

We designed our gossip + suspicion protocol by incorporating a new type of message disseminated in the group via gossiping: the Suspicion message. Every node maintains an additional incarnation number attribute for the members. The member can only increment this incarnation number when it receives a suspicion message suspecting it. The suspected node then spreads out the new message containing a higher incarnation number and such that supersedes all the other suspicions with smaller incarnation numbers. If no such information is spread back in the network in a period, a node will eventually be identified as removed.

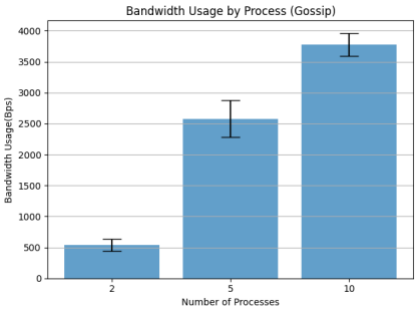
Experiments

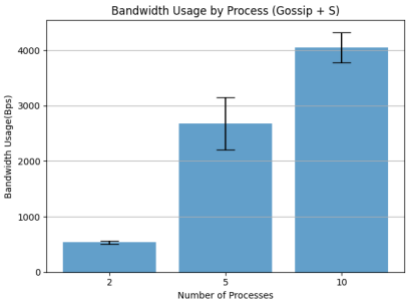
To assess the efficiency and resilience of our program, we've established a series of experimental tests. Initially, we measured and contrasted the bandwidth usage of both gossip protocol variants. Next, we examined the failure detectiIon time across various failure scenarios,

offering insights into the system's efficiency. Lastly, to simulate intermittent packet loss common in real-world settings, we introduced an artificial packet drop mechanism during receipt. This approach furnishes a critical metric for gauging the robustness of our program.

Result Evaluation

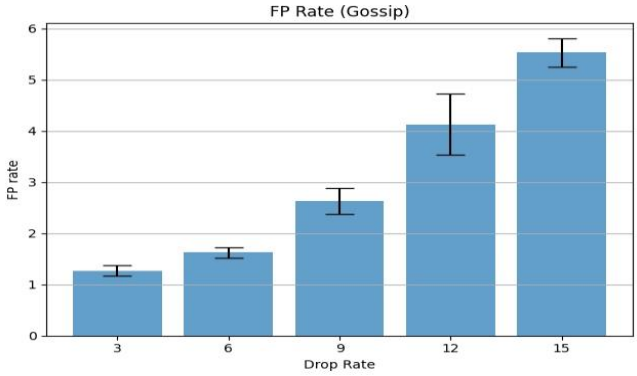
Gossip showed lower bandwidth in terms of bandwidth in the no-failure scenario, whereas Gossip+S exhibited but higher standard deviation. This have revealed the efficiency of gossiping under piggyback setup of UDP messages.

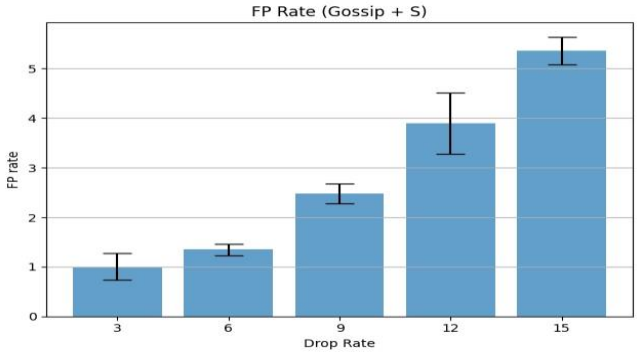




The detection time graph illustrates that as the number of failures increased, Gossip detects faster, while Gossip+S is more stable.

With the introduction of artificial message drops, Gossip had a higher false positive rate, while Gossip+S had better convergence. This have shown that the suspicion mechanism has provided positive effect in reducing the false positive phenomena under packet loss scenario.





In conclusion, the comparative analysis between Gossip and Gossip+S provides valuable insights into their performance and reliability under various conditions. The plots not only depict their behaviors but also pave the way for understanding the underlying factors that contribute to these observed trends.