The system has two main modules – the **Listener** and the **Pinger**.

1. **Listener**: This module listens for a bunch of different types of messages and responds to them as shown in the diagram below. Whenever a message is received, a new **Message Handler** (for taking appropriate actions) thread is spawned for the message and the listener goes on to wait for other messages without blocking for each message to be processed first before moving on to the next message.
2. **Pinger**: This periodically sends *(after 750ms)* a ping message to a randomly chosen process from its member list. If the process responds with an ACK, it knows the process is alive so it goes to sleep until the next round of pinging. If the process does not respond back with an ACK in the timeout period *(100ms)* then it starts a *ping-req* by choosing *k = min(member_list.size, 3)* other members from the member list and asking them to send a ping message to the "suspicious" process who might have failed – if any of them replies back within a timeout period (200ms) then the process is known to be alive and the Pinger goes back to sleep until the next round of pinging. If neither of them replies, the same *ping-req* mechanism is retried *(2 more times)* – if still no response is received then the process is marked as failed and this information is broadcasted to every other process in the member list using the **fail** message.

Bootstrapping: Whenever a new process joins the group, it should know the group leader already. It will send an **init** message to the group leader who will respond back with a list of processes it knows in the group. The leader will also broadcast all the processes it knows about the arrival of a new process using the **join** message.

Leader: The leader runs a **Backup** thread which periodically *(every 1000ms)* serializes the members list and writes it to the filesystem. If the leader crashes and joins back again, it can read this backup file to restore its members list.

Scalability: No part of the protocol we have implemented turns out to be a bottleneck when scaling the number of processes – each ping round is independent of the number of total number of processes because only 1 processes is selected at random and only k processes are selected at random for ping-req. Similarly, when a node joins or leaves/fails, a broadcast message is sent to all processes but the bandwidth for this is fairly low as demonstrated later in the report.

Debugging: The distributed grep developed in MP1 came in handy when taking the logs from various processes and trying to figure out where the messages were being lost or when the appropriate response was not being sent back – our debug log contains the entire packet being sent and received so it was relatively easy to find the discrepancies using the distributed grep to search through logs being generated on different physical machines.

Message Format:
type <ping, ack, pingReq, leave, join, fail, init>, srcIP, srcPort, ID (of the recipient process), sequenceNumber, pingReqTarget (only for pingReq messages to specify who started the pingReq)

Bandwidth Usage: The bandwidth usage of a group of 4 is showed in the following graph. The measurement utilized the actual length of message calculating in the program. The background bandwidth includes ping and ack message. The join bandwidth includes size of member list and broadcast message. The fail bandwidth includes broadcast message and ping-req message. The leave bandwidth includes broadcast message. Assuming each member is allowed to send N byte per second.

Bandwith type	Bandwidth usage (bytes/sec)
Background bandwidth	300 N
Join	387N
Leave	246 N
Fail	1288 N

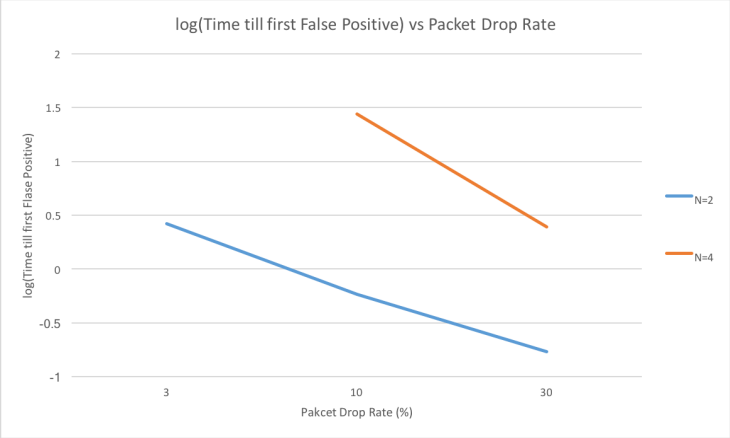
False Positive Rate: To simulate the real Ethernet environment, the packet is manually dropped before sending out by rate of 3%, 10%, and 30%. To keep the member list correct in the beginning, the "join" message and "init" message are not dropped. The experiment considered the time when *first false detection appeared* in the group. Say that group member is N. Sample number is 16.

N = 2

Packets dropped rate	Average time	Standard deviation	Confidence interval(90%)
3%	2.65 s	0.86	2.274~3.026 s
10%	0.58 s	0.28	0.457~0.703 s
30%	0.17 s	0.086	0.132~0.208 s

N = 4

Packets dropped rate	Average time	Standard deviation	Confidence interval(95%)
3%	More then 60 min	----	----
10%	27.5 s	16.77	20.2~34.8 s
30%	2.46 s	1.04	2.0~2.9 s



Discussion: Higher the first false detection time is, lower the false positive rate is. The data shows that false positive is increase when the packet loss rate increase, and reduce when the group size increase.

Architecture: