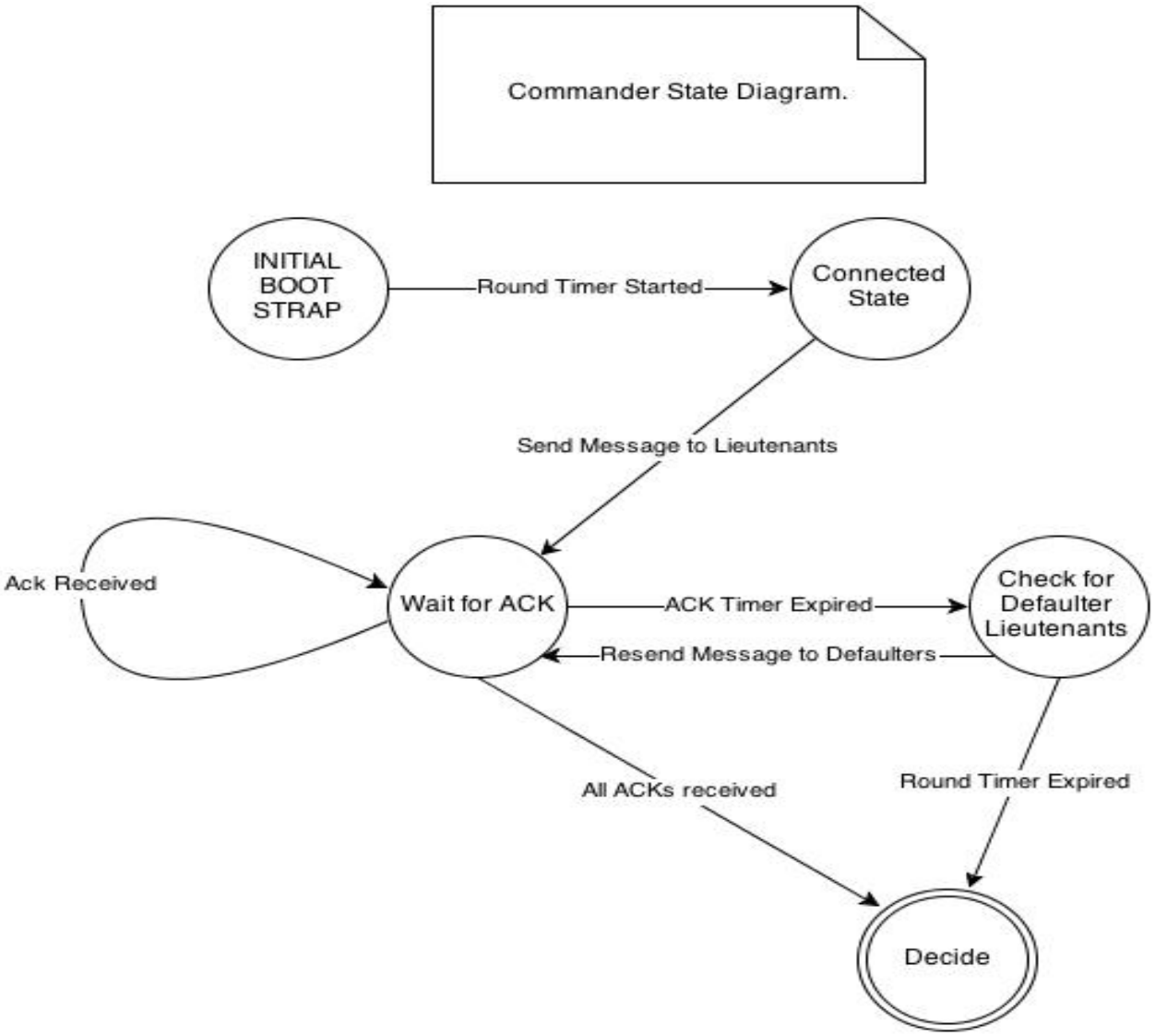
CS505 - Project 1 -Byzantine Agreement with Authenticated Messages

==========

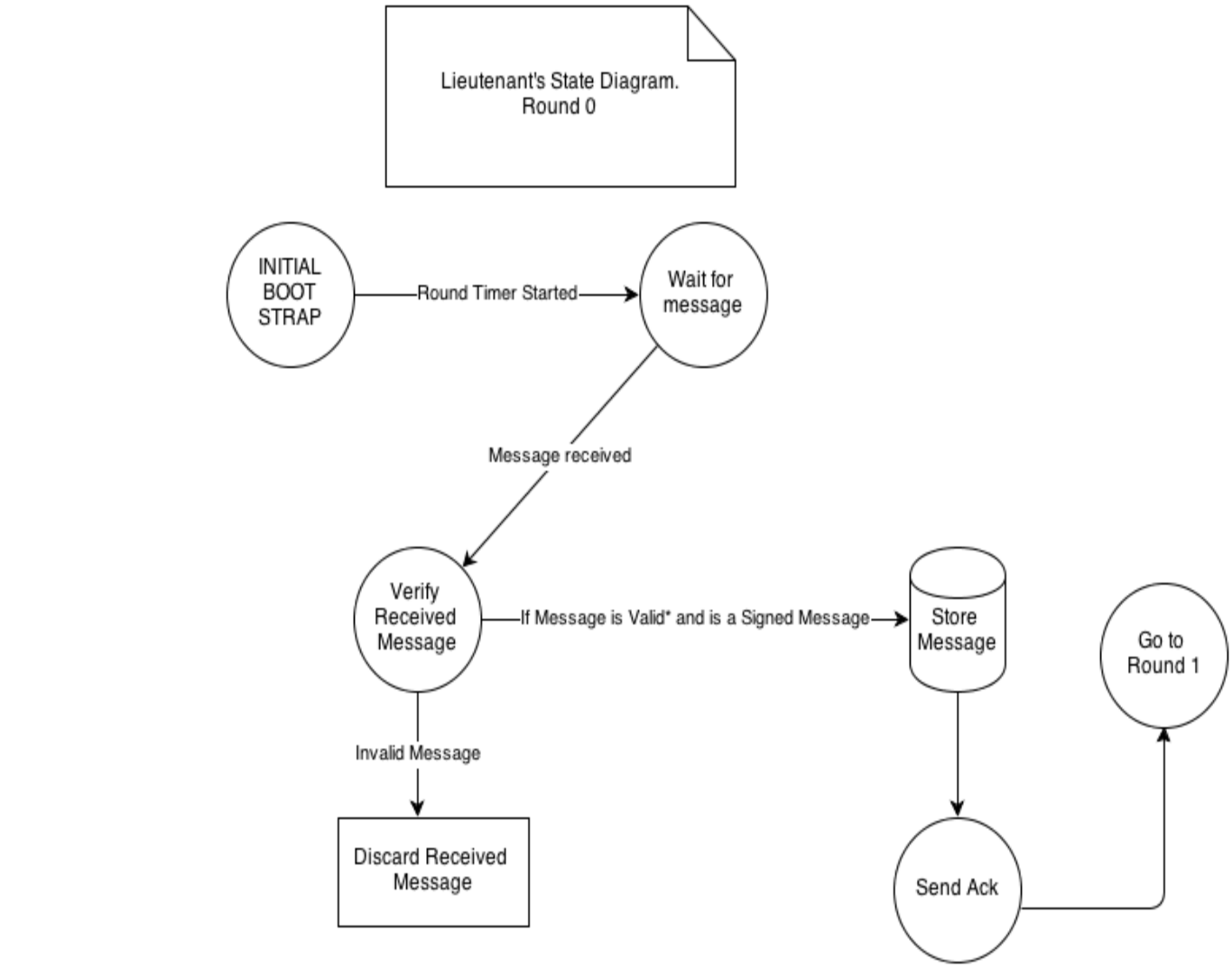
State Diagrams for Commander and lieutenant:

Commander:



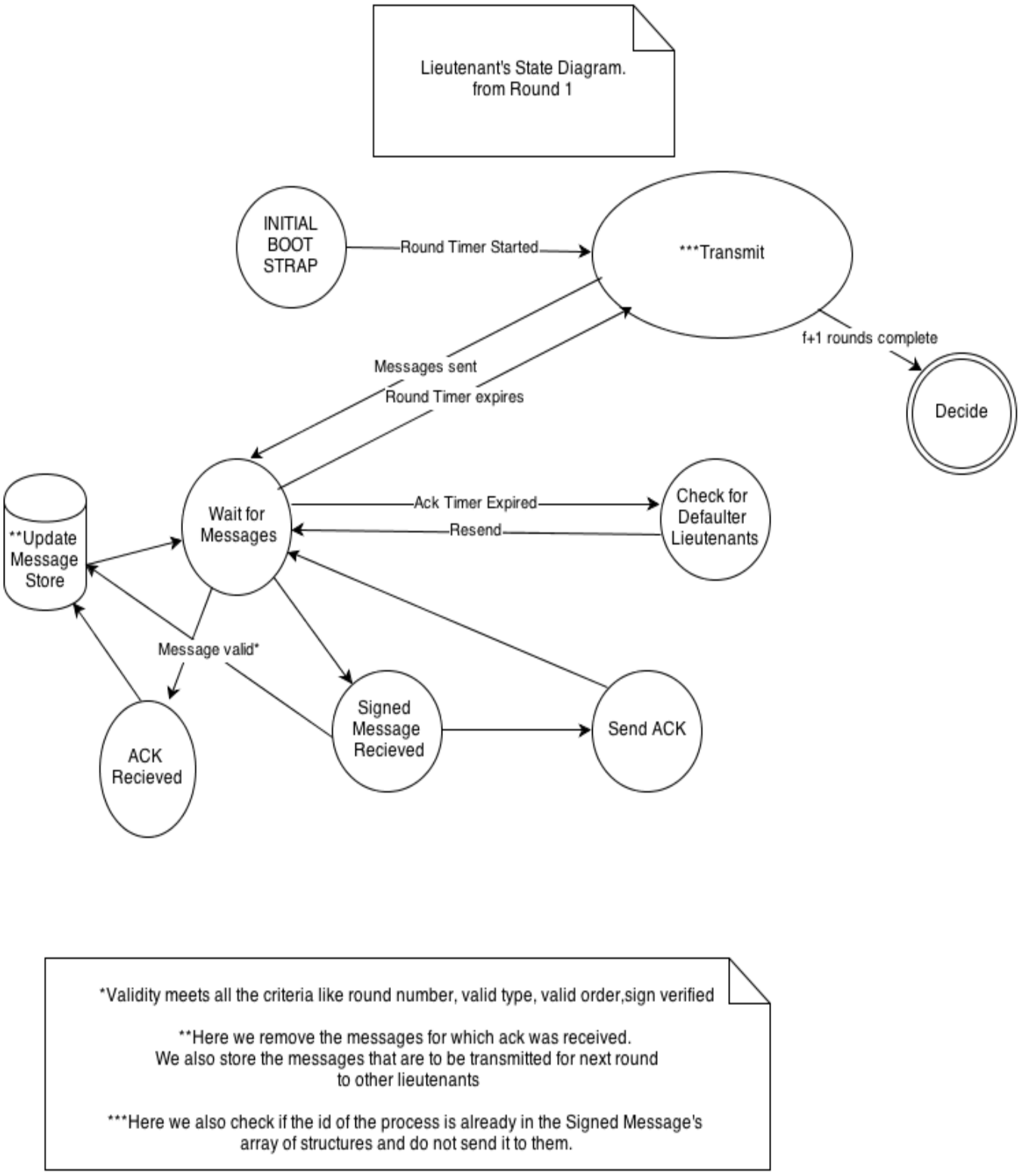
Defaulter Lieutenants are those who have not send the ACK's

Lieutenant:



Initial Boot Strap means that socket are properly opened and binded.
*Valid message should meet the round criteria, order criteria

Lieutenants Round 1-f+1:



System Architecture:

The system solves the consensus problem given a set of byzantine systems up to 'f' in a set of 'n' synchronous systems. Here we assume that the processes do not crash. Each of the processes has a set of values at the end of the algorithm and they decide on the value which agrees with all the processes which are not byzantine. If the commander is not byzantine, his value should be chosen by all the processes which are not byzantine.

The system has two kinds of components implemented in a single program which communicate synchronously using a timer:

1. Commander

We assign the first host in the given hostfile.txt as a commander. The system gets initiated by the commander sending a signed order to the lieutenants in round 0. The commander also waits for an acknowledgement from each of the lieutenants for the message he sent else he retransmits after a defined system parameter called acknowledgement retransmit timer. The commander runs for a time - round time also a system parameter. After the completion of a round, the commander prints his decision and exits.

2. Lieutenant.

The other hosts or processes in the hostfile.txt act as lieutenants. The lieutenant receives a signed order from the commander and verifies the signature. Later he retransmits the message to all the other lieutenants in the system. The lieutenant also maintains a record of which message has been sent as he has to retransmit a message if he doesn't receive an acknowledgement. The lieutenant runs for f+1 rounds where 'f' is the number of faulty hosts or processes in the system. After completion of f+1 rounds he prints his decision and exits.

Design Decisions:

1. To make the commander and lieutenants synchronous used a timer which will help make the systems synchronous. The commander in some cases may start late so, the lieutenants wake up after a minimum lag of a round's time. So, here the delay guarantees for the initial round are assumed to be minimal.
2. Implemented a non-blocking receive function which helps following a defined round time.
3. Used a UDP retransmit time-out to be about one-fifth of a round's time.
4.. The commander signs the order he is to send using RSA techniques instead of signing the whole message. Tried to sign the whole 'sigs' array but was unable to verify it. Later on each lieutenant signs the signature (after verification) of the previous general (commander or lieutenant) before sending his

message.

5. Used gettimeofday() to both check the round timer and the ack timer. The “Server.h” file contains the functions “get_now” to get the current time in milli seconds and “time_to_seconds” finds the difference between two given times.

6. Each lieutenant has a list to maintain the set of messages to be sent in the next round. This is tracker using two c++ map objects which store the hostname as key and Signed Message as a value.

```
map< char *,SignedMessage *> sender_msg_to_list_processes;
map< char *,SignedMessage *> reciever_msg_to_list_processes;
```

7. A map of hostnames to IP addresses is also being used to track the hosts to which the message is to be sent. A map of hostname to id’s to be used in the ‘sig’ structure is also being used.

8. An ack will always be sent to the sender whether the message is valid or invalid to reduce the number of retransmissions on the network.

9. The “Server.h” file contains a round time and a udp retransmit timer of 500 and 100 milliseconds respectively as follows:.

```
#define ROUND_TIME 500
#define UDP_RETRANSMIT_TIMER 100
```

These can be changed in order to look for other scenarios.

Implementation Issues:

1. Delay guarantees are assumed to be minimal. Assumed that commander does not start after a predefined sleep time in the script of 5 secs. Made the “recvfrom” non-blocking to count the rounds exactly.

2. The processes are assumed to be able to communicate with each other. If they can’t open a socket or bind fails, the program will exit.