Mandatory 3 – Chit Chat

Alexander Llewelyn Yan Fatt Soo & Aske Thomas Neye

System Architecture:

Our chat service uses a centralized *server-client* architecture. The *server* maintains a list of all connected *clients* and their message streams. The *clients* connect to the server, send messages, and receive broadcast messages.

gRPC communication model

To decide which type of RPC communication to use (unary / server streaming / client streaming / bidirectional streaming) we look at the requirements of each part of the chat service.

When the client publishes a message it sends an individual message using a unary gRPC type (SendMessage).

Looking at the server it broadcasts messages to all client using server streaming but the client must also receive updates while sending messages.

So, we use bidirectional streaming between each client and the server, which makes it possible to achieve this.

Proto design

```
// =====================
// === Service Layer ===
// =====================
service ChatService {
  rpc Chat(stream JoinMsgLeave) returns (stream ServerMessage);
}
```

The Chat RPC method is implemented as a bidirectional streaming call. This means both the *client* and the *server* can send messages independently over an open stream once the connection is established. This is an asynchronous network. Since the two streams are independent, the *client* and *server* can read and write messages in any order.

Clients send messages of type JoinMsgLeave to the server. These messages can represent three event types: a *client* joining (JOIN), sending a chat message (MESSAGE), or leaving (LEAVE) the chat.

The *server* broadcasts messages of type ServerMessage to all participants.

How do we handle the Lamport clock

In a fully distributed system, each *client* and the *server* would keep their own Lamport clocks. Our system uses a centralized version of the Lamport clock. This is because the gRPC chat uses bidirectional streaming, where all communication goes through the server. The server sees all messages and therefore can maintain a consistent logical ordering for everyone, and on a more practical level: because this is how we ended up doing it. Just because it seemed like the way to go about it. Only too late did we realise that we were probably supposed to update each client and server separately.

On the *client* side:

We keep a local variable:

```
var lamport int64 = 0
```

Before sending any message (Join, Message, Leave) we increment the clock:

```
lamport++
msg := &pb.JoinMsgLeave{
    Id:         id,
    ClientName: name,
    Type:       pb.EventType_MESSAGE,
    Timestamp:  lamport,
    Msg:        text,
}
stream.Send(msg)
```

On the *server* side:

The ChatServiceServer struct has the Lamport clock

```
clock   int64 // Lamport clock
```

When the server receives any type of event from a client it updates the clock

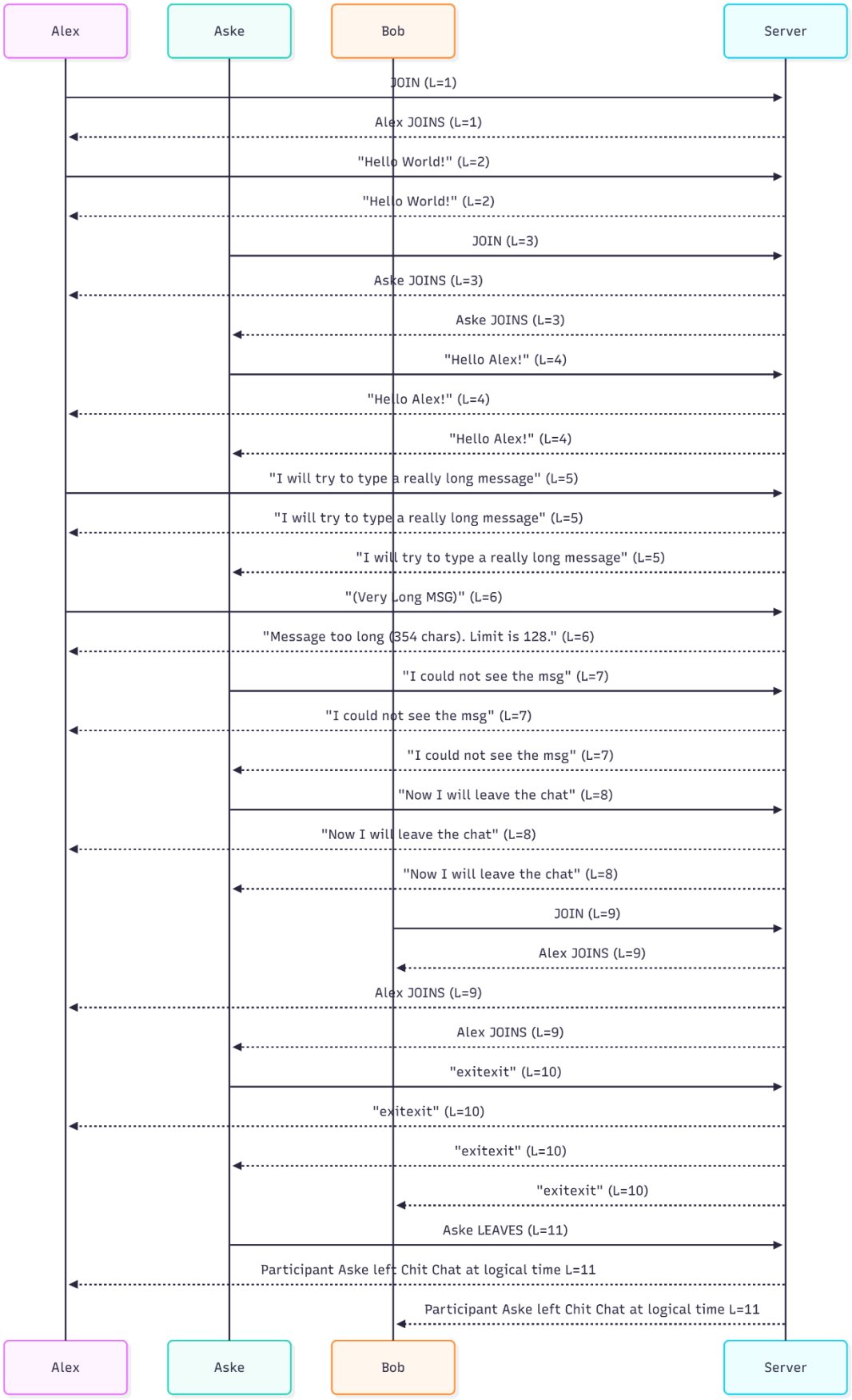
```
s.clock = max(s.clock, msg.Timestamp) + 1
currentLamport := s.clock
```

Then the server broadcasts as part of the log the current Lamport timestamp.

```
log.Printf("[Server] Message from %s: %s (L=%d)", msg.ClientName, msg.Msg, currentLamport)
```

Diagram of RPC calls with the Lamport timestamps

Diagram is made using log until L=11 – Until Aske Leaves the chat


Alex
Aske
Bob
Server
JOIN (L=1)
Alex JOINS (L=1)
"Hello World!" (L=2)
"Hello World!" (L=2)
JOIN (L=3)
Aske JOINS (L=3)
Aske JOINS (L=3)
"Hello Alex!" (L=4)
"Hello Alex!" (L=4)
"Hello Alex!" (L=4)
"I will try to type a really long message" (L=5)
"I will try to type a really long message" (L=5)
"I will try to type a really long message" (L=5)
"(Very Long MSG)" (L=6)
"Message too long (354 chars). Limit is 128." (L=6)
"I could not see the msg" (L=7)
"I could not see the msg" (L=7)
"I could not see the msg" (L=7)
"Now I will leave the chat" (L=8)
"Now I will leave the chat" (L=8)
"Now I will leave the chat" (L=8)
JOIN (L=9)
Alex JOINS (L=9)
Alex JOINS (L=9)
Alex JOINS (L=9)
"exitexit" (L=10)
"exitexit" (L=10)
"exitexit" (L=10)
"exitexit" (L=10)
Aske LEAVES (L=11)
Participant Aske left Chit Chat at logical time L=11
Participant Aske left Chit Chat at logical time L=11
Alex
Aske
Bob
Server