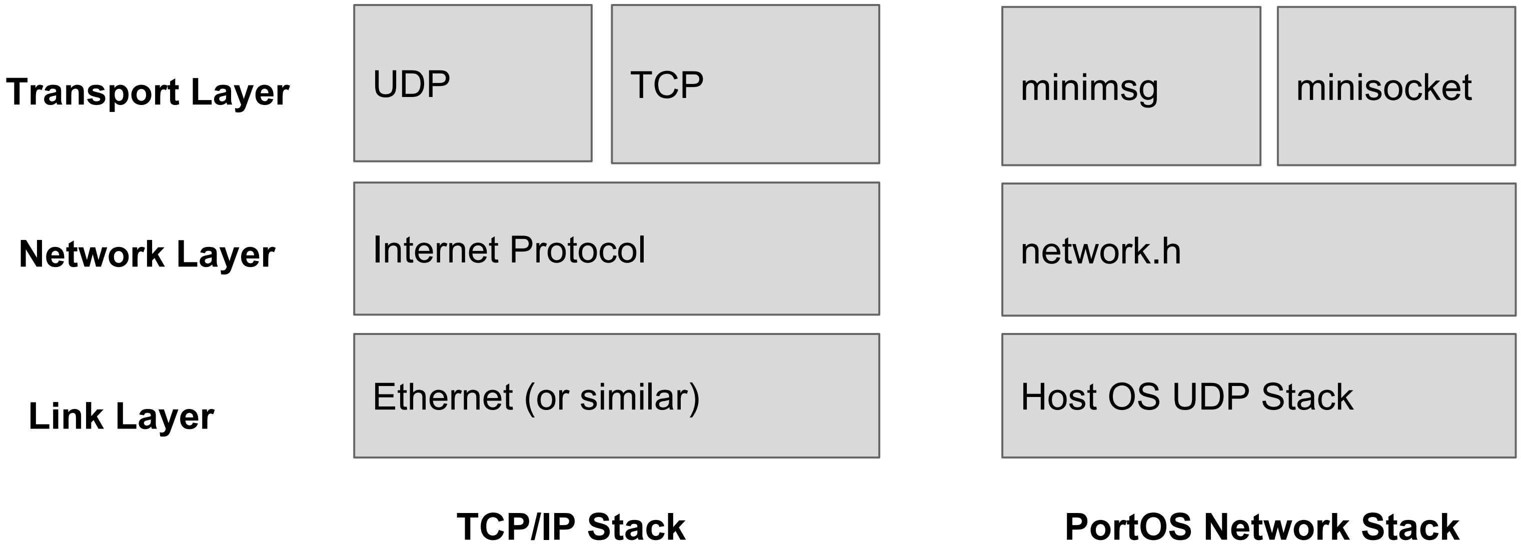
Project 4: Reliable Networking

presented by Kai Mast

Announcements

- Project 4 is already released
- I assume you’ve read the project description
- Due November 4th
- This is a pretty complex project ➪ Start early!

Our network stack vs. the real world

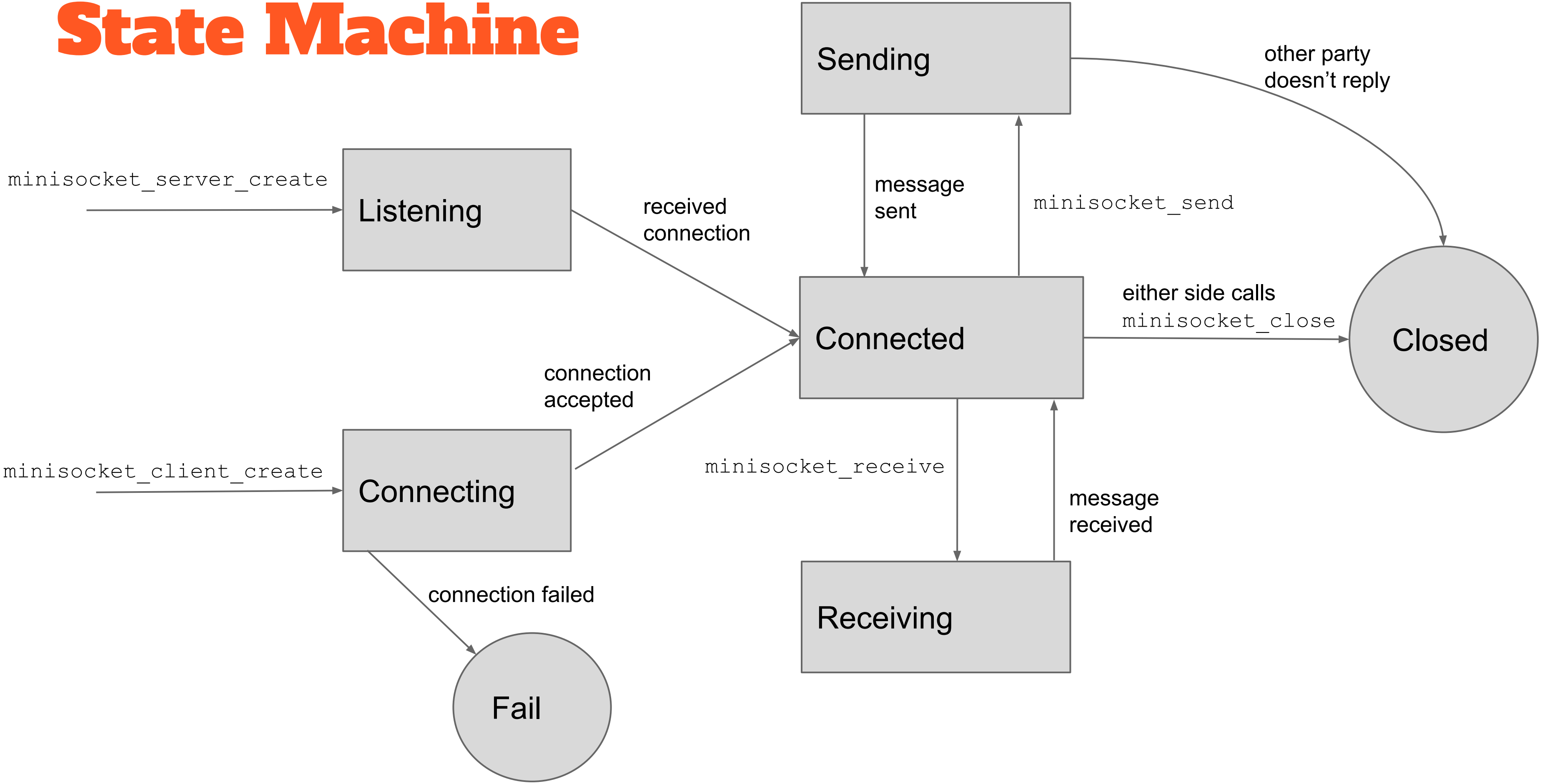


	TCP/IP Stack		PortOS Network Stack	
Transport Layer	UDP	TCP	minimsg	minisocket
Network Layer	Internet Protocol		network.h	
Link Layer	Ethernet (or similar)		Host OS UDP Stack	

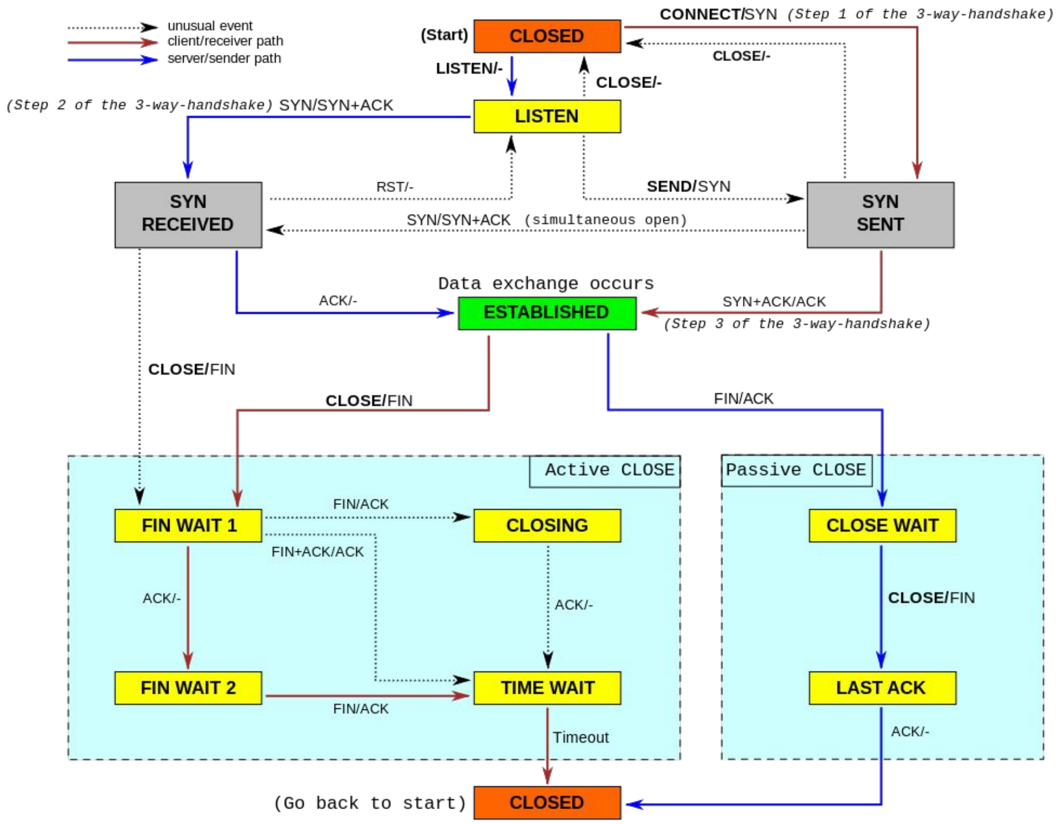
Minisocket is a simplified TCP

- Protocol is connection oriented
  - You must find a way to establish a connection between two endpoints
- Data is sent as a continuous stream of bytes
  - Messages are an application level concept
  - Minisocket must maintain correct ordering
- No limit on message sizes
  - You must fragment and reassemble the data

State Machine



Of course, it's much more complicated…



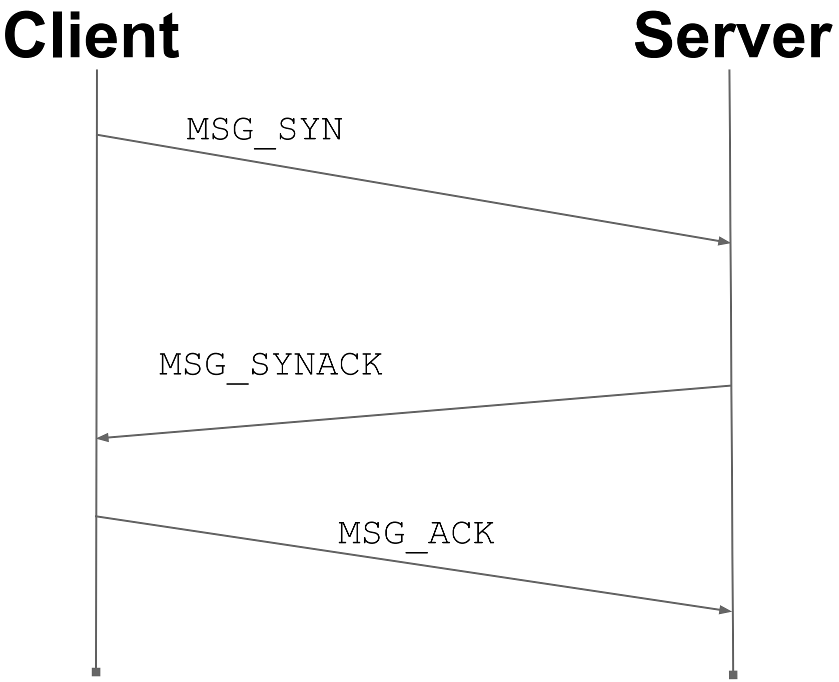
TCP State Machine
Source: Wikipedia/Cube00
License: CC BY-SA 3.0

What can go wrong?

- Any party can die
- Messages can get lost
- Data might be reordered
- Network might be partitioned

Welcome to the fun world of distributed systems!

Three-Way Handshake

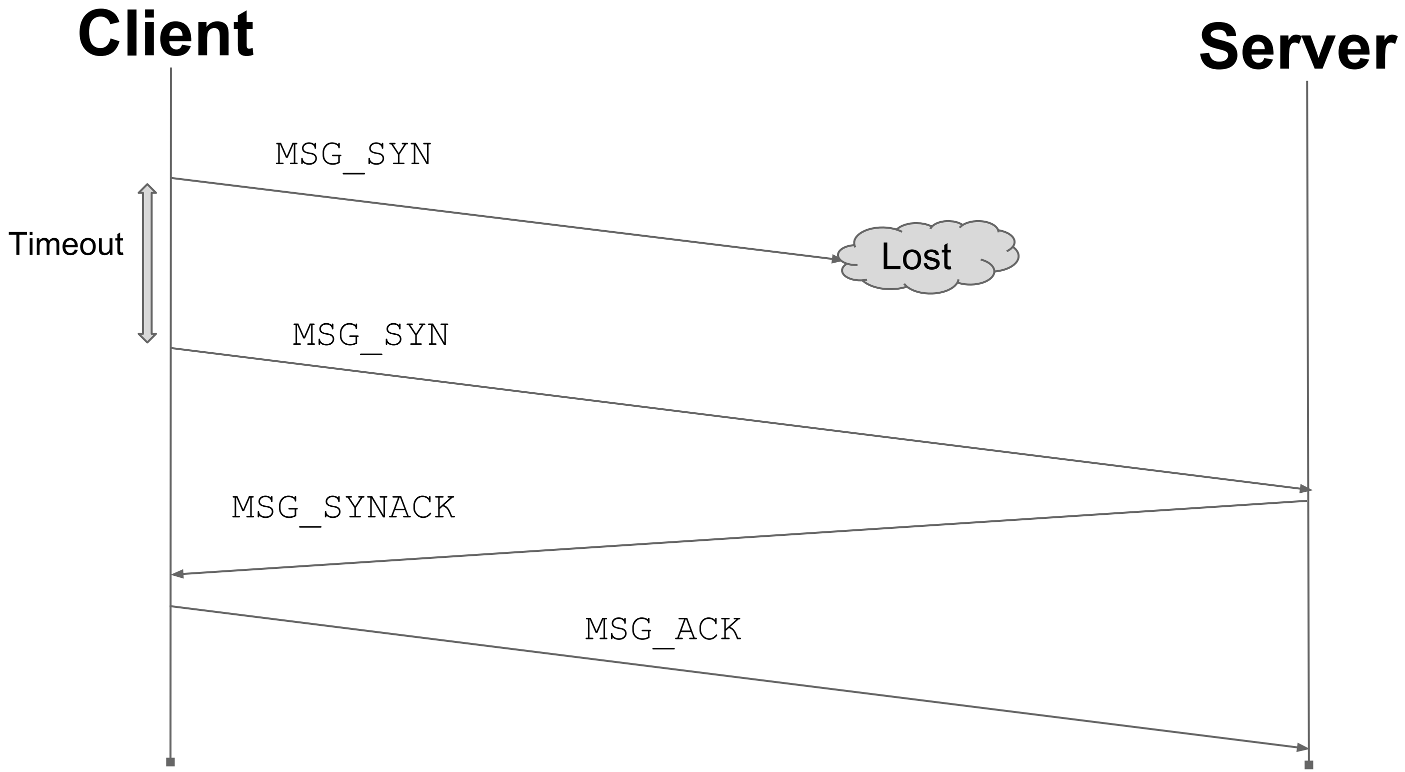


Non-blocking protocol

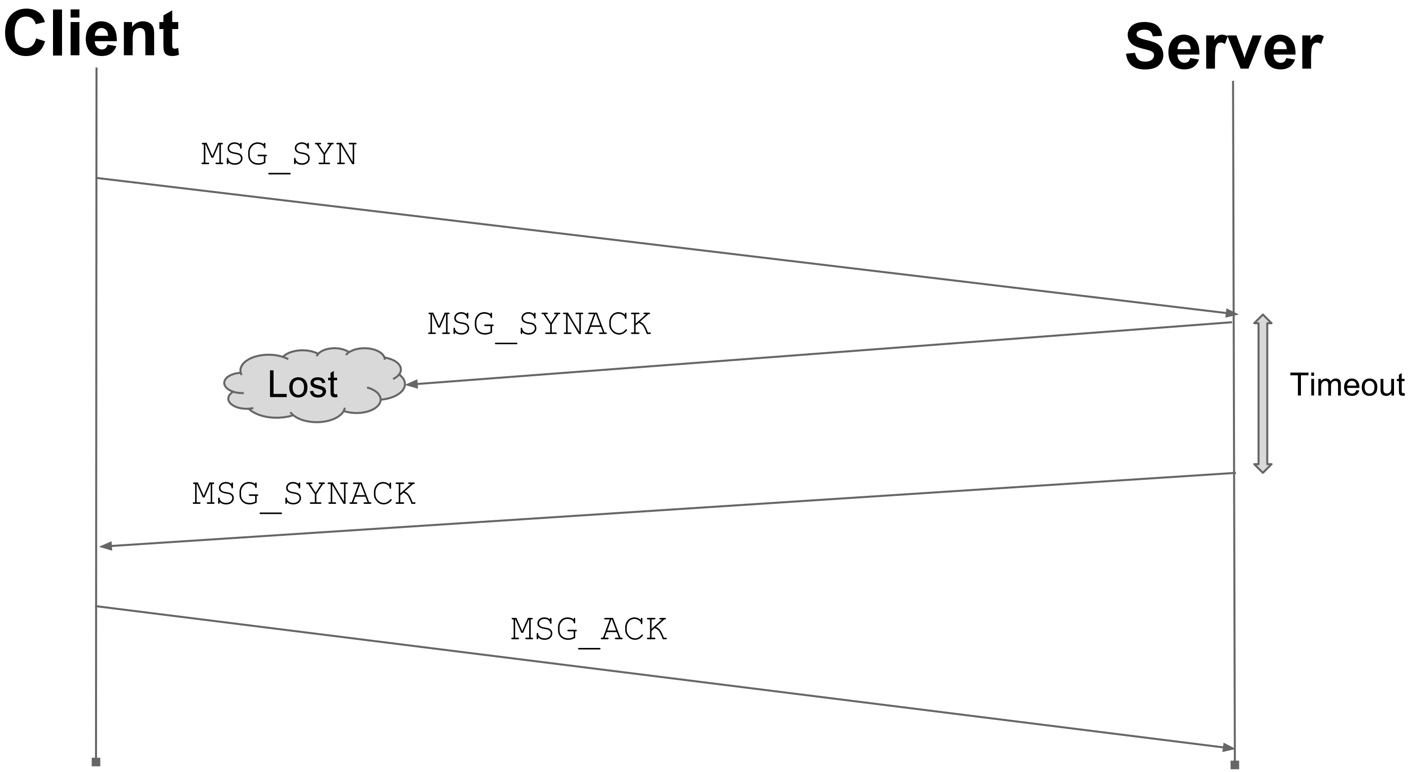
- Any packet might be lost
- Will be resent up to seven times
- Timeout doubles every time

Initial Timeout: 100ms
⇛ Give up after 12.7s

Messages can get lost

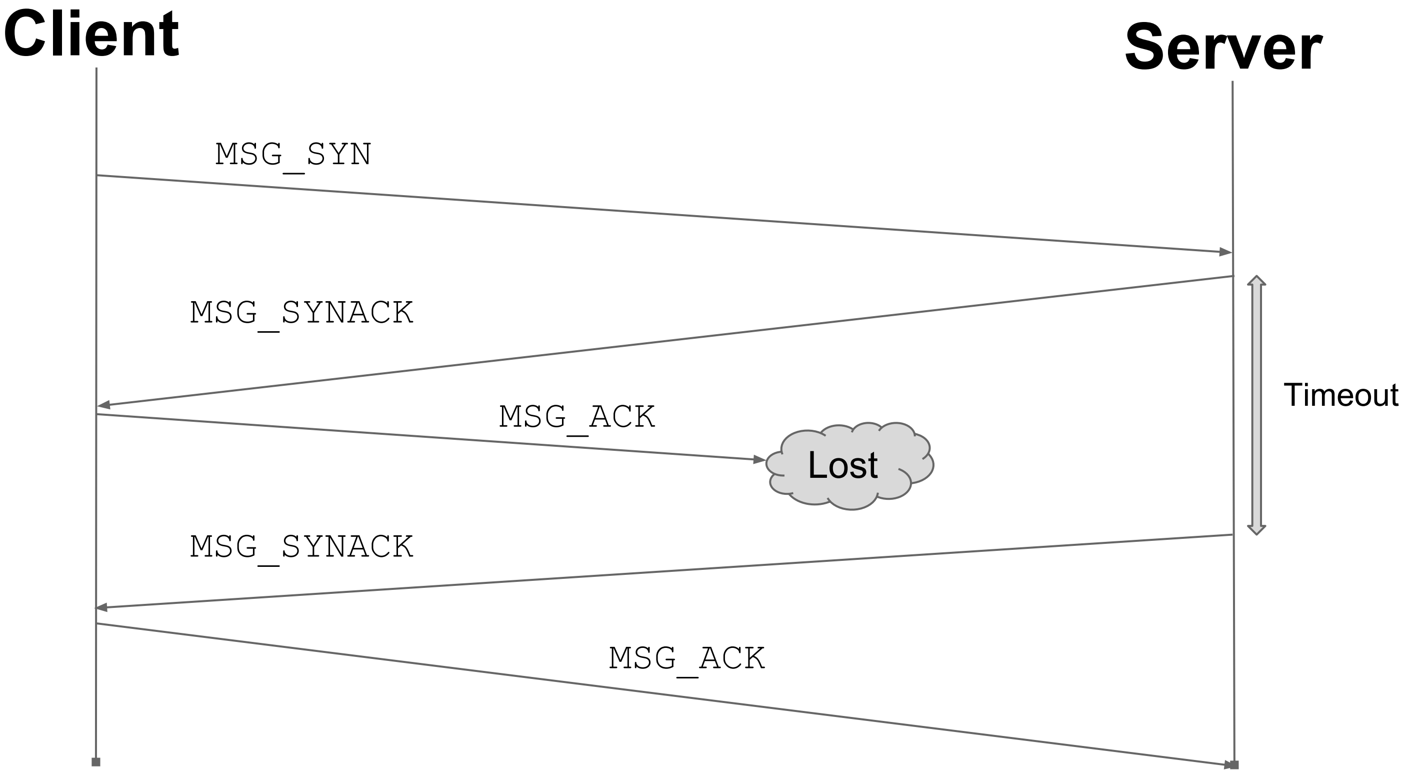


Messages can get lost

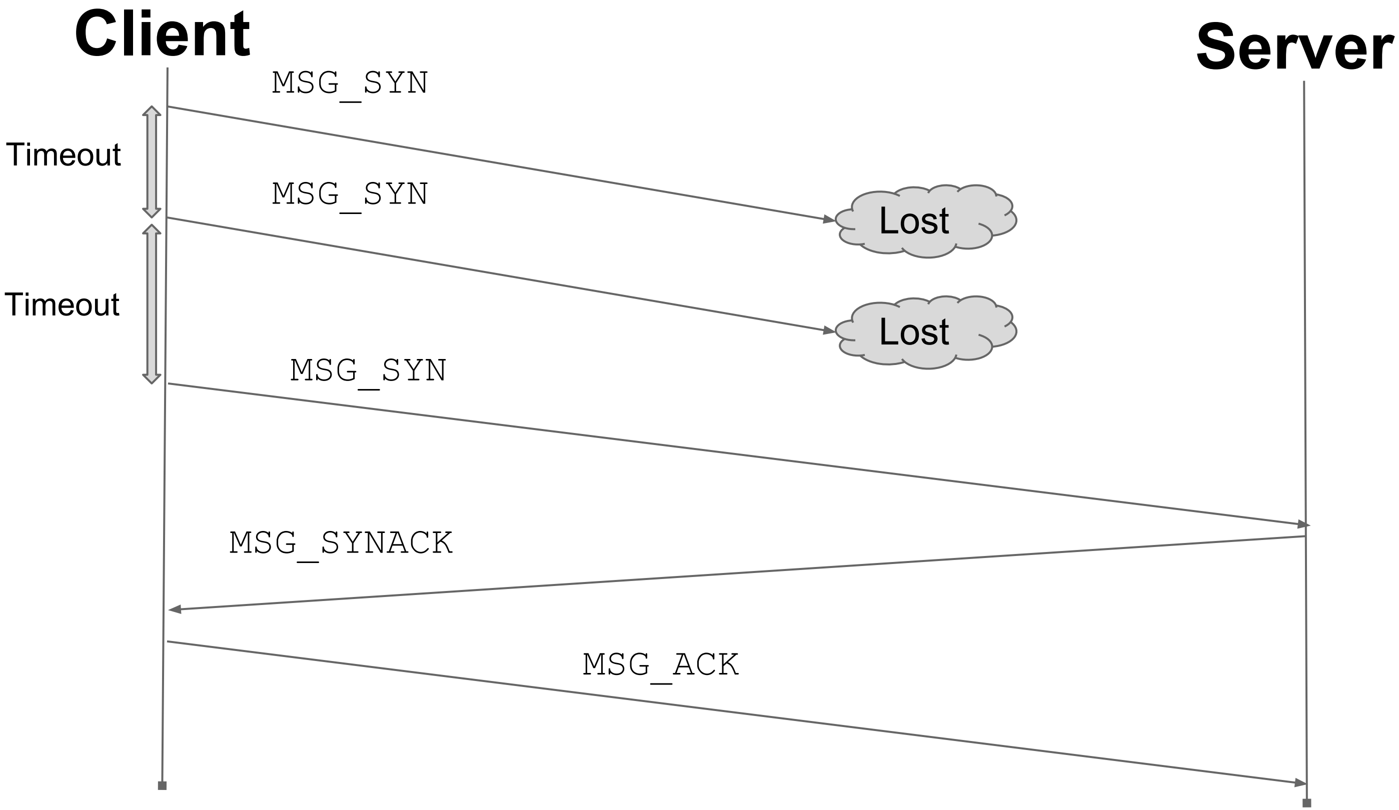


Note: In this case both parties might retransmit

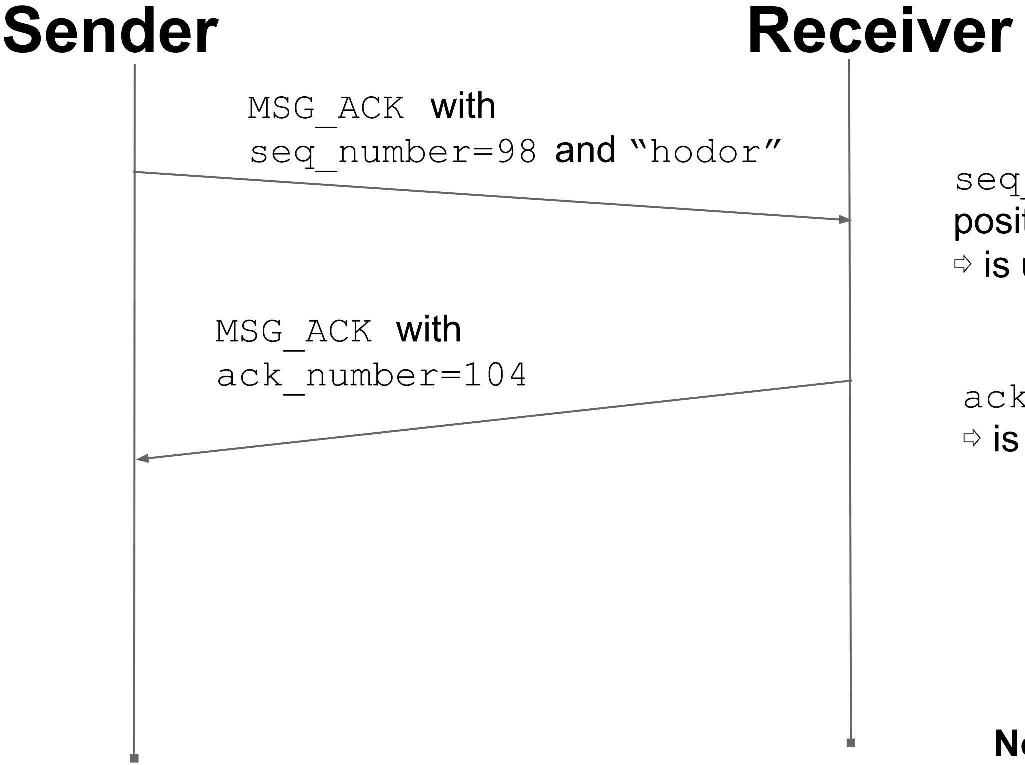
Messages can get lost



Messages can get lost multiple times



SEQ and ACK Numbers



Sender

Receiver

`MSG_ACK` with `seq_number=98` and `“hodor”`

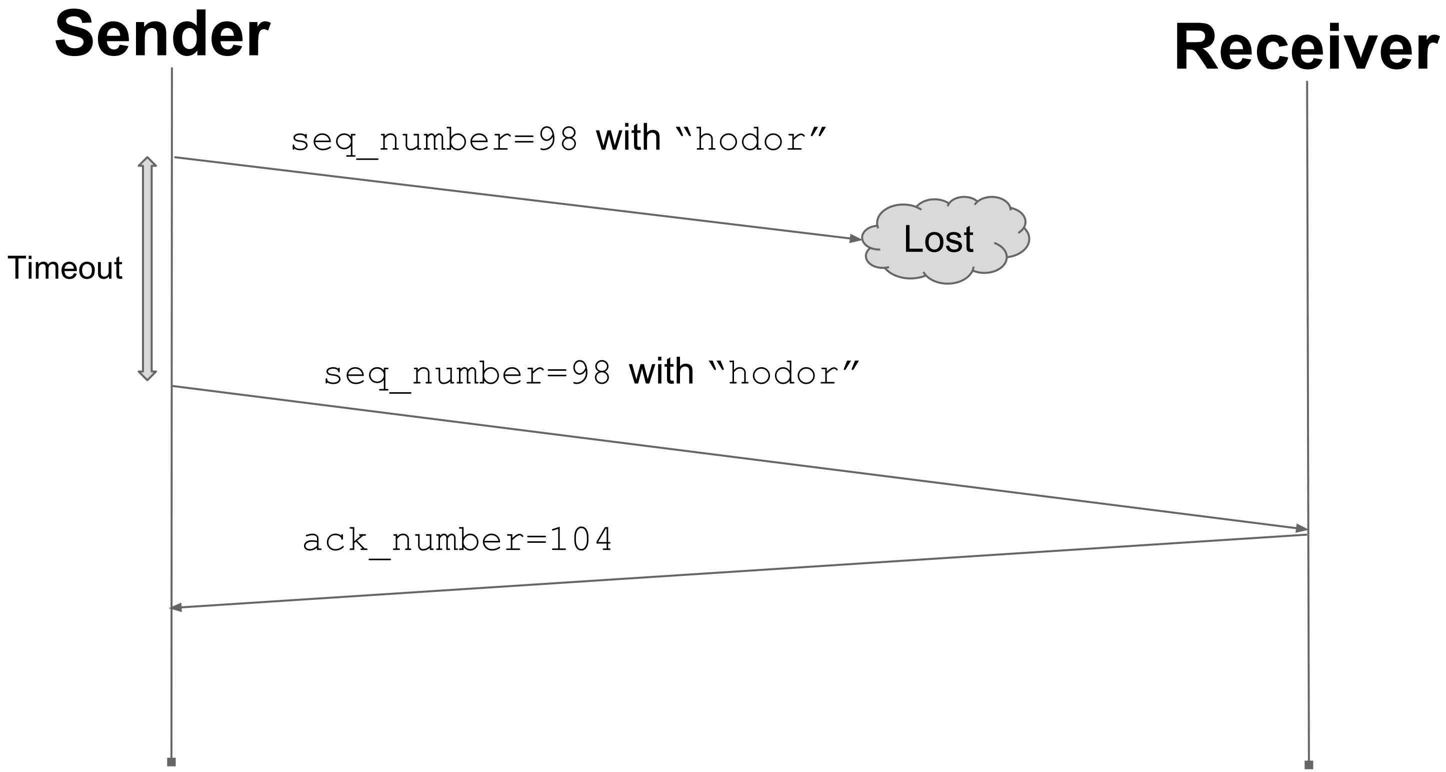
`MSG_ACK` with `ack_number=104`

`seq_number` shows current write position
➪ is used to order messages

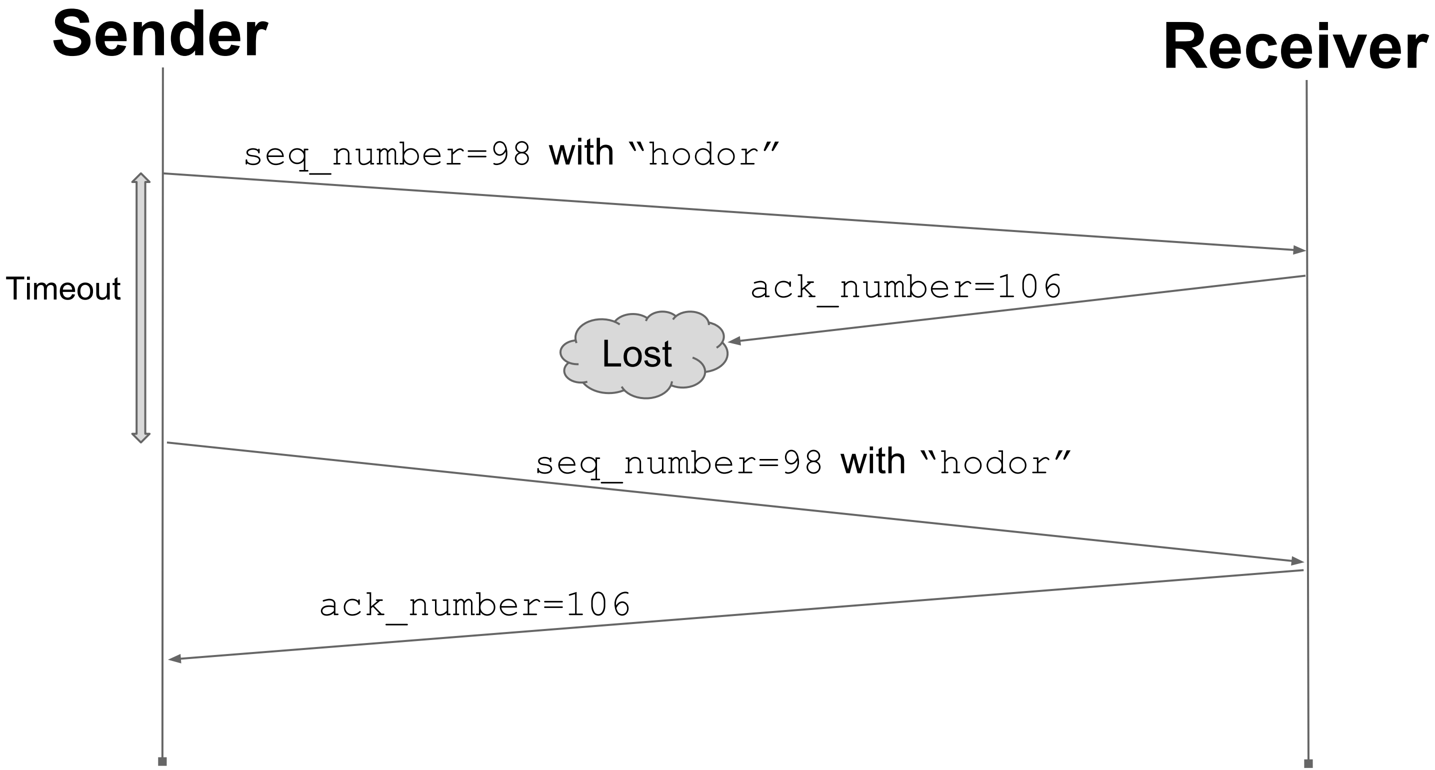
`ack_number` shows total received bytes
➪ is used to resend lost messages

Note: This is a symmetric channel. Both parties can send and receive.

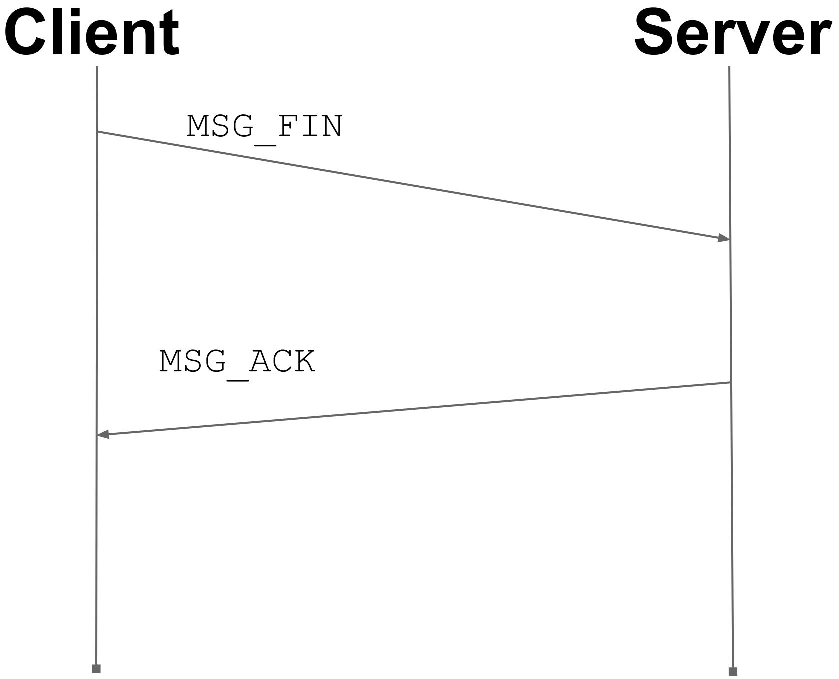
Again, messages can get lost



Again, messages can get lost

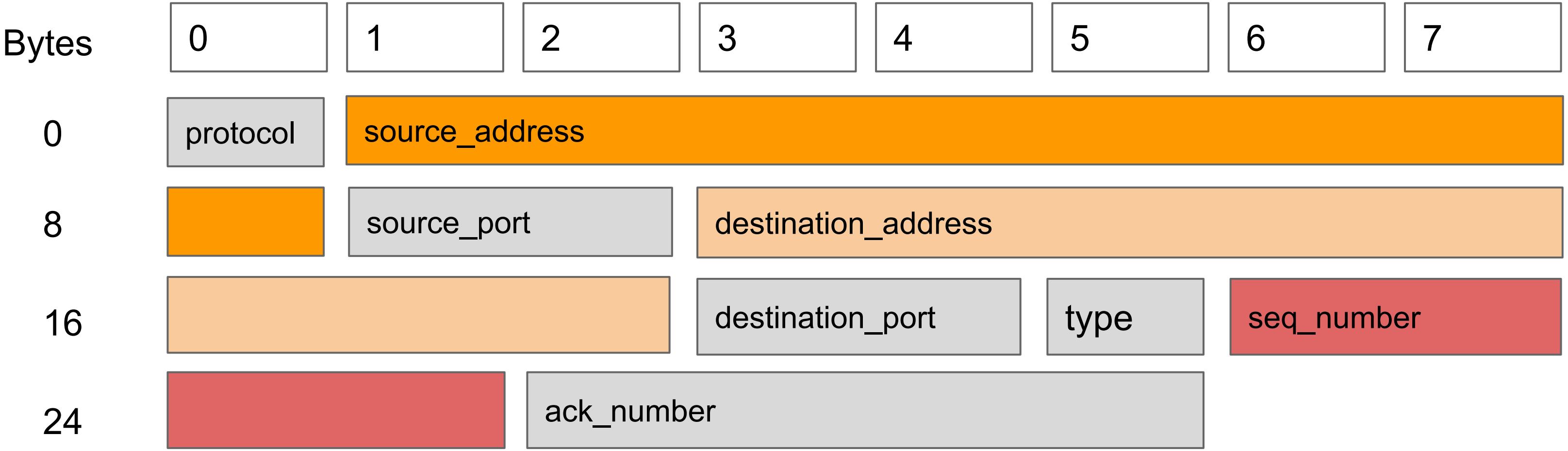


Closing connections



Again, this is a symmetric protocol.
Both sides can close the connection.

Minisocket Header



The first 20 bytes are identical to minimsg_header!

Use protocol field to multiplex protocols.

Tricky Part: How to implement timeout?

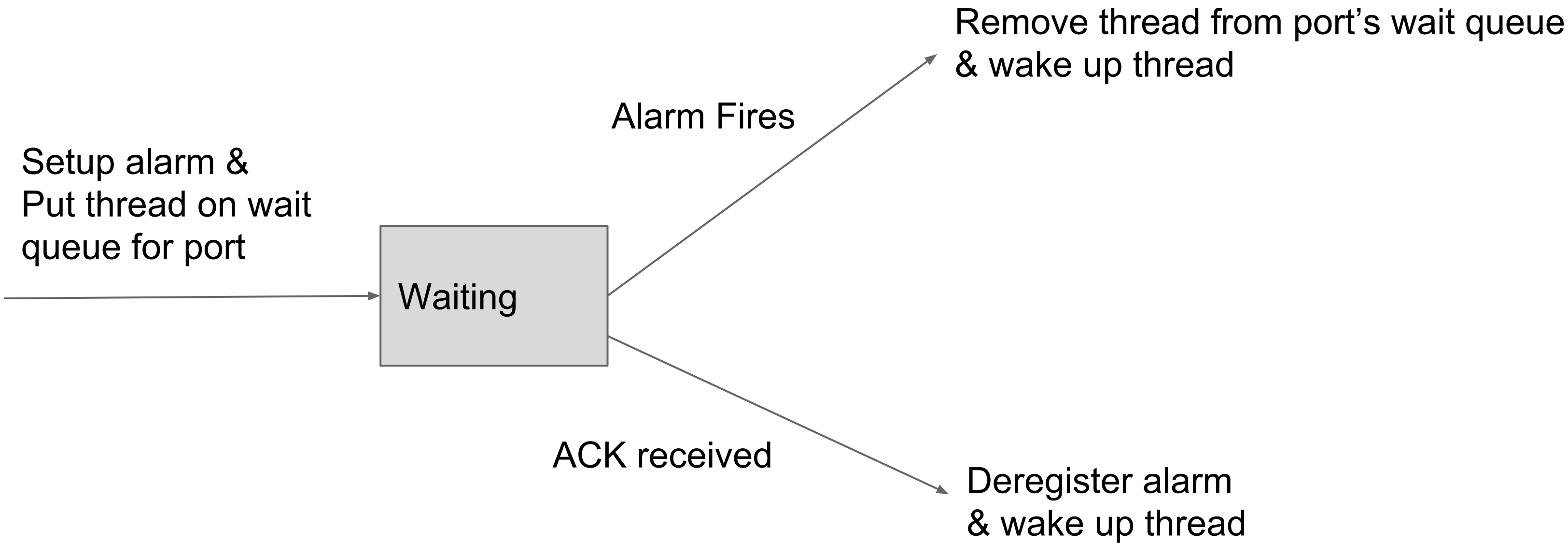
Remember that:

- Parties might never respond
- Multiple threads can call minisocket_send() on the same port

Things you must avoid:

- Putting threads on the run queue more than once
- Thread keeps waiting after message is received
- Thread blocks infinitely

Tricky Part: How to implement timeout?



Setup alarm &
Put thread on wait
queue for port

Waiting

Alarm Fires

Remove thread from port’s wait queue
& wake up thread

ACK received

Deregister alarm
& wake up thread

To make it a little easier

- You don’t have to implement congestion control
- Sending one packet at a time is sufficient
- minimsg_send can block until corresponding ACK is received

But you can implement window sizes > 1 if you want to!

(and have the time…)

Where to start

- Think about the state machine from earlier!
- Try to make connection setup and termination work first.
- Test with no loss and single-thread access

Test all the code!

- What happens if you send very large messages?
- Can you handle a lot of messages?
- What if there is loss?
- If one party crashes the other one shouldn’t.
- What if multiple threads are sending/receiving from the same port?

Test all the code!

In network.c:

```
double loss_rate = 0.0;

double duplication_rate = 0.0;

bool synthetic_network = false;
```

These change the behavior of the network

You have to set this to true for the other values to have any effect!

Updating your project

Merge by hand

- Copy new function signatures header files
- Make sure everything compiles!

Files that changed:

network, miniheader, Makefile

New files:

minisocket, conn-network[1-3]

Good Luck

Questions?

As always, if you need help, come to office hours!