Project 4: Reliable Networking

Slide heritage: Previous TAs
Announcements

● Project 4 has been released
● I assume you’ve read the project description
● Due Friday, April 14th
● This is a pretty complex project ⇝ Start early!
Our network stack vs. the real world

<table>
<thead>
<tr>
<th>Transport Layer</th>
<th>TCP/IP Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>PortOS Network Stack</td>
</tr>
<tr>
<td>TCP</td>
<td></td>
</tr>
<tr>
<td>Internet Protocol</td>
<td></td>
</tr>
<tr>
<td>Ethernet (or similar)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Link Layer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet (or similar)</td>
<td>Host OS UDP Stack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Layer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Protocol</td>
<td>network.h</td>
</tr>
<tr>
<td></td>
<td>minimsg</td>
</tr>
<tr>
<td></td>
<td>minisocket</td>
</tr>
</tbody>
</table>

TCP/IP Stack

PortOS Network 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
minisocket_server_create

Listening

receiving connection

message sent

minisocket_send

Sending

message received

other party doesn't reply

Received

either side calls

minisocket_close

Closed

connection accepted

Connected

message received

Other party sends FIN

minisocket_receive

Receiving

connection failed

Fail

minisocket_client_create
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!
Connecting: 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

Client

MSG_SYN

MSG_SYN

MSG_SYNACK

MSG_ACK

Server

Lost

Timeout
Messages can get lost

**Note:** In this case both parties might retransmit
Messages can get lost
Messages can get lost multiple times

Client

MSG_SYN

Timeout

MSG_SYN

Timeout

MSG_SYN

Timeout

MSG_SYNACK

Server

Lost

MSG_ACK
Sending Data: SEQ and ACK Numbers

**Sender**

- MSG_ACK with seq_number=34 and “hodor”

**Receiver**

- seq_number represents how many packets have been sent
  - is used to order messages
- ack_number shows total received packets
  - 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

Sender

seq_number=34 with “hodor”

Receiver

ack_number=34

Lost

seq_number=34 with “hodor”

ack_number=34
Either side can send and receive!

Participant 1

seq=34,ack=12 with “hodor”

seq=12,ack=34

seq=13,ack=34 with “arya”

seq=34,ack=13

Participant 2
Closing connections

Again, this is a symmetric protocol. Both sides can close the connection.
The first 21 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_receive() on the same port
● At most one thread can call minisocket_send() on a 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

Your project has been merged with the latest code. Make sure everything compiles and nothing is missing

New files:
minisocket, conn-network[1-3]
Good Luck

Questions?

As always, if you need help, come to office hours or post your questions on Piazza!