

Sockets for Interprocess Communication

Note Title

4/3/2007

TCP/IP recap

- IP:

Every host has a unique IP address (routers use this)

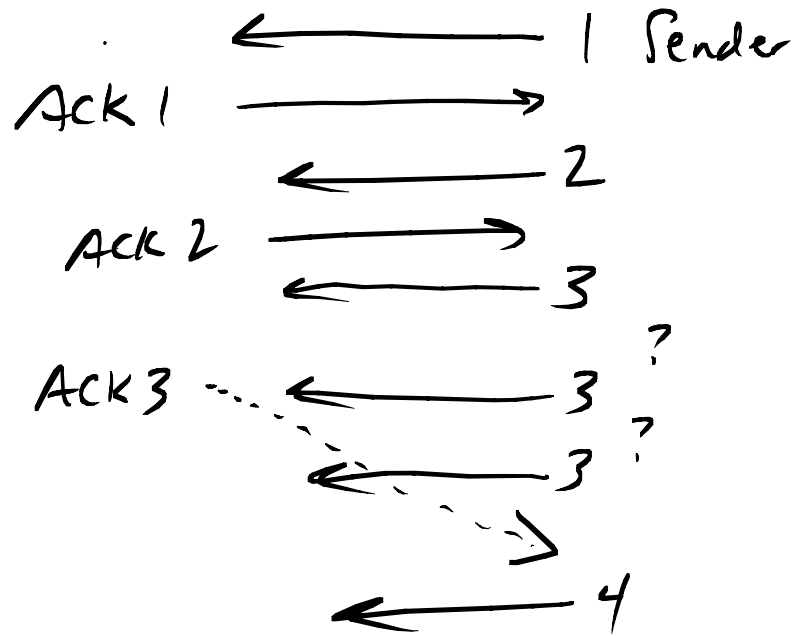
IP is "best effort" — messages (packets) can be lost,
reordered,
duplicated

- TCP provides FIFO stream abstraction over IP
handles:

- reordering packets at receiver
- retransmission of missing packets

Reasoning:

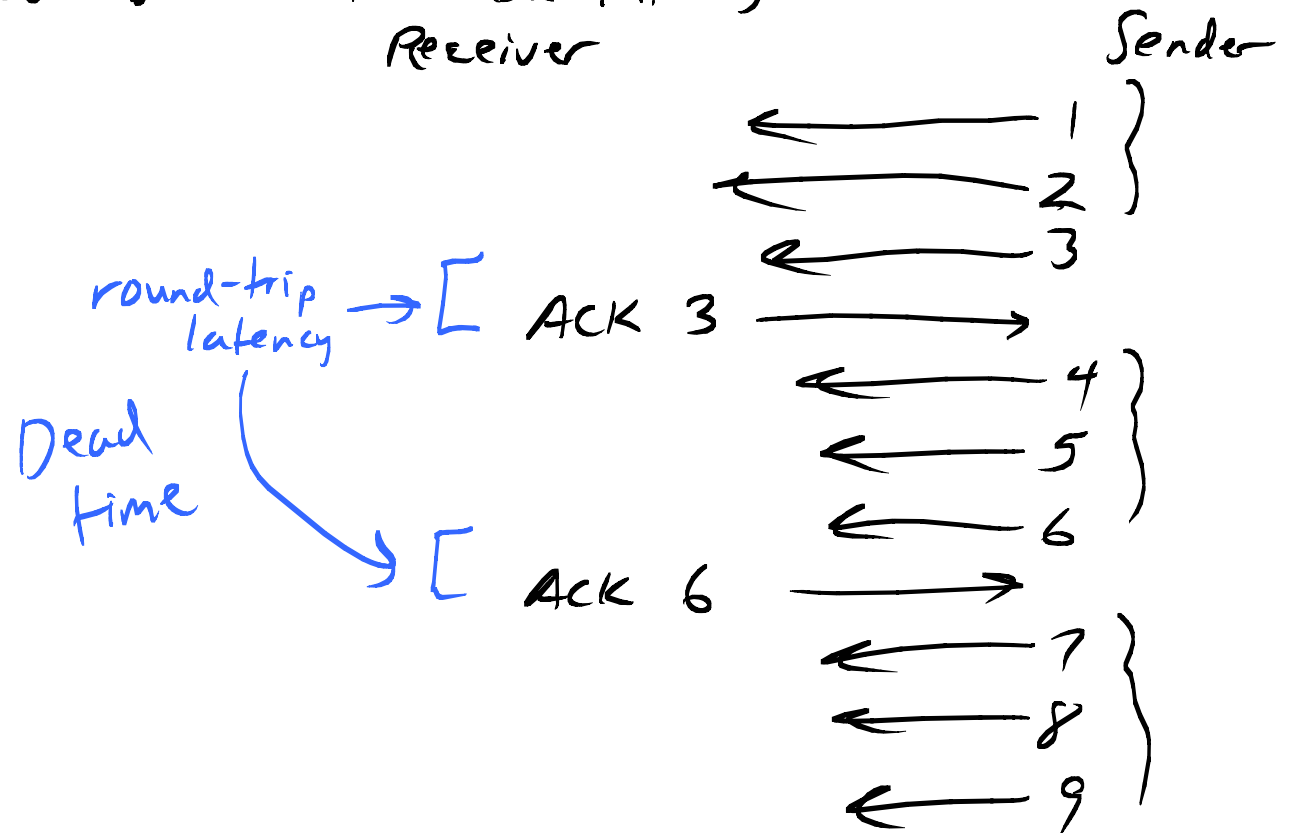
Simplest: Sender waits for ACK of packet n before sending packet $n+1$



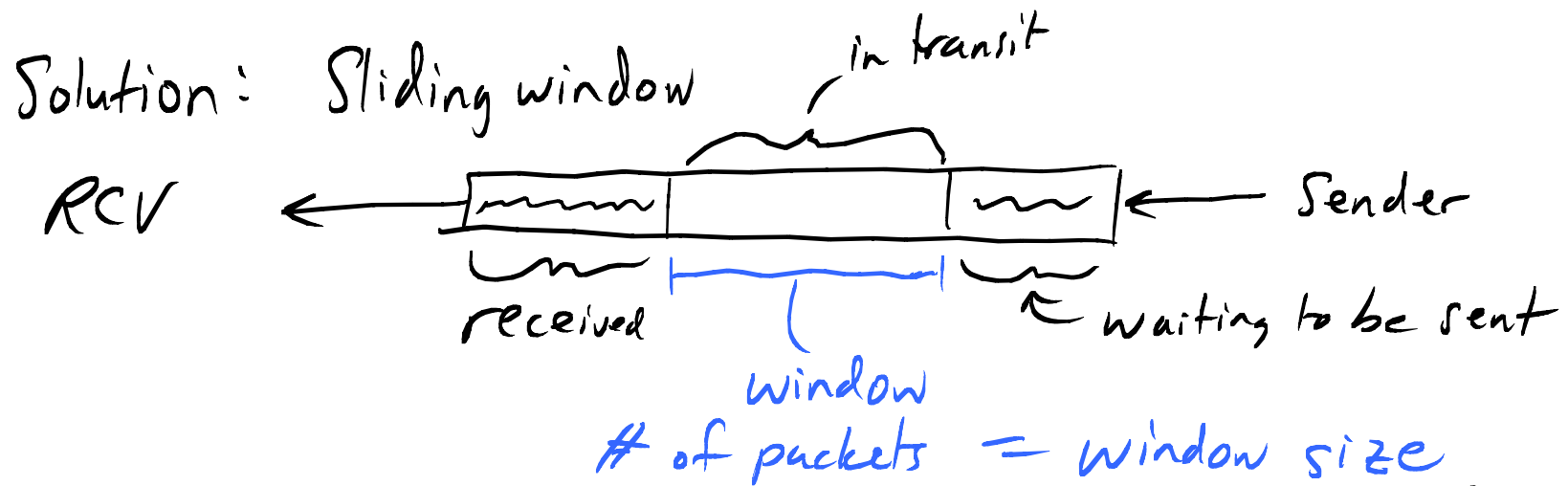
Problem:

Latency bounds
throughput

Send batches to mask latency



Idea: Send packet 4 as soon as get ACK for packet 1
Keep pipe full — some packets always in transit



Sender waits & decreases size of window if

- window is full
- haven't yet received ack for "oldest" packet in the window
→ retransmit for missing ACK (timers)
- gradually ramp back up

Receiver: continuous ACK of largest seq. # in contiguous (no gaps)

Java supports TCP streams with Sockets

- Socket - connection between 2 processes

Socket s

s.getInputStream()

s.getOutputStream()

Can wrap these w/ any of the java.io wrappers.

- How to establish connection?

Asymmetric

Player 1

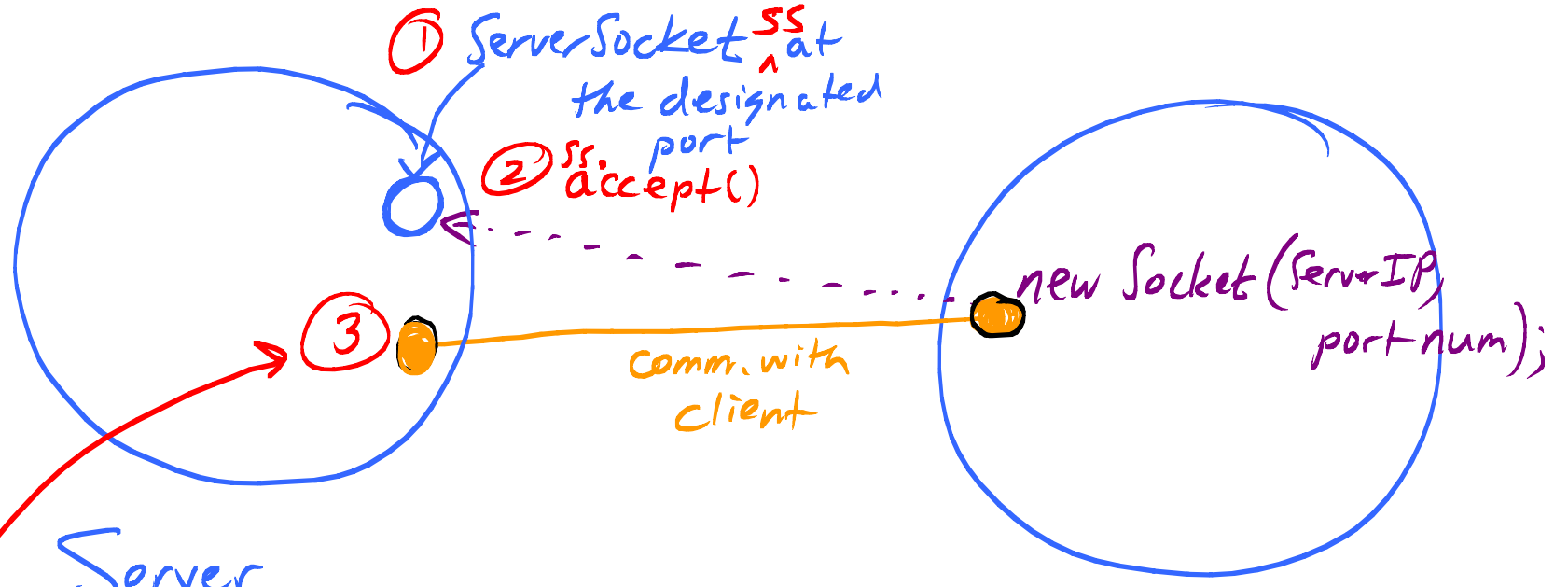
be "ready" to accept a connection

SERVER

Player 2

find player 1 &
ask for a connection

CLIENT



Server

Make itself available on the designated port.

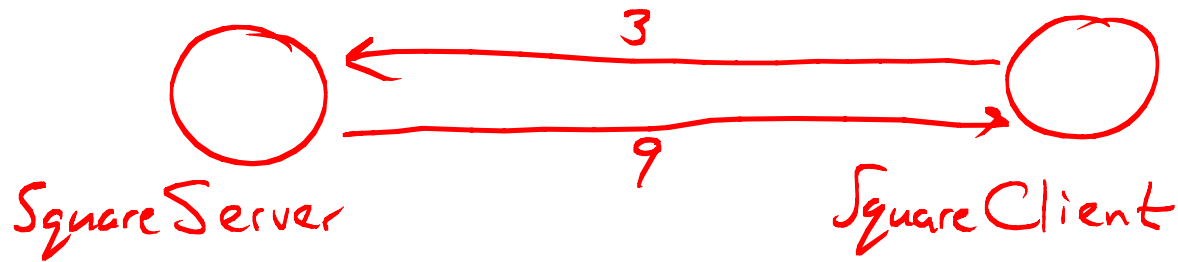
accept() is a blocking call
 ↓ returns a new Socket

↑
 not a server socket

Client

needs to know:

- Server's IP address
- port number



```
public class SquareServer {
```

```
...
```

port number [10,000
-30,000]

```
ServerSocket ss = new ServerSocket(10450);  
Socket s = ss.accept(); // BLOCKING CALL  
DataInputStream in =  
    new DataInputStream(s.getInputStream());  
double d = in.readDouble();  
DataOutputStream out =  
    new DataOutputStream(s.getOutputStream());  
out.writeDouble(d*d);  
s.close();  
ss.close();
```

Could
put in
a loop

```
...
```

On client side

```
public class SquareClient {
```

"localhost" if server is on the same computer

⋮

```
Socket s = new Socket(serverIPaddress, 10450);
```

```
DataOutputStream out =
```

```
    new DataOutputStream(s.getOutputStream());
```

```
out.writeDouble(3);
```

```
..... DataInputStream in =
```

```
    new DataInputStream(s.getInputStream());
```

```
System.out.println(in.readDouble());
```

⋮

```
}
```

Client

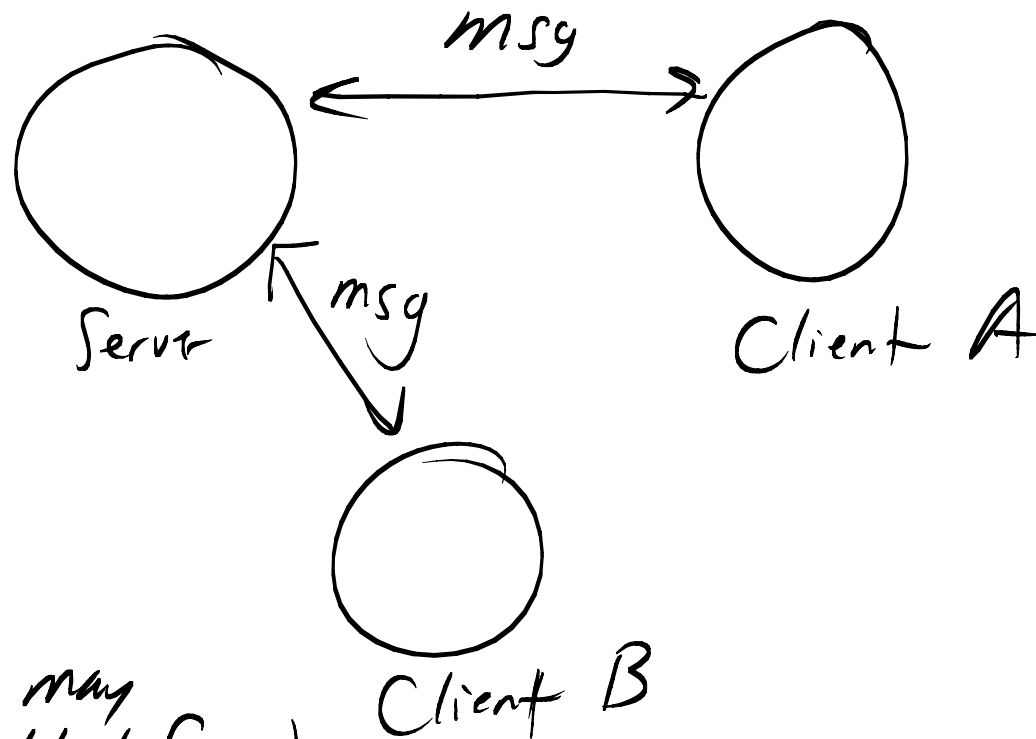


Server

Problem: low level
lots of set-up

Better:
Appl.
Developer
model

- hide connection setup
- support concurrent clients
(some clients may be slow or block forever)



Design a client/server appl. framework:

Asynchronous
model

What should client developers do?

- define message types [classes \Rightarrow object streams]
 - • GUI for user (sitting on top)
 - way to process messages received
- API design choice {
- ① active loop to read incoming messages in app.
 \Rightarrow `Message receive();`
 - ② react as we're told about messages (event-driven)
 \Rightarrow `void messageReceived(Message m);`
- active behavior of client (including sending messages)
(optional, but allow & support Runnable client objects)
 - `send(Message m)` — sends `m` to the server (nonblocking)

• way to process messages received

API design choice

- ① active loop to read incoming messages in app.
⇒ `Message receive();`
- ② react as we're told about messages (event-driven)
⇒ `void messageReceived(Message m);`

①

Client sends m
Client waits for reply

or

Client sends m,
sends m2
Client waits for
reply 1
reply 2

②

Client sends m
⋮

①

AbstractClient

Blocking

Reactive

②

meanwhile,
client handles incoming messages from server

	Vote
①	0
②	8
③	→ 8

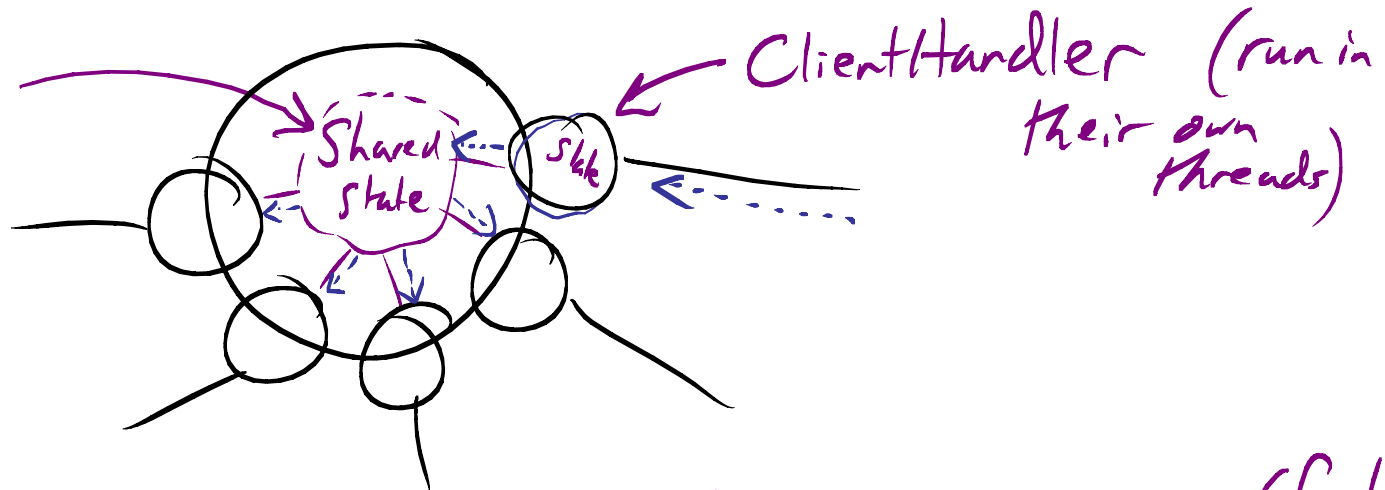
If both are available, using both simultaneously would be awkward: Don't know whether blocking call or msg handler should get the next msg.

Server side:

Server developer will provide:

- mechanism for handling each client

should be
thread
safe



- need a way to create ClientHandler objects

(factory method)

same question

- way of handling messages received

- sending messages

- active? (Runnable?)

Runnable?

- Server Active computation (or just passive?)