Assignment 5
Reliable networking with minisockets

Ari Rabkin

A filesystem postmortem

- It's hard, we know. We aren't out to get you; grading won't be too brutal.
- Late submission until 3am tonight without penalty.
- Also, we'll drop lowest project score
- Remember; if you need extra time, or clarification, ***ask early***.

Project overview

- Datagram networking is ugly:
  - Packets get lost
  - Forces programmer to think about packets (too low level)
- We're going to build reliable data streams, which have neither drawback.
- Closely mimics TCP

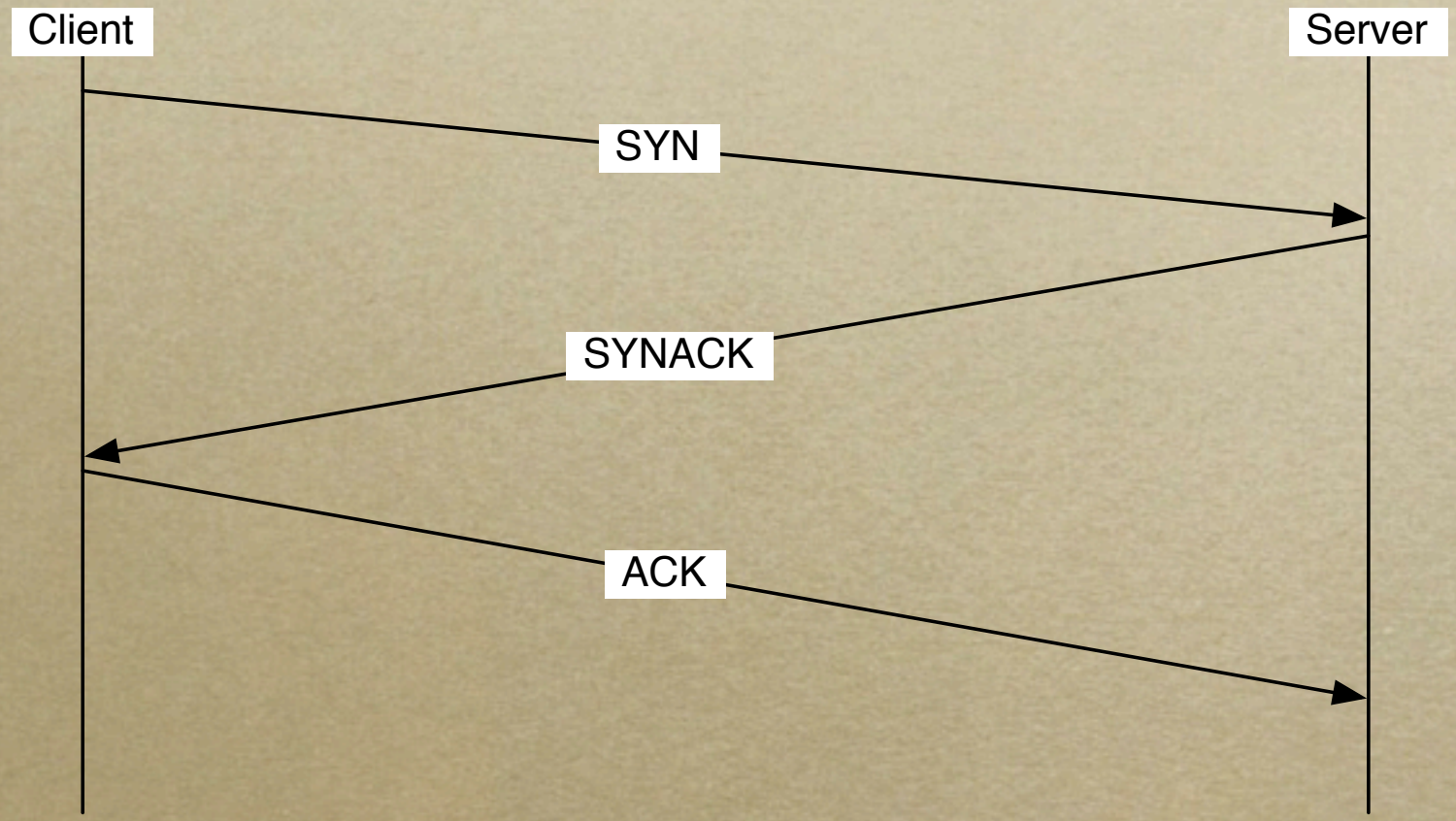
Sockets

- A socket is an endpoint of a reliable communication stream
- Program can read or write any quantity of data; stream interface divides into packets.
- Yours are called minisockets -- see minisocket.h and .c

Server and Client Sockets

- Sockets come in two kinds: server (listening) and client
- Server socket created by *minisocket_server_create*(); call blocks until client connects.
- Clients create sockets with *minisocket_client_create*(), which connects.

Creating a socket



The three-way handshake

Two-way-ness

- Note that after connection setup, the stream is two-way.
- The server and client sockets become interchangeable after the create succeeds.

Closing sockets gracefully

- Either side can send a close message.
- Other side responds with ACK
  - After receiving ACK or timeout, close-initiator can treat socket as closed.
  - Other side waits 15 seconds, then treats it as closed. (why?)

Packet loss

- Networks lose packets for lots of reasons.
- Easy fix: retransmit if needed.
- Catch: what if destination is down?

Using ACKs

- Detecting failure perfectly is impossible. In practice, we do well enough.
- Send a packet with sequence #, wait for ACK to that packet.
- Resend if no ACK received by timeout.

Retransmitting

- Wait 100 ms for ACK, then retransmit
- Retransmit seven times, doubling wait each time after each failure.
- Use alarms and semaphores: note that if a packet arrives, should wake up right away.
- So sleep-with-timeout is no good here.

Special cases

- In the three-way handshake, the SYN and the SYNACK are retransmitted reliably.
- So is the CLOSE on socket teardown.

Packet Formats

- Remember that ‘type’ field in packets from Project 3? Here’s where it comes in handy.
- Need to separate minisocket and minimsg packets.
- Probably also wise to divide socket control and data packets.

Minisocket Packets

- You need SYN, SYNACK, DATA, ACK, CLOSE.

Streams

- Key idea of streams is that the application doesn't see packet boundaries.
- All app sees is a stream of data, some of which is available.
- Stream must be **reliable**, since app can't easily implement reliability again on top.

Read and write

- We're building streams on packets: *minisocket_send*() with a big buffer should split it into packets and send them reliably.
- Likewise, *minisocket_receive*() should return whatever is available, or a full buffer.
- If more data than a buffer is available, can't lose it. (Save it in queue somehow)

The State Machine

- Use the state machine abstraction in designing, and also in implementing.
- Track a state for each socket, and when a packet comes in, switch on socket state and packet type.
- See the TCP state machine (online) for further insight.

Notes

- Thread safety is important here.
- Don’t sleep when you have work to do: need to wake listening thread as soon as packet arrives