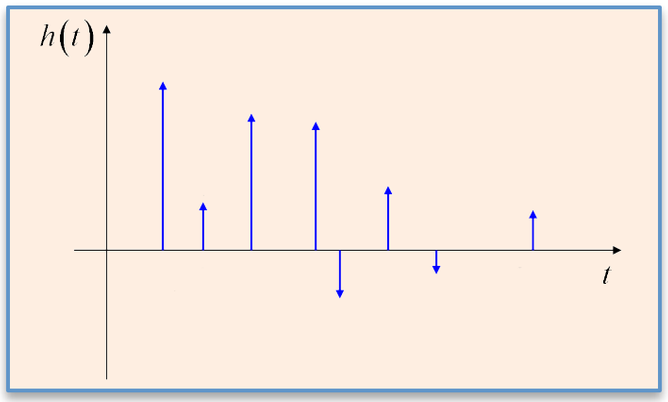
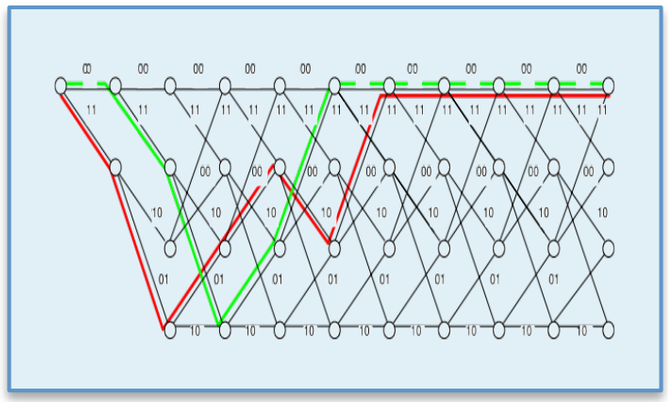
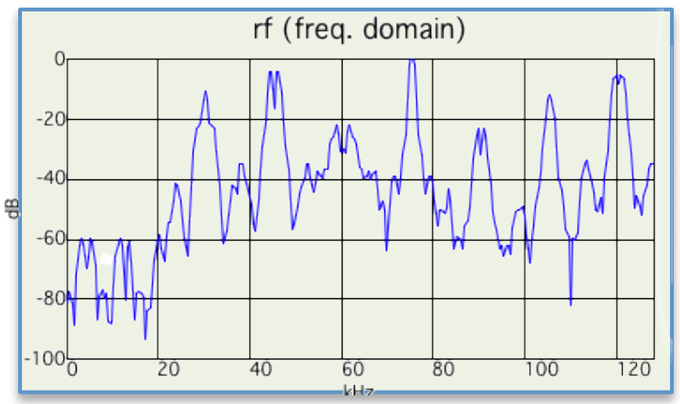
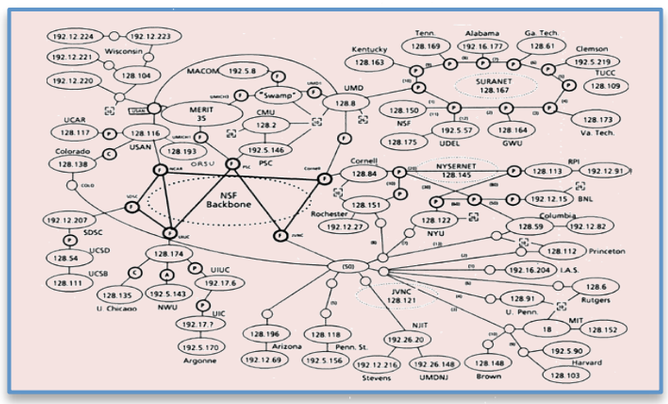








INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2014
Lecture #22

- Reliable transport
  - Stop-and-wait protocol
  - RTT estimation

Unanswered questions

(about packet-switched networks)

~~How do nodes **determine routes** to every other node?~~

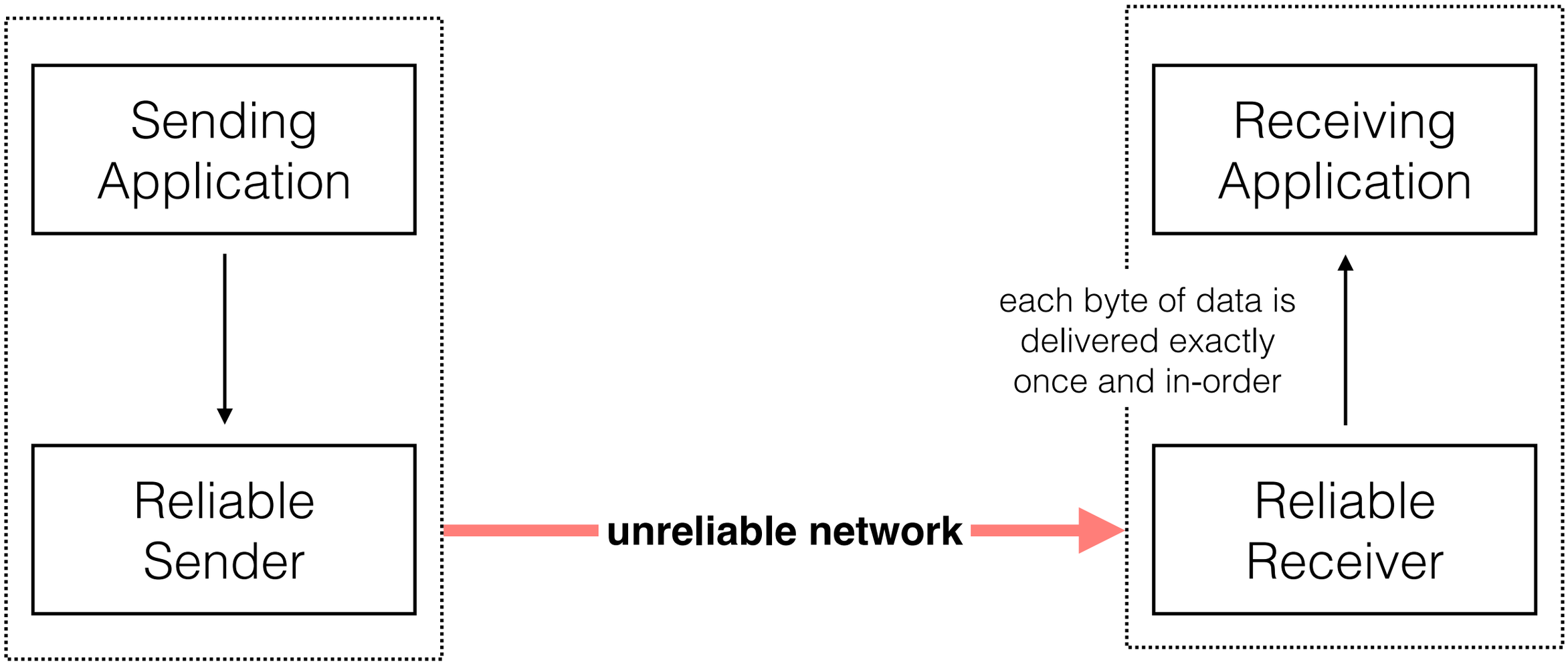
Nodes determine routes via either **link-state**, **distance-vector**, or path-vector routing

~~How do nodes **route around link failures**?~~

Routing protocols will eventually converge, but experience different problems along the way
(routing loops, counting-to-infinity, etc.)

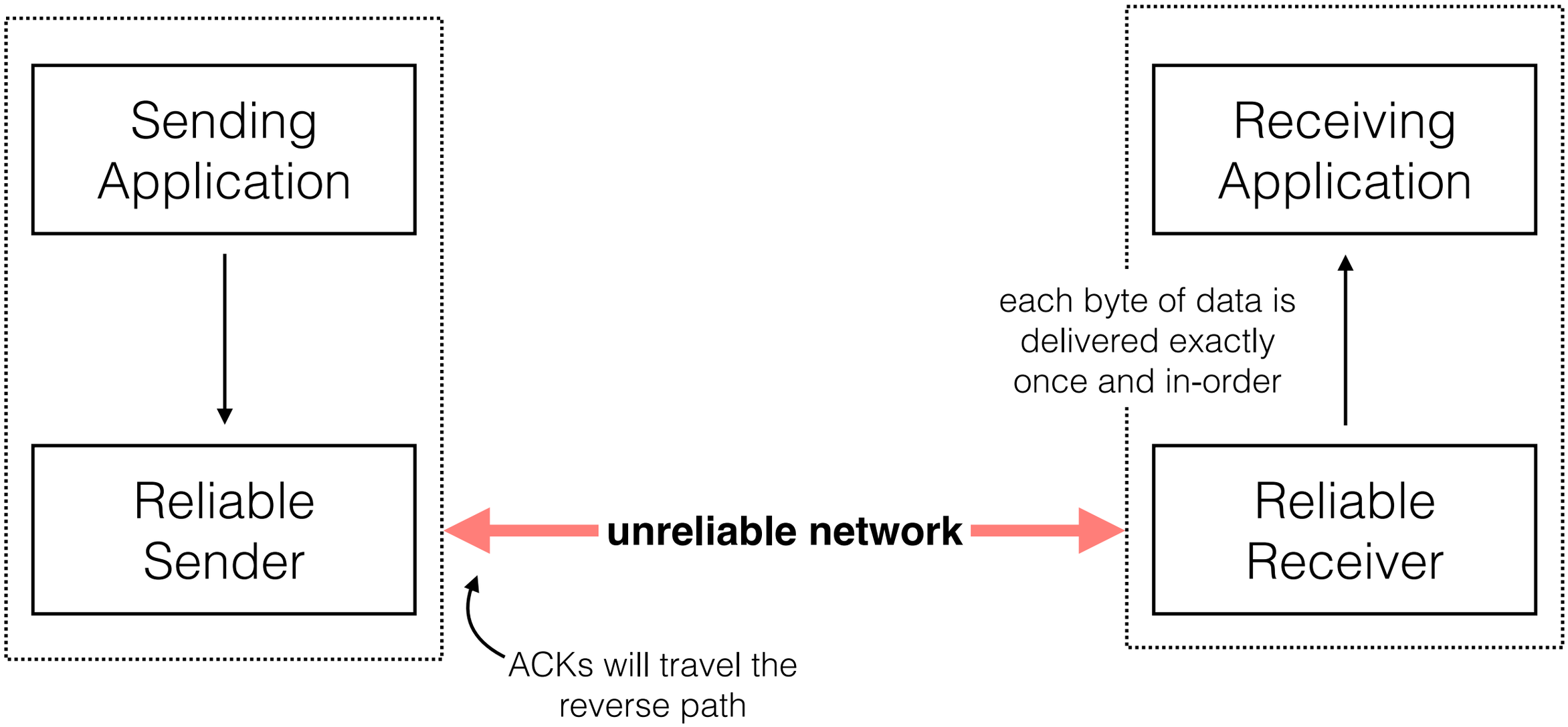
How do nodes **communicate reliably**
given that the network is best-effort?

Reliable Communications



today's goal: develop a reliable transport protocol for the reliable sender and reliable receiver to use

Reliable Communications



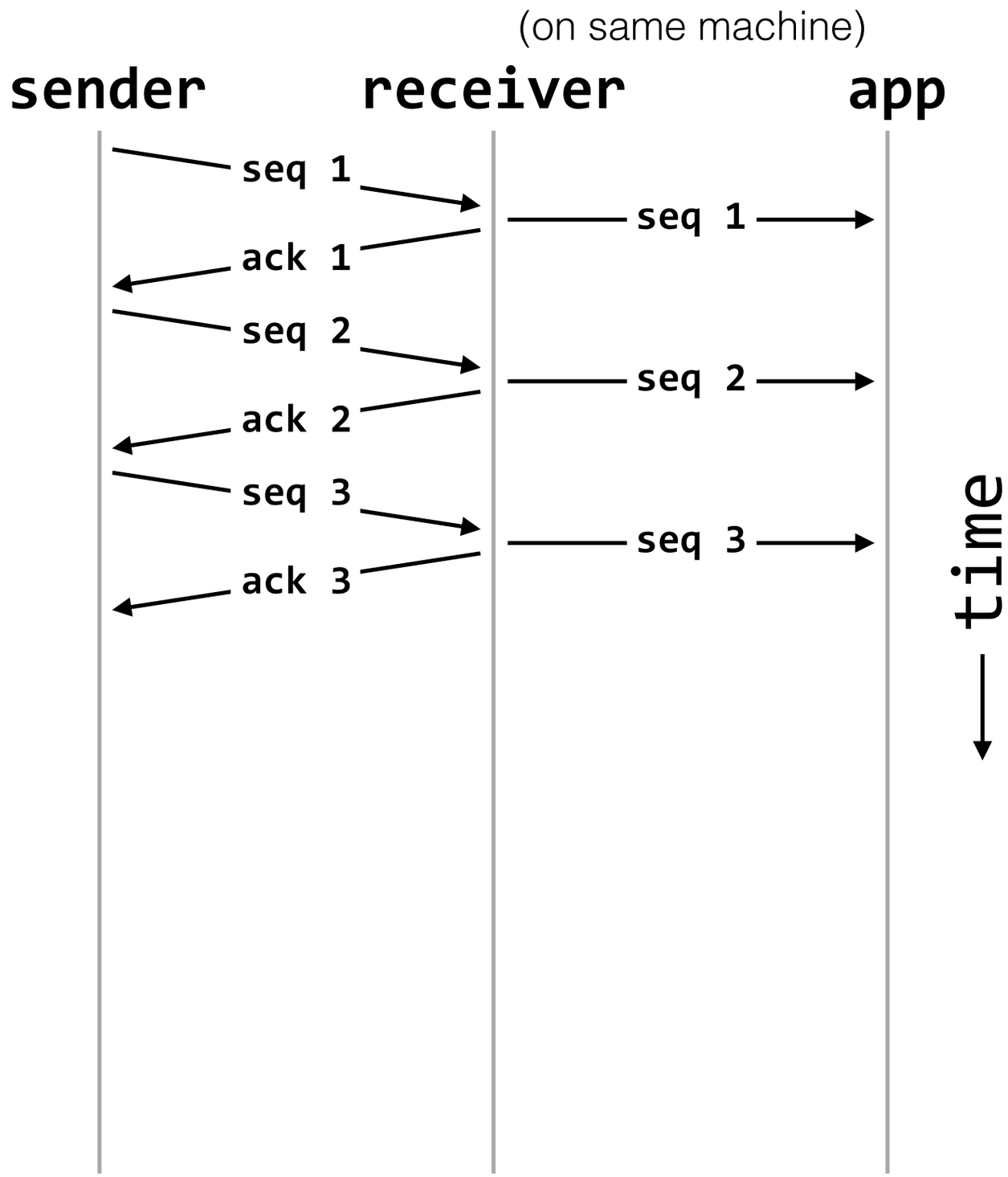
Protocol Attempt #1

At sender:

- Send a packet, keep track of its sequence number
- When an ACK is received for that packet, increment the stored sequence number and repeat

At receiver:

- Upon receipt of packet k, send an ACK for k and deliver the packet to the application



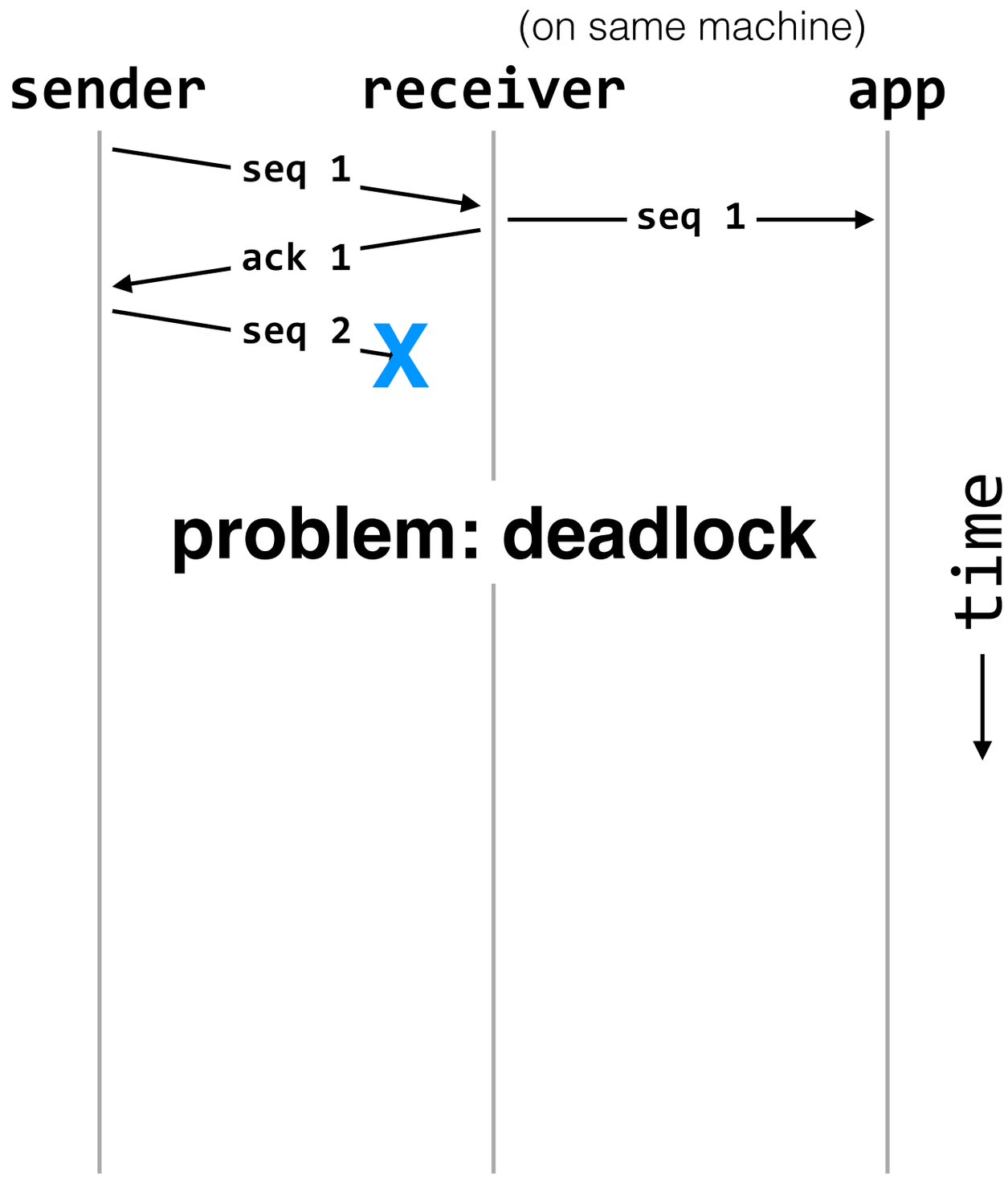
Protocol Attempt #1

At sender:

- Send a packet, keep track of its sequence number
- When an ACK is received for that packet, increment the stored sequence number and repeat

At receiver:

- Upon receipt of packet k, send an ACK for k and deliver the packet to the application



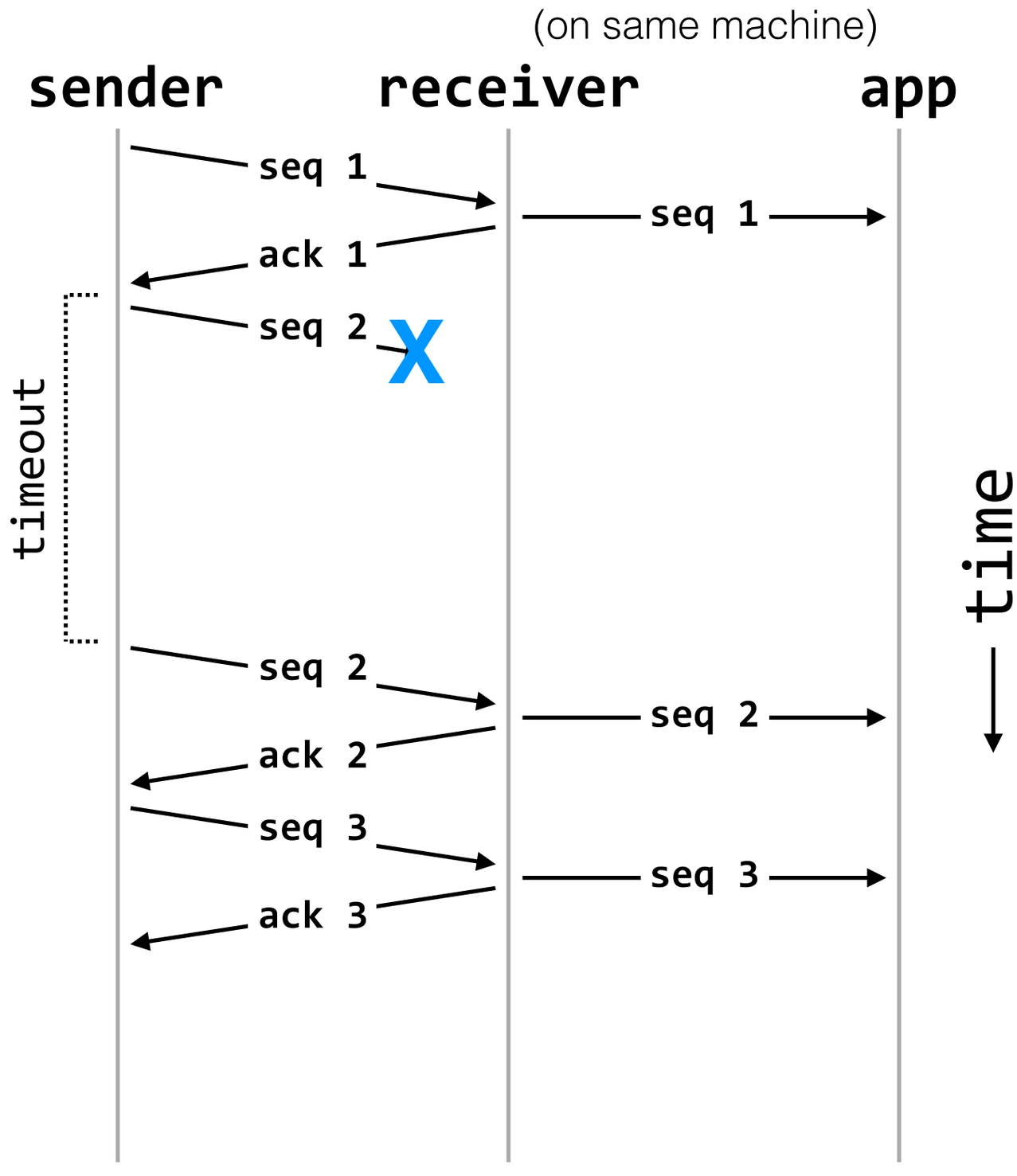
Protocol Attempt #2

At sender:

- Send a packet, keep track of its sequence number
- When an ACK is received for that packet, increment the stored sequence number and repeat
- **If an ACK for the outstanding packet hasn't been received after `timeout` seconds, re-transmit the packet**

At receiver:

- Upon receipt of packet k, send an ACK for k and deliver the packet to the application



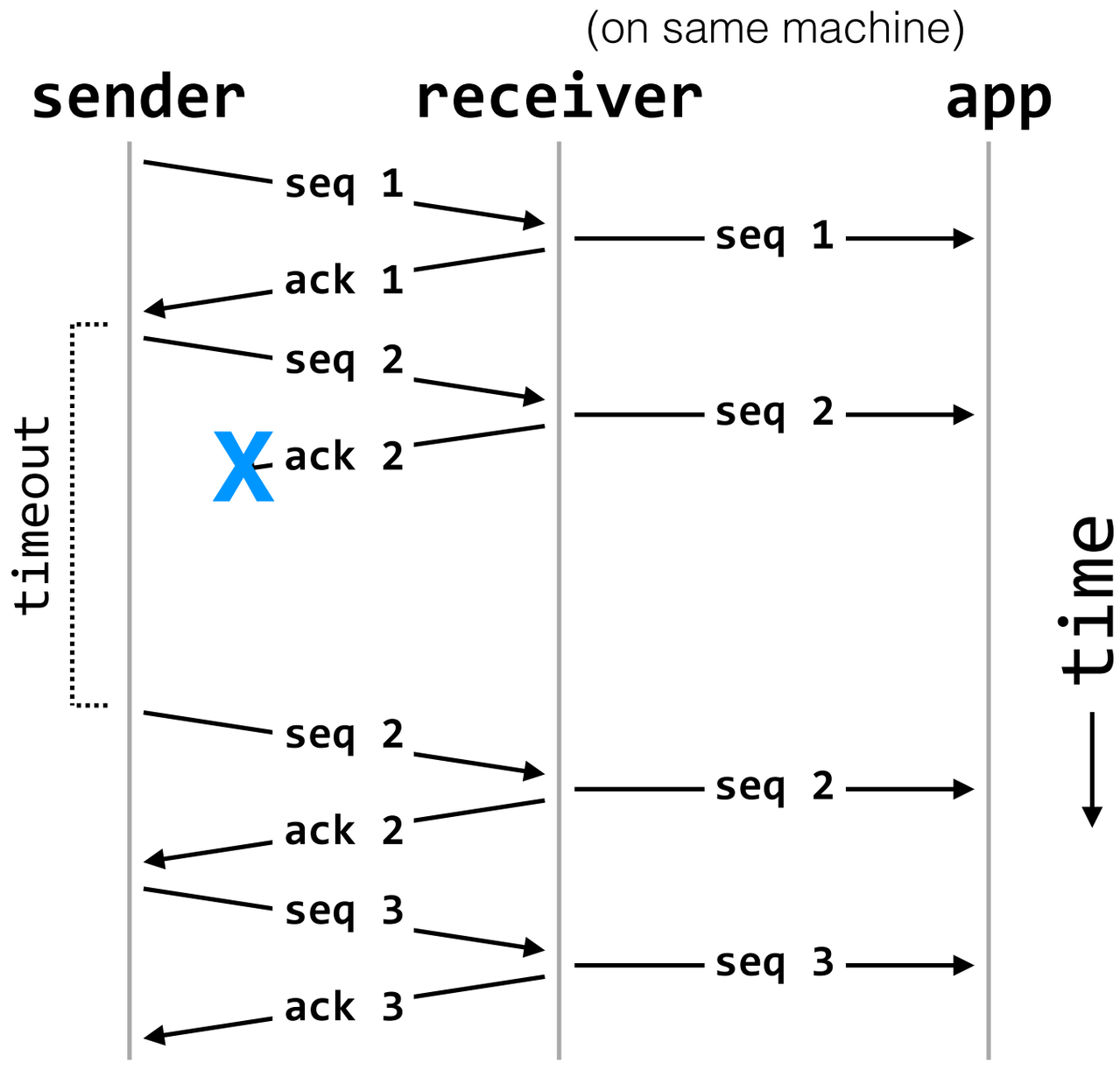
Protocol Attempt #2

At sender:

- Send a packet, keep track of its sequence number
- When an ACK is received for that packet, increment the stored sequence number and repeat
- **If an ACK for the outstanding packet hasn't been received after `timeout` seconds, re-transmit the packet**

At receiver:

- Upon receipt of packet k, send an ACK for k and deliver the packet to the application



problem: this is *at-least once* delivery, not exactly-once

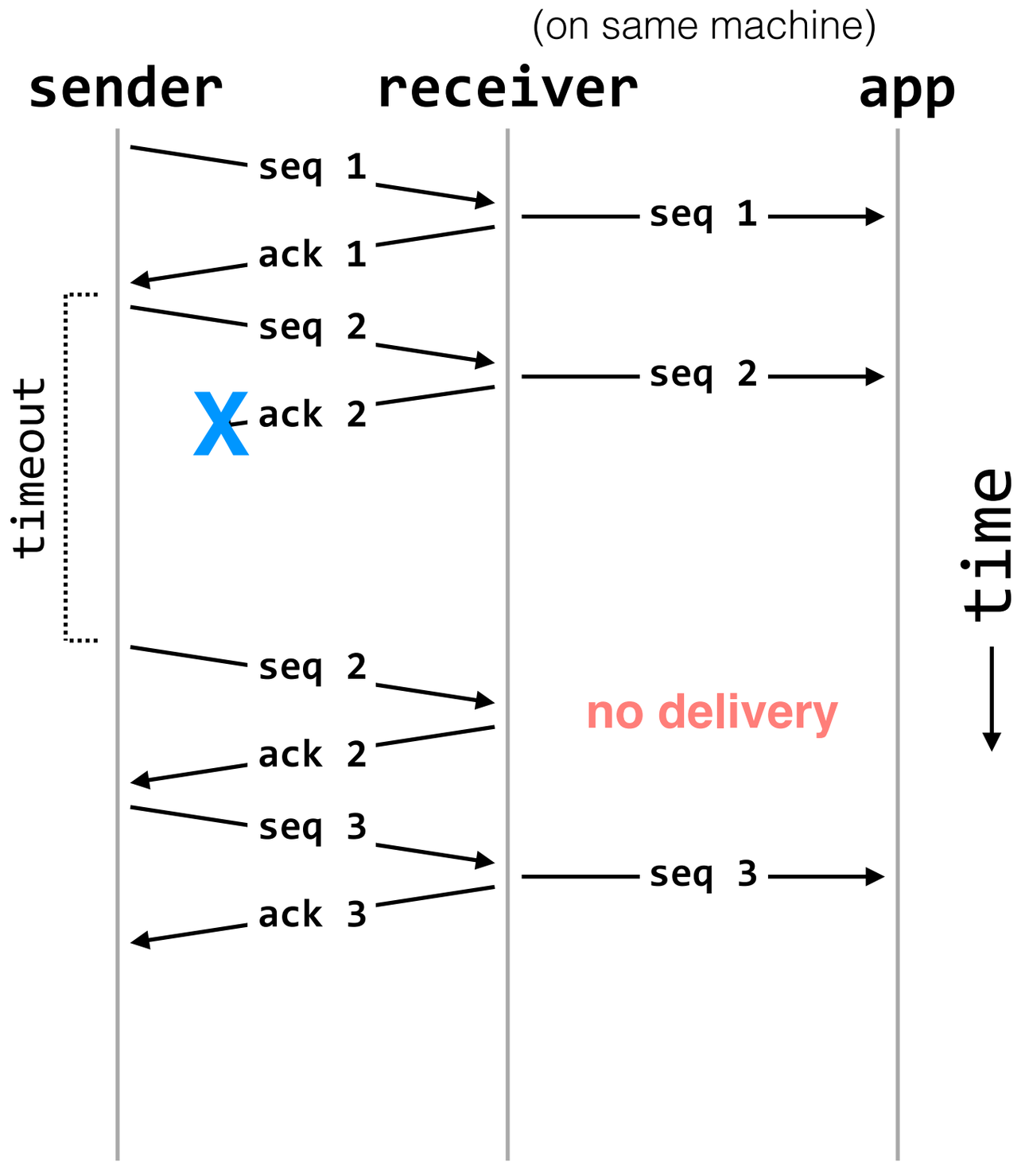
Protocol Attempt #3

At sender:

- Send a packet, keep track of its sequence number
- When an ACK is received for that packet, increment the stored sequence number and repeat
- If an ACK for the outstanding packet hasn't been received after `timeout` seconds, re-transmit the packet

At receiver:

- Upon receipt of packet k, send an ACK for k
- **If k is greater than the last received sequence number, deliver packet to app**



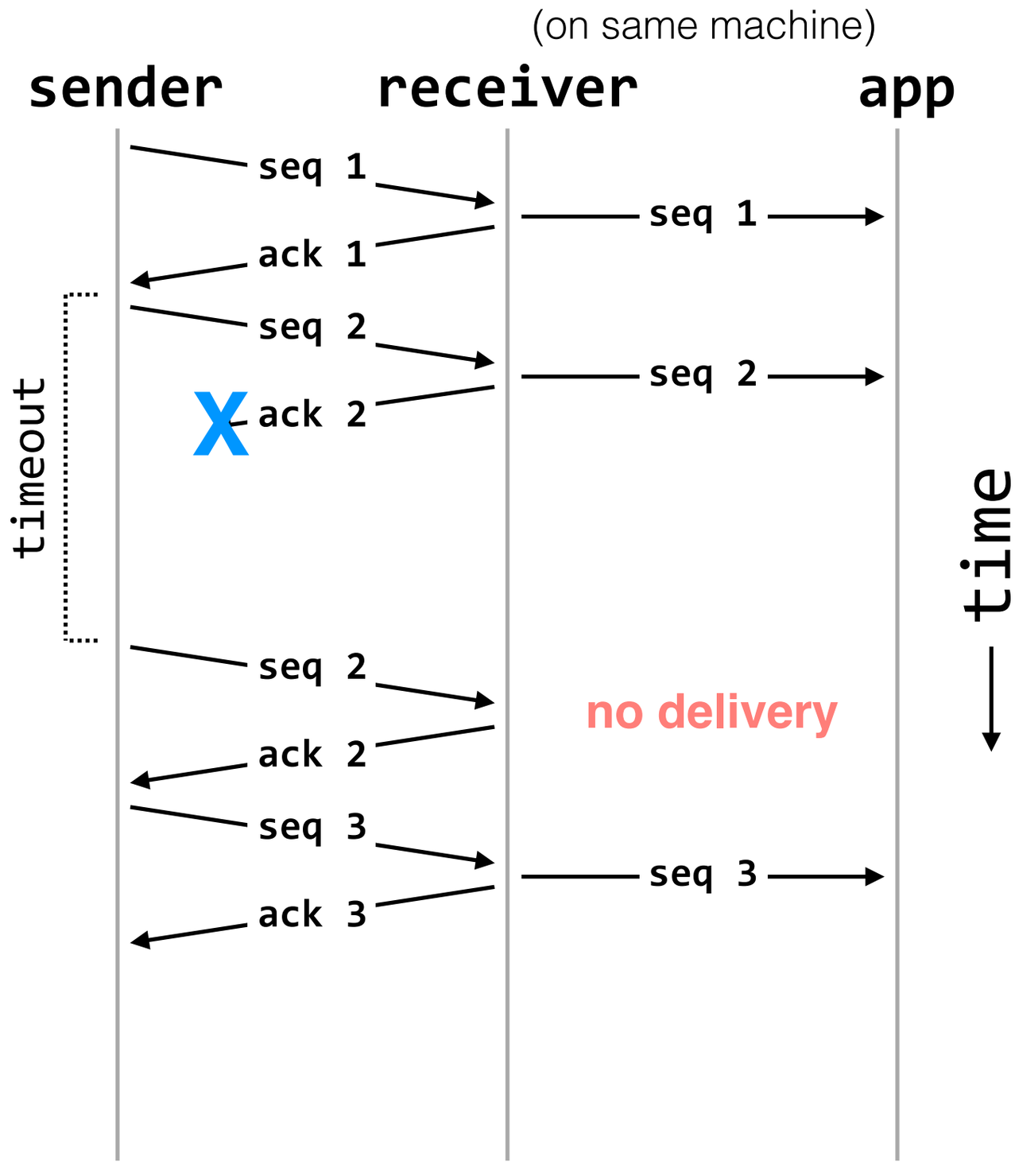
Stop-and-wait Protocol

At sender:

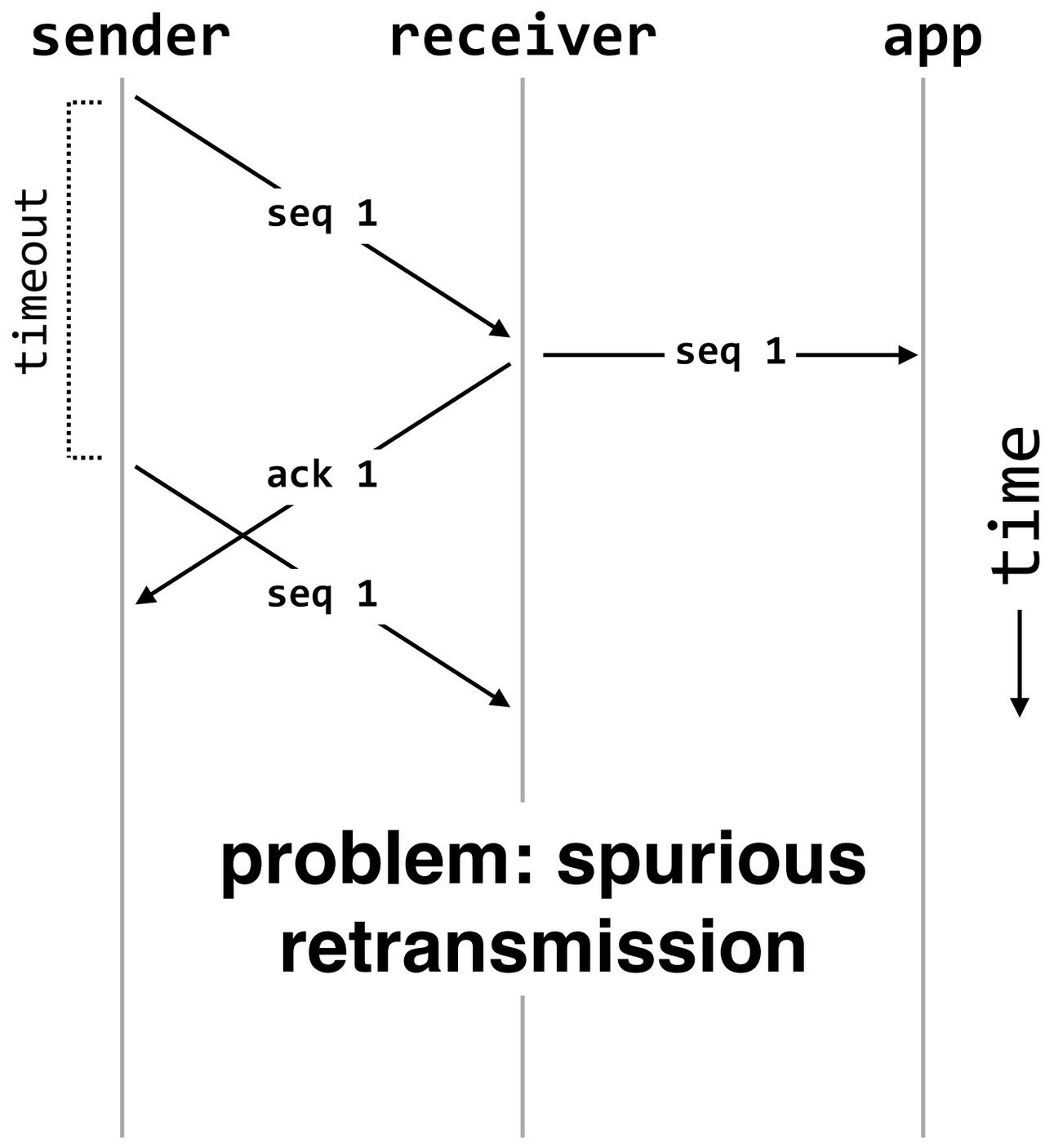
- Send a packet, keep track of its sequence number
- When an ACK is received for that packet, increment the stored sequence number and repeat
- If an ACK for the outstanding packet hasn't been received after `timeout` seconds, re-transmit the packet

At receiver:

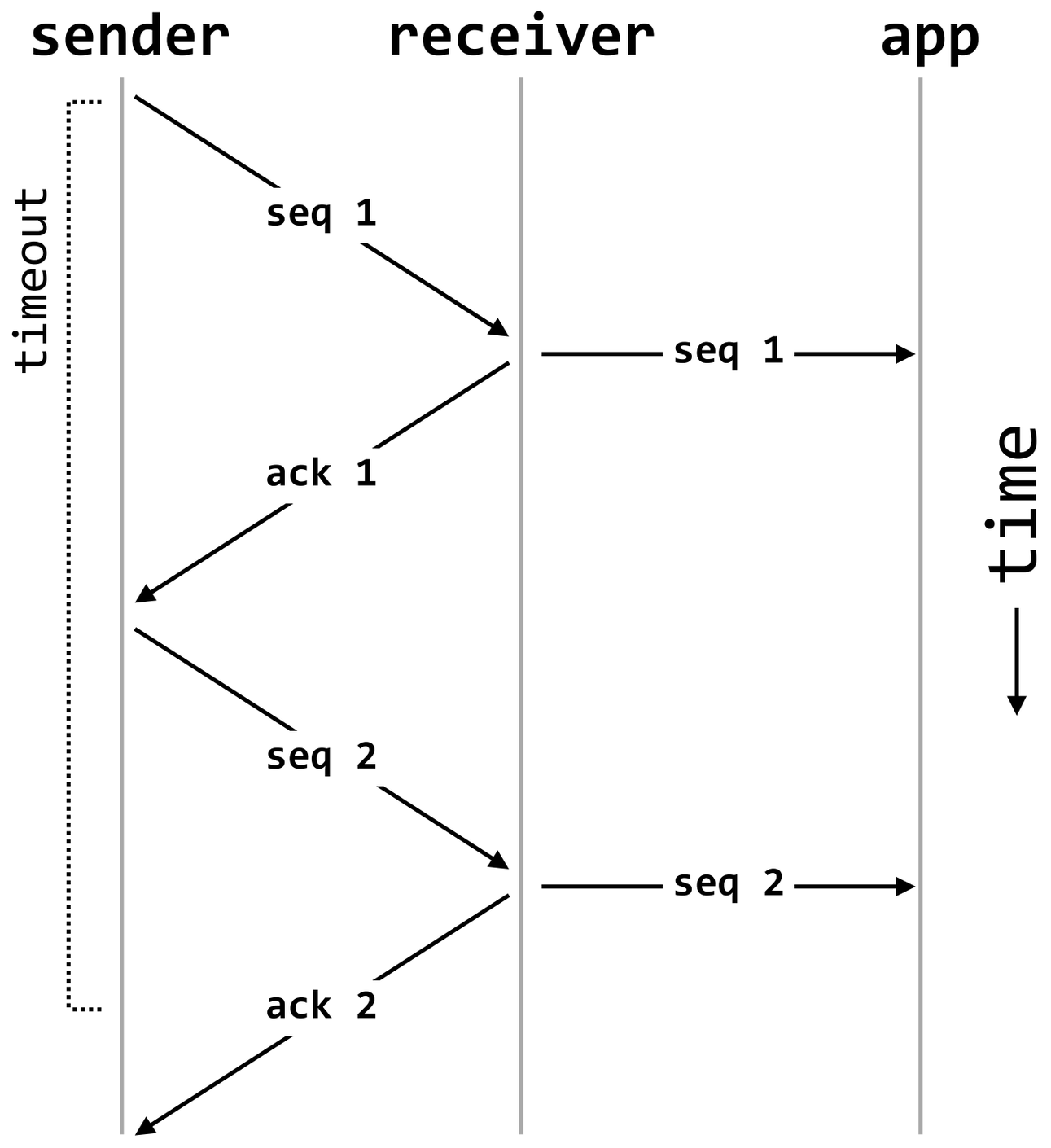
- Upon receipt of packet k, send an ACK for k
- If k is greater than the last received sequence number, deliver packet to app



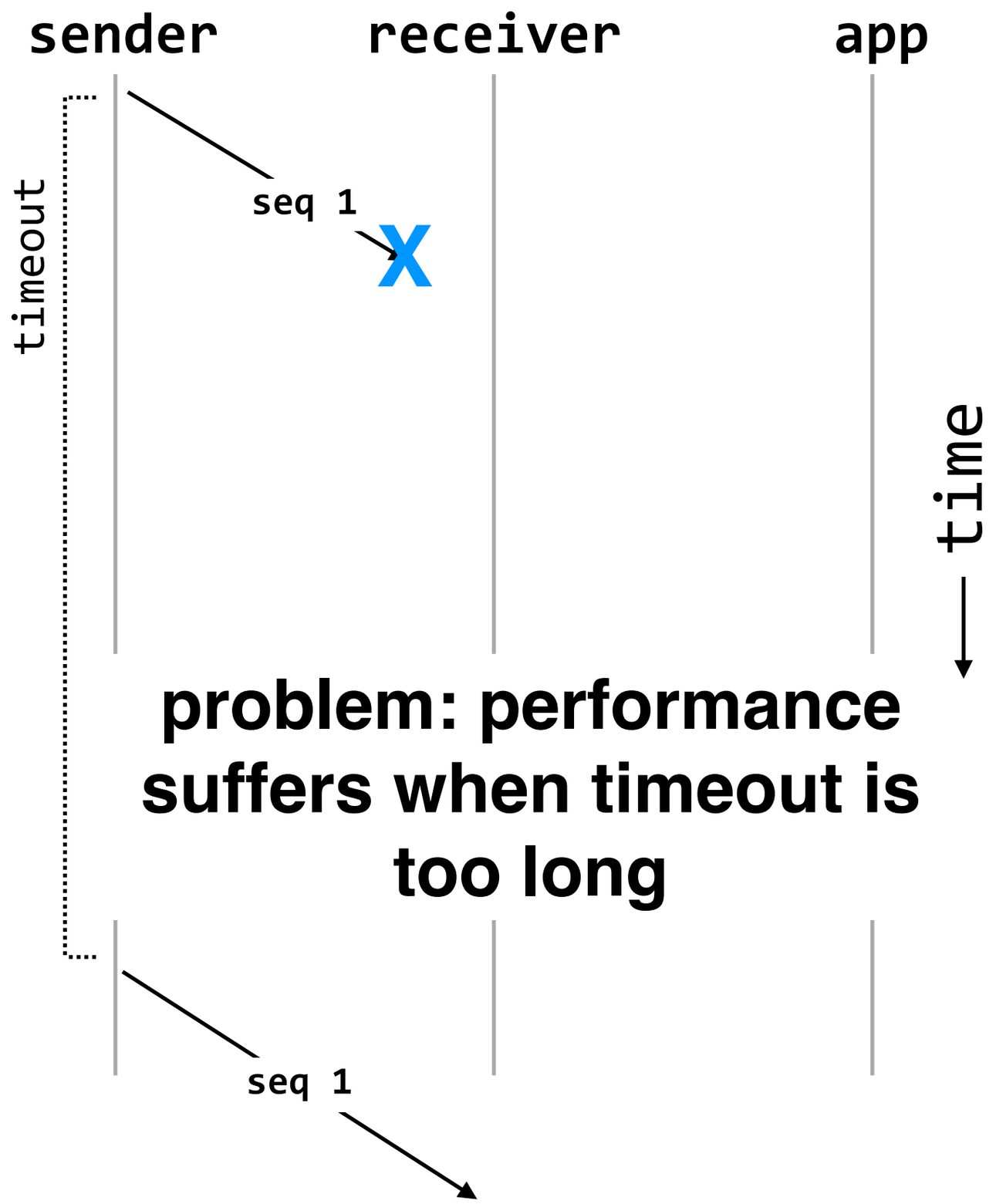
Setting Timeouts



Setting Timeouts



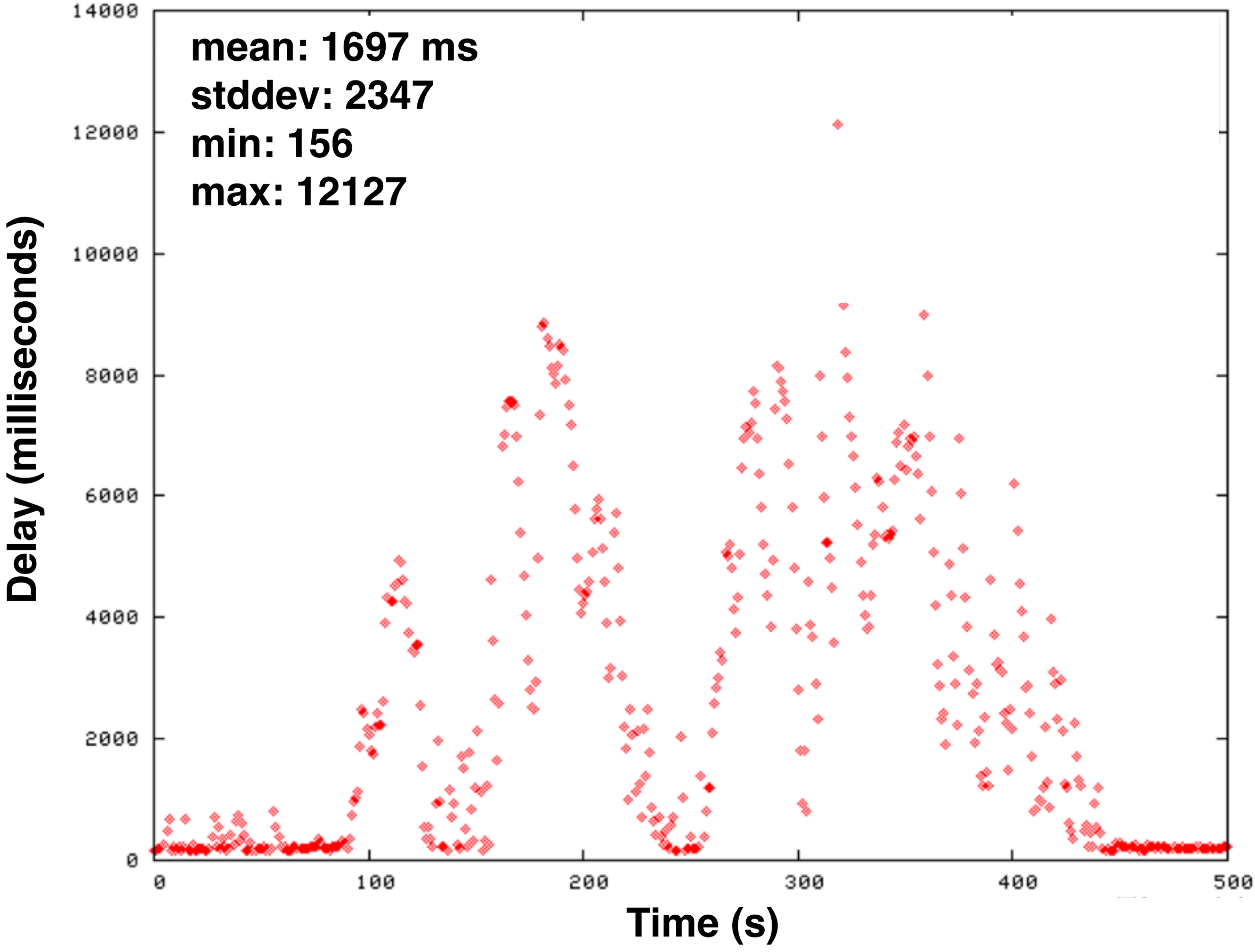
Setting Timeouts



goal: estimate the network delay so that we can set the timeout appropriately

problem: network delay changes over time

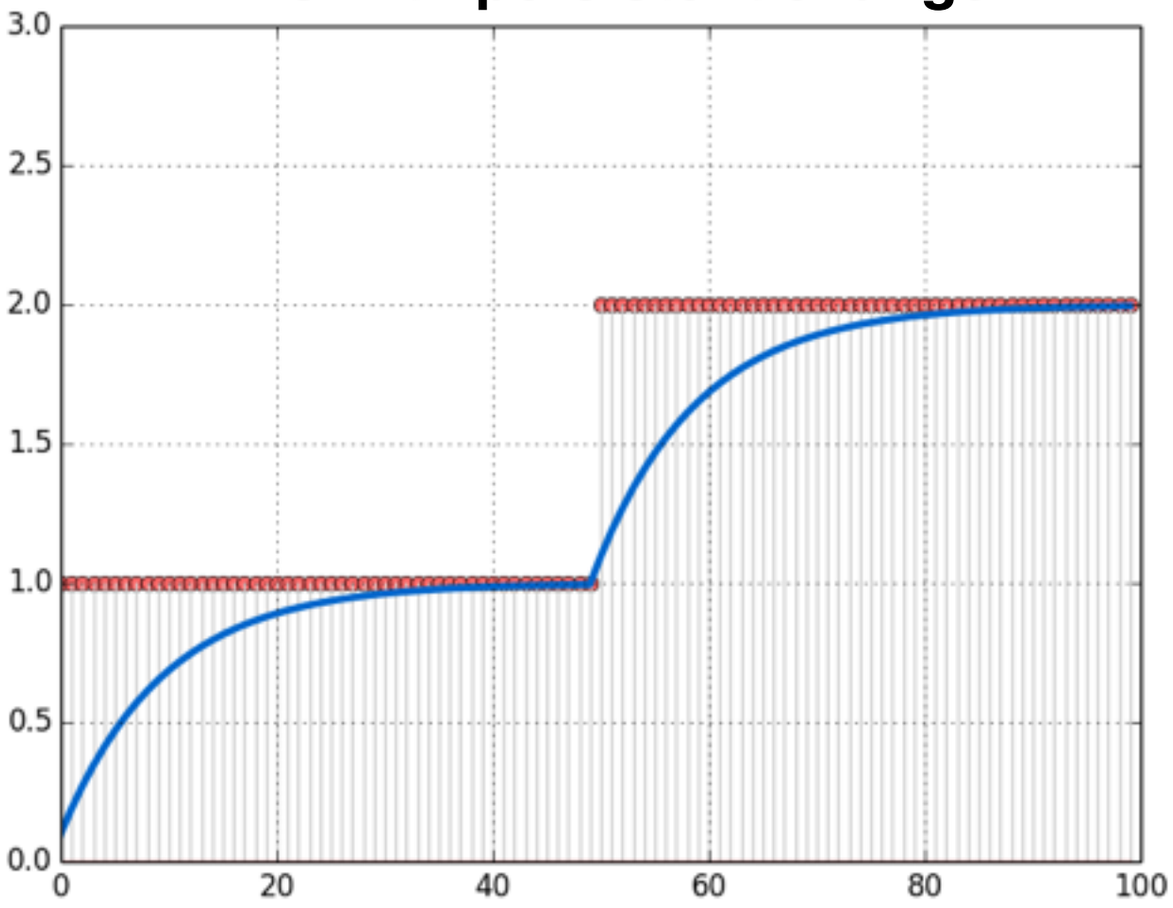
ATT wireless over 3G (from 2012)



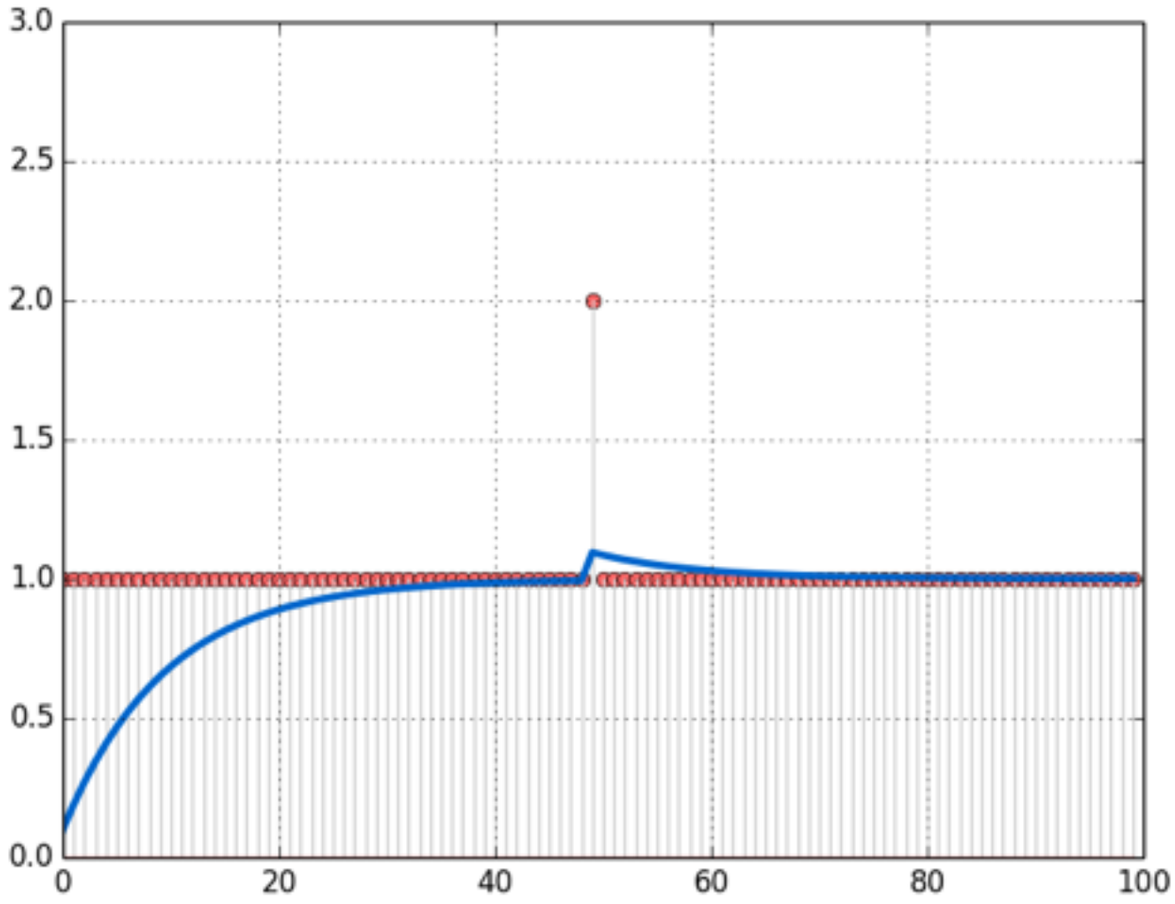
goal: estimate the network delay so that we can set the timeout appropriately. whatever method we use should react to persistent changes and ignore outliers

Setting α

RTTs with persistent change



RTTs with one outlier

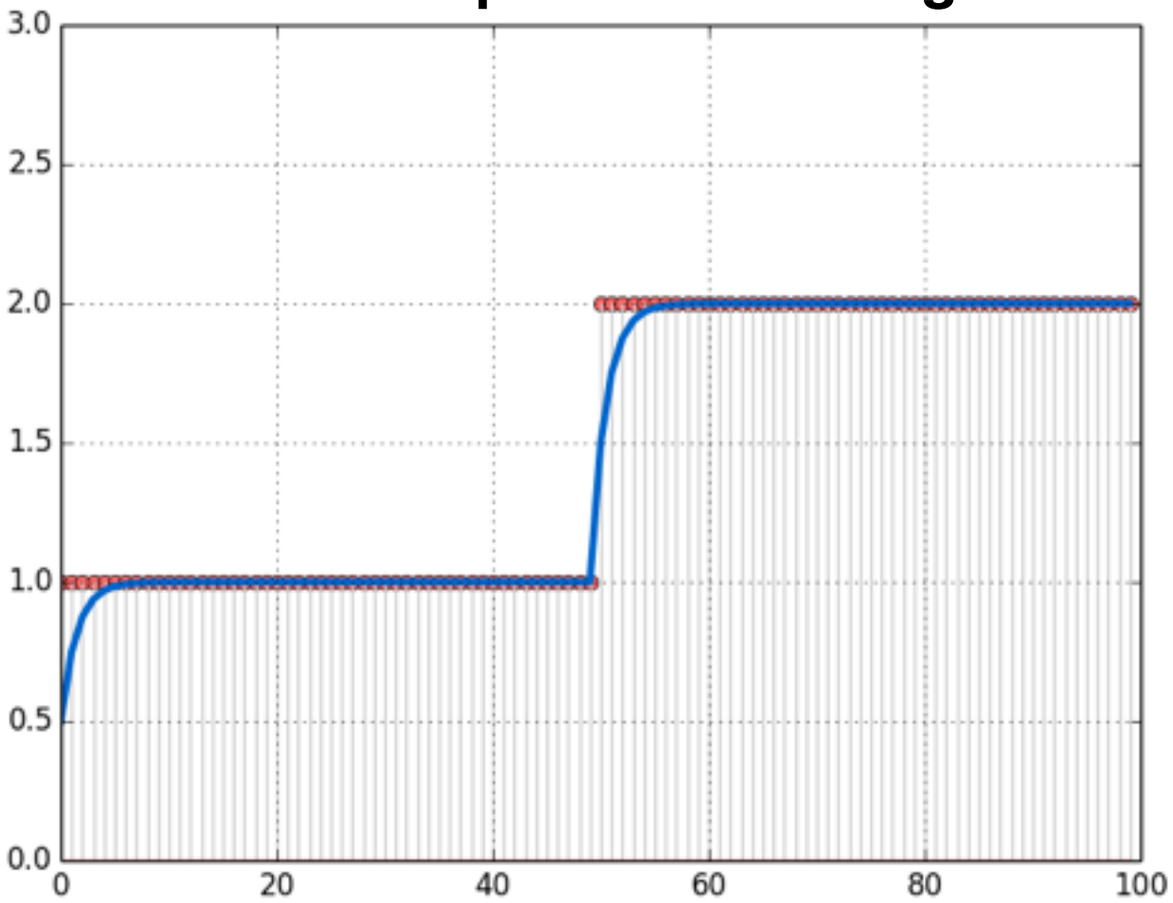


doesn't react quickly
to persistent change

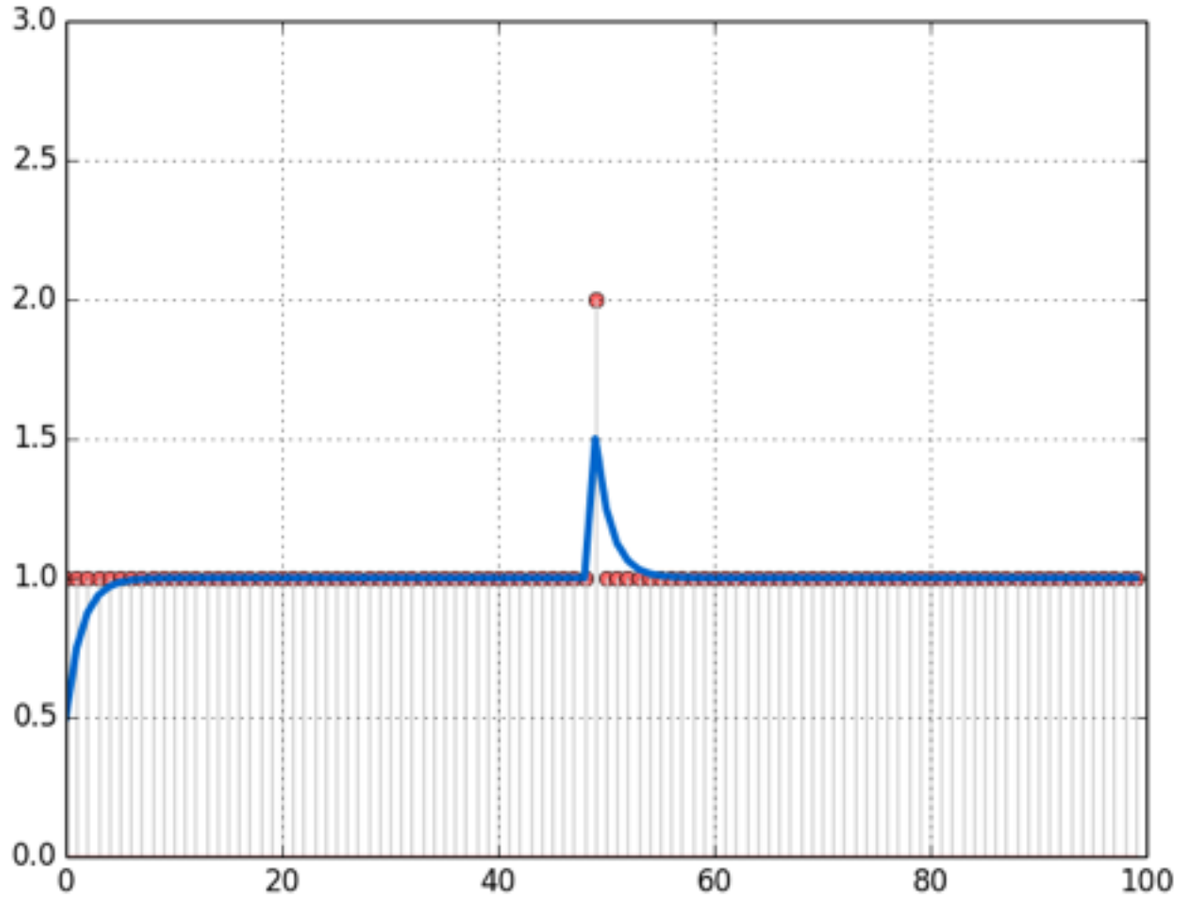
$$\alpha = .1$$

Setting α

RTTs with persistent change



RTTs with one outlier



reacts too quickly to outliers

$$\alpha = .5$$

Timeout Estimation

```
alpha, beta are floats in [0, 1]
k is a small integer

srtt[n] = alpha * rtt_sample + (1 - alpha) * srtt[n-1]
dev_sample = |rtt_sample - srtt|
srttdev[n] = beta * dev_sample + (1 - beta) * srttdev[n-1]
timeout = srtt[n] + k*srttdev[n]
```

we will solve this problem in the next lecture

- **Stop-and-wait protocol**
  Uses sequence numbers, acknowledgements, and timeouts to ensure exactly-once delivery; results in **poor utilization** in many environments

- **RTT estimation**
  Used to set timeout correctly; tricky because RTTs vary both over short timescales and long timescales