recall: sockets

open connection then ...
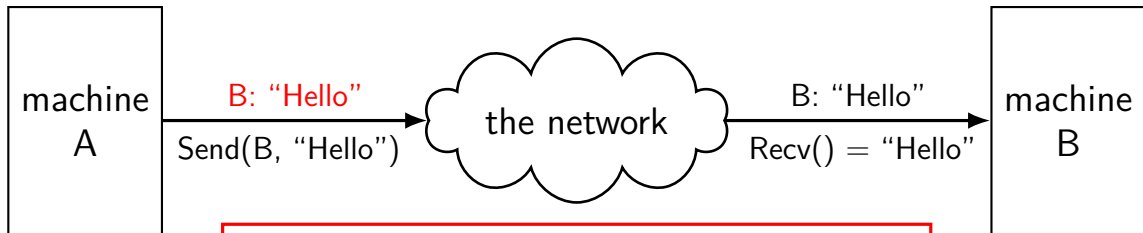
read+write just like a terminal file

doesn't look like individual messages

"connection abstraction"

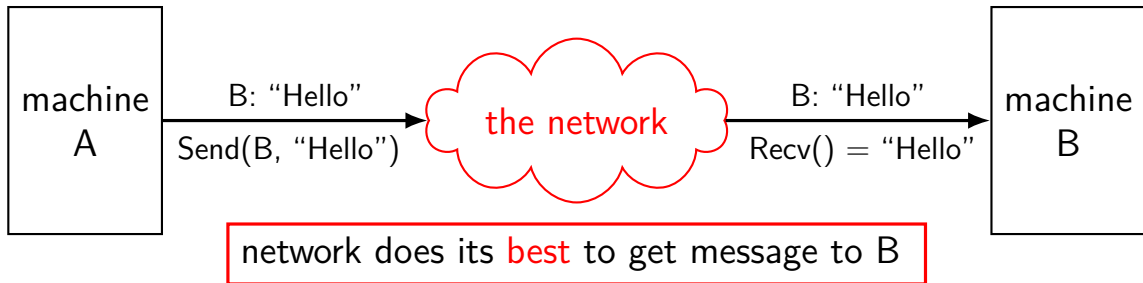# mailbox model

mailbox abstraction: send/receive messages

machine A — B: "Hello" / Send(B, "Hello") → the network → B: "Hello" / Recv() = "Hello" → machine B

A sends "letter" to B
"envelope" tells network it's addressed to B
data in this example: "Hello"

mailbox model

mailbox abstraction: send/receive messages
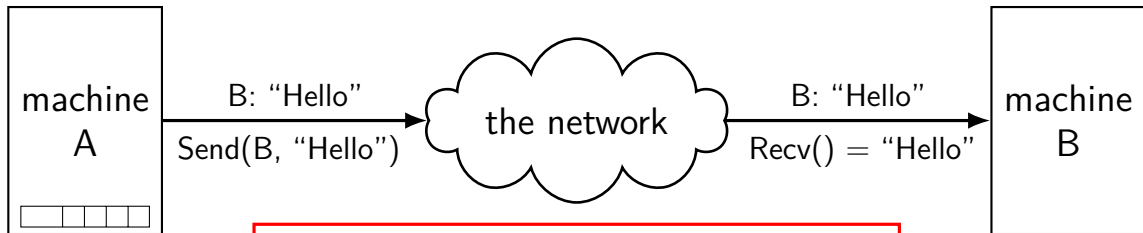

network does its **best** to get message to B

mailbox model

mailbox abstraction: send/receive messages

mailbox model

mailbox abstraction: send/receive messages

machine A — B: "Hello" / Send(B, "Hello") → the network → B: "Hello" / Recv() = "Hello" → machine B
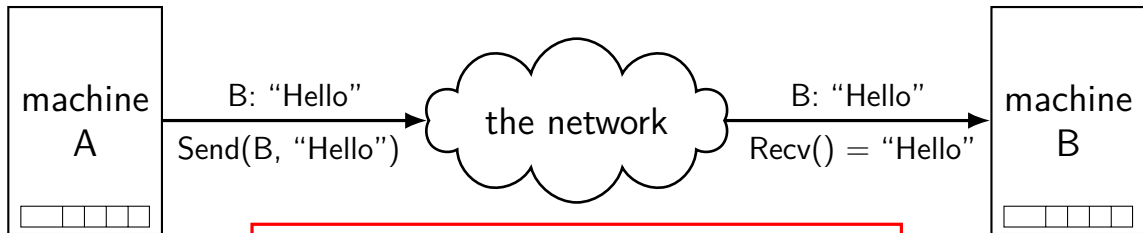
queue ('incoming mailbox') of messages not yet received by receiving program

connections over mailboxes

real Internet: mailbox-style communication

send "letters" (packets) to particular mailboxes

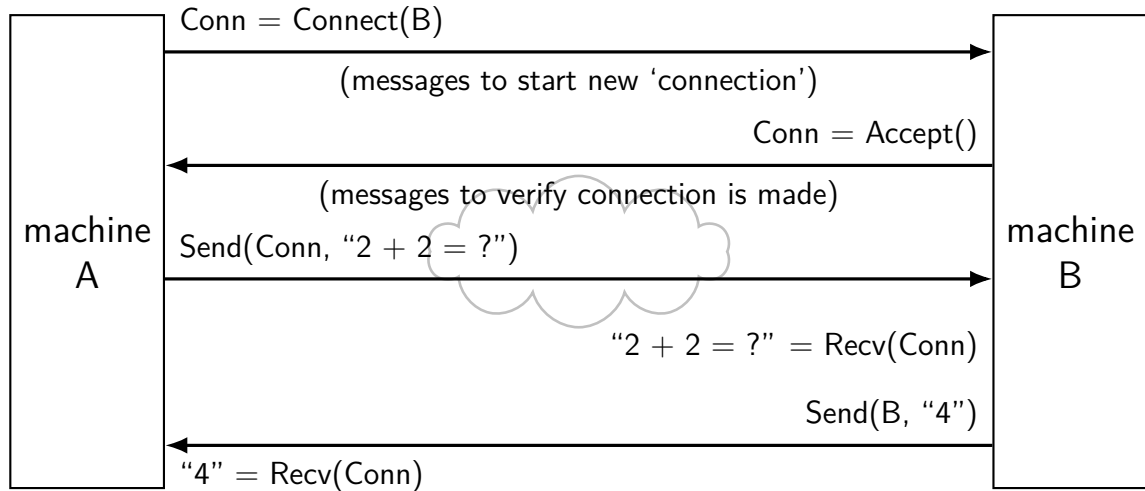have "envelope" (header) saying where they go

"best-effort"

no gaurentee on order, when received

no gaurentee on *if* received

sockets implemented on top of this

conections

machine A

machine B

Conn = Connect(B)

(messages to start new 'connection')

Conn = Accept()

(messages to verify connection is made)

Send(Conn, "2 + 2 = ?")

"2 + 2 = ?" = Recv(Conn)

Send(B, "4")

"4" = Recv(Conn)

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

network limitations/failures

messages lost

messages delayed/reordered

messages limited in size

messages corrupted

network limitations/failures
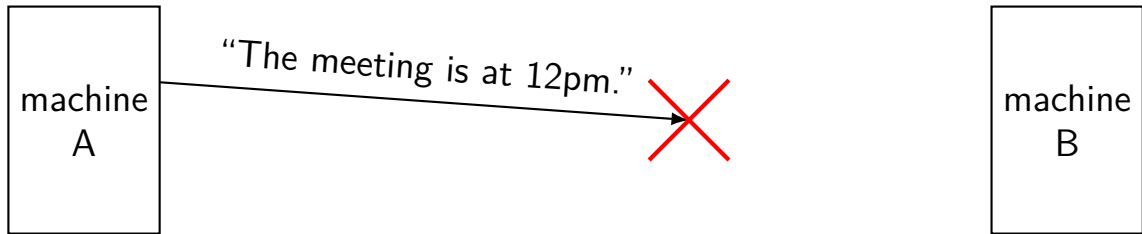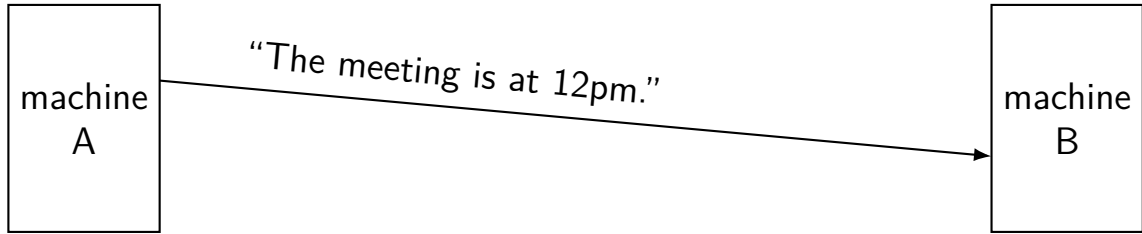
messages lost

messages delayed/reordered

messages limited in size

messages corrupted

dealing with network message lost

handling lost message: acknowledgements

handling lost message

machine
A

"The meeting is at 12pm."

machine
B

handling lost message

machine A

"The meeting is at 12pm."

"timeout"
A doesn't get reply
after waiting too long

machine B

handling lost message

machine A → machine B: "The meeting is at 12pm." ✗

"timeout"
A doesn't get reply
after waiting too long

machine A → machine B: "The meeting is at 12pm."

machine B → machine A: Got it!

11
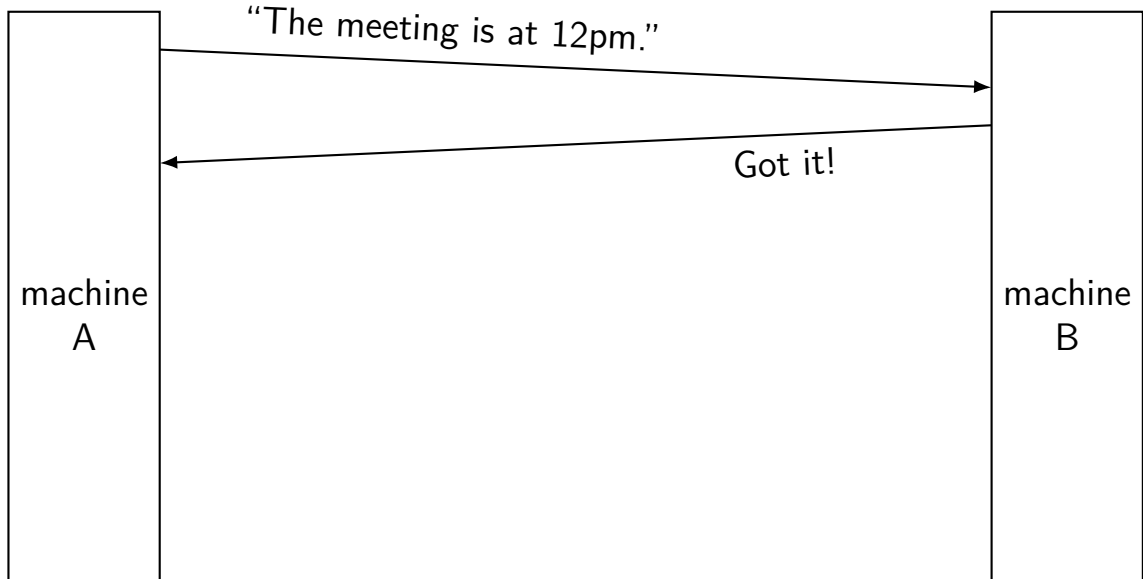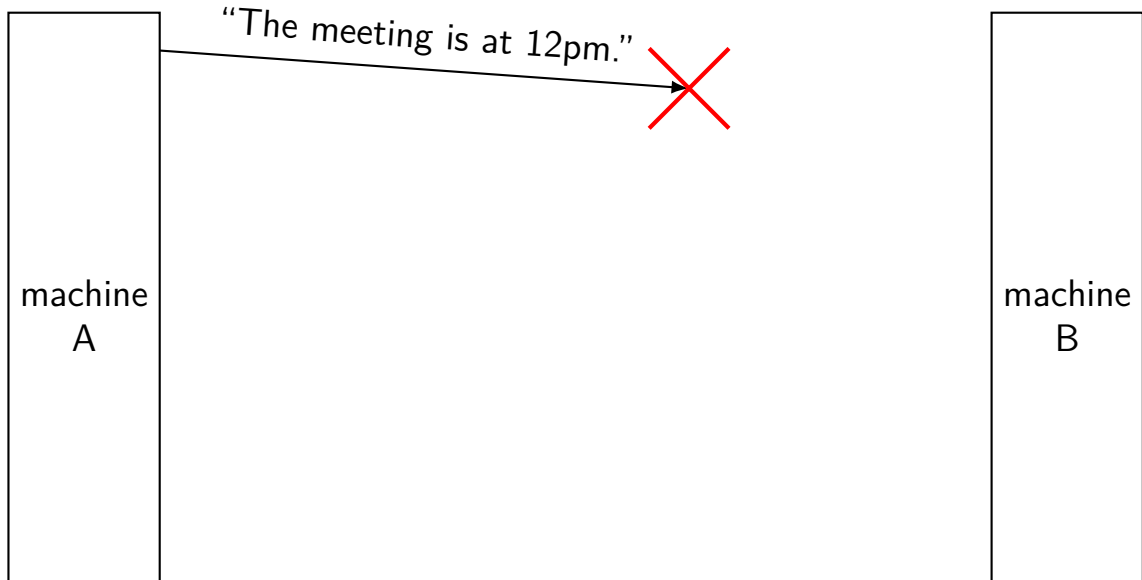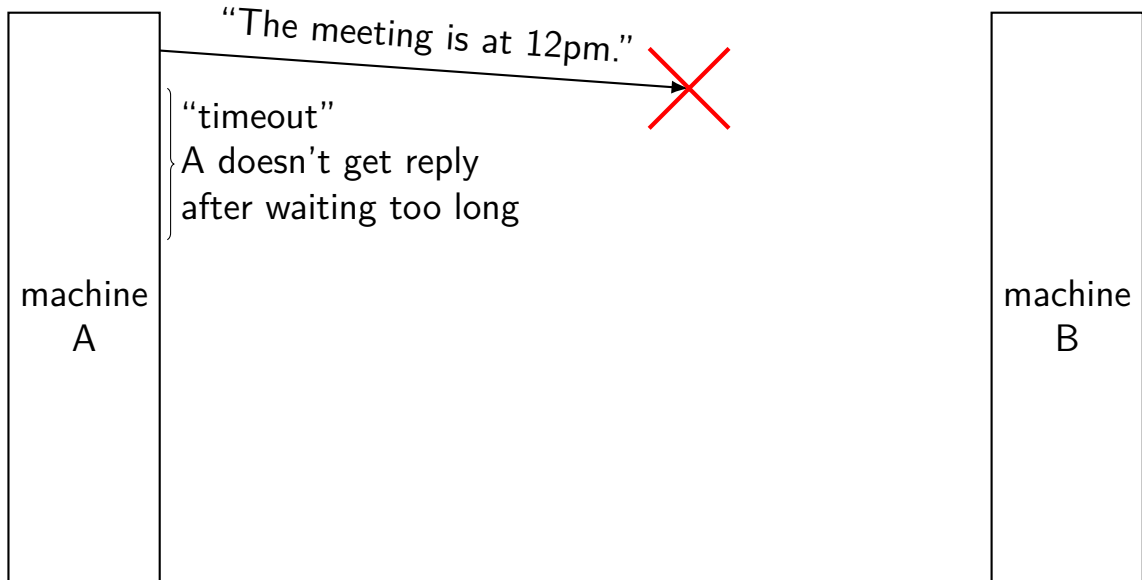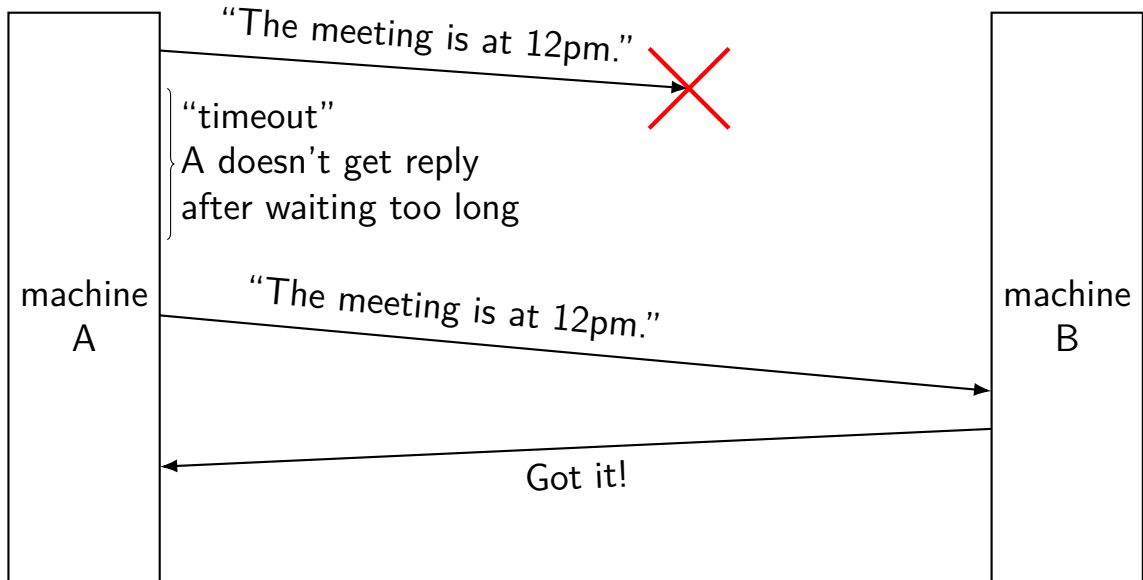
exercise: lost acknowledgement

"The meeting is at 12pm."

machine
A
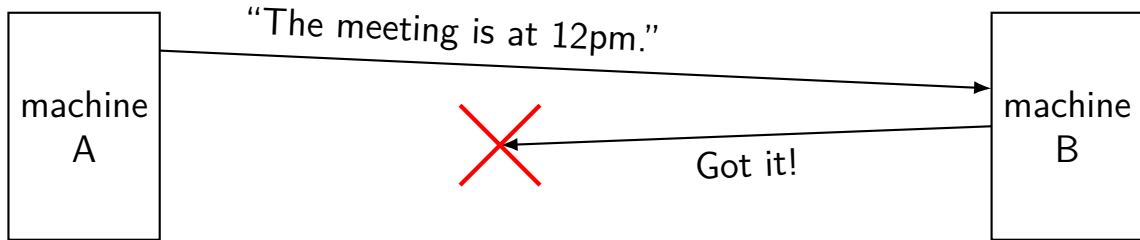
machine
B

Got it!

exercise: how to fix this?
  A. machine A needs to send "Got 'got it!' "
  B. machine B should resend "Got it!" on its own
  C. machine A should resend the original message on its own
  D. none of these

answers

send "Got 'got it!' "?
> same problem: Now send 'Got Got Got it'?

resend "Got it!" own its own?
> how many times? — B doesn't have that info

resend original message?
> yes!
> as far as machine A can be, *exact same situation* as losing original message

lost acknowledgements
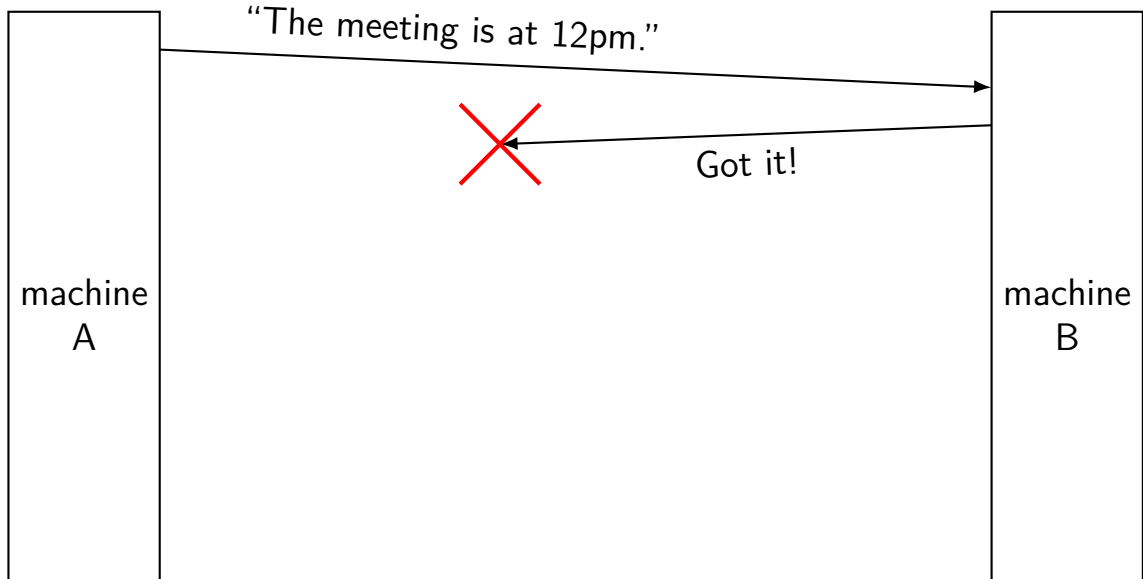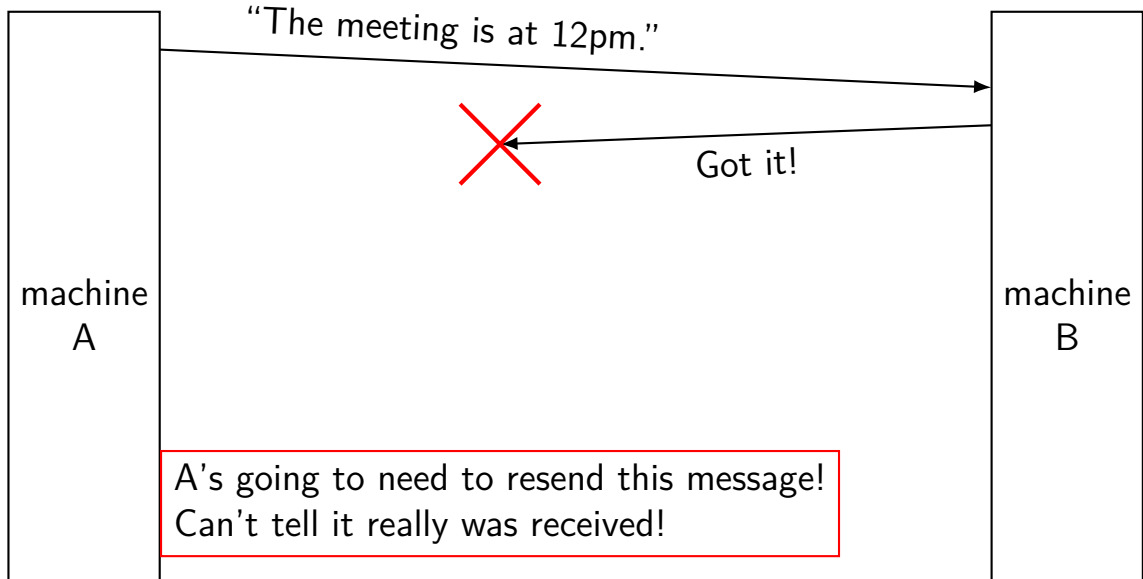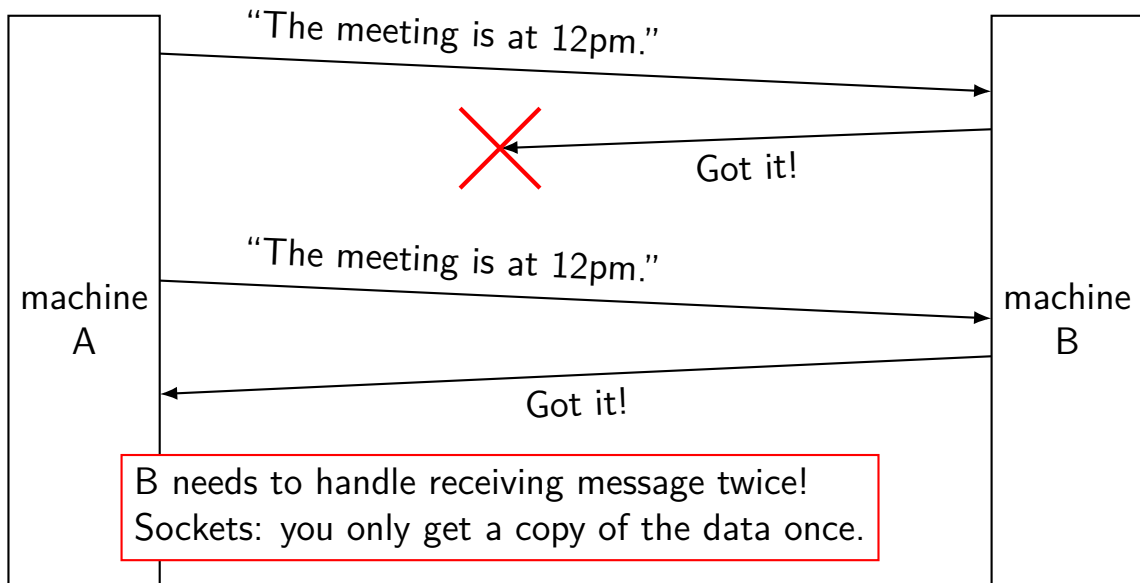
"The meeting is at 12pm."

Got it!

machine A

machine B

lost acknowledgements

"The meeting is at 12pm."

Got it!

machine A

machine B

A's going to need to resend this message!
Can't tell it really was received!

lost acknowledgements

"The meeting is at 12pm."

Got it!

machine A

"The meeting is at 12pm."

machine B

Got it!

B needs to handle receiving message twice!
Sockets: you only get a copy of the data once.

network limitations/failures

messages lost

messages delayed/reordered

messages limited in size

messages corrupted

delayed message

delayed message
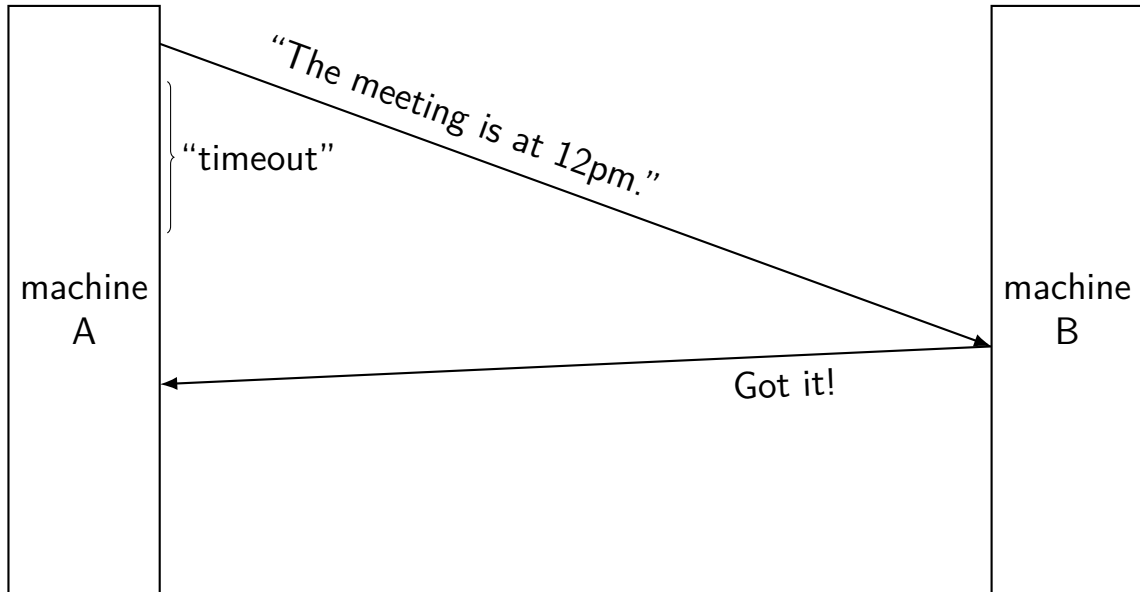
delayed message
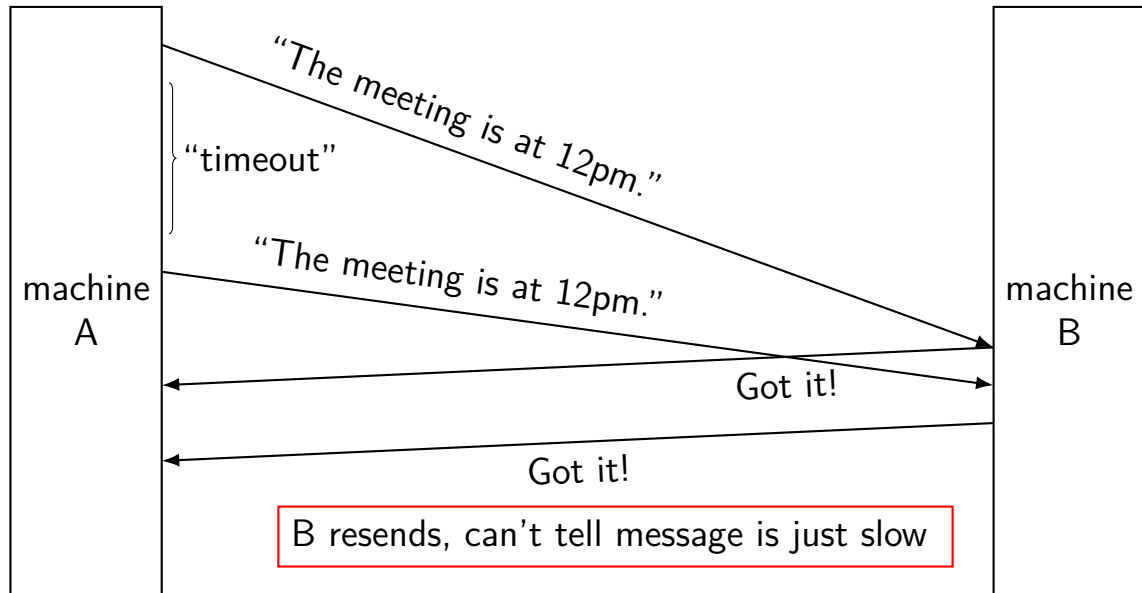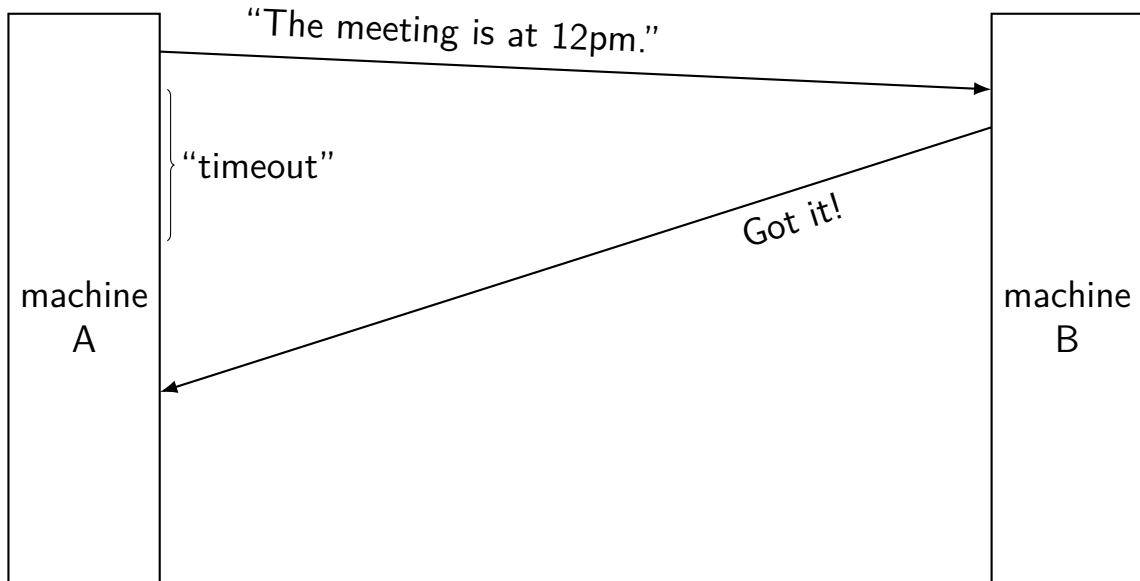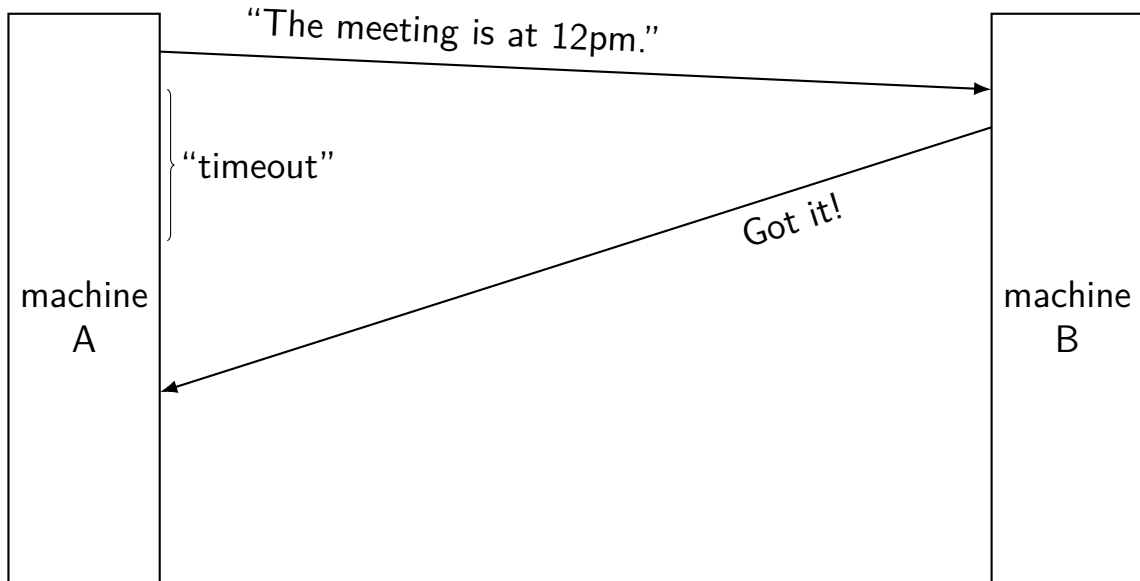
machine A → machine B: "The meeting is at 12pm."

"timeout"

machine A → machine B: "The meeting is at 12pm."

machine B → machine A: Got it!

machine B → machine A: Got it!

B resends, can't tell message is just slow

delayed acknowledgements

machine A — "The meeting is at 12pm." → machine B

"timeout"

Got it!

18

delayed acknowledgements

machine A

"The meeting is at 12pm."

"timeout"

Got it!

machine B

18

delayed acknowledgements

"The meeting is at 12pm."

"timeout"

"Got it!"

"The meeting is at 12pm."

machine
A

machine
B

Got it!

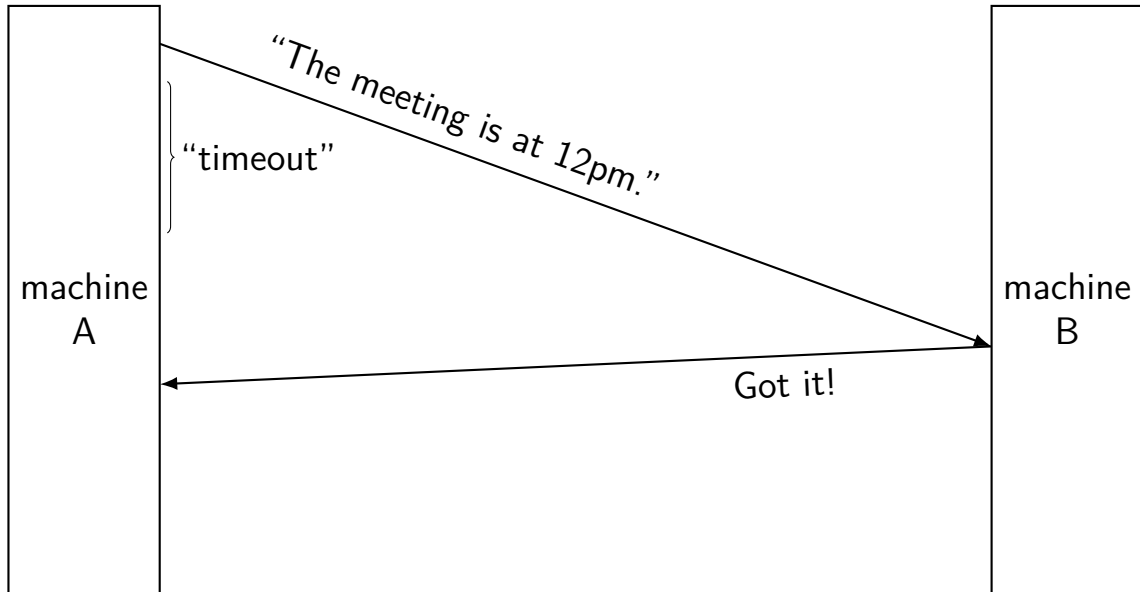B can't tell that first acknowledgment wasn't lost
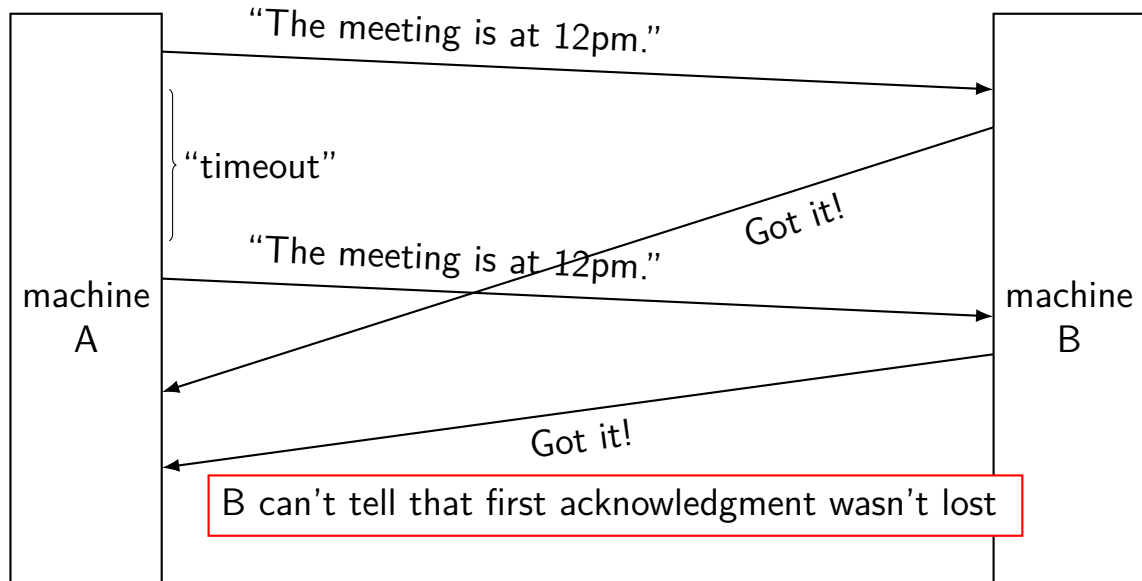
network limitations/failures

messages lost

messages delayed/reordered

messages limited in size

messages corrupted

splitting messages: try 1

machine A — "The meeting" → machine B

machine A ← "got it" — machine B

machine A ← " is at 12pm." — machine B

machine A ← "got it" — machine B
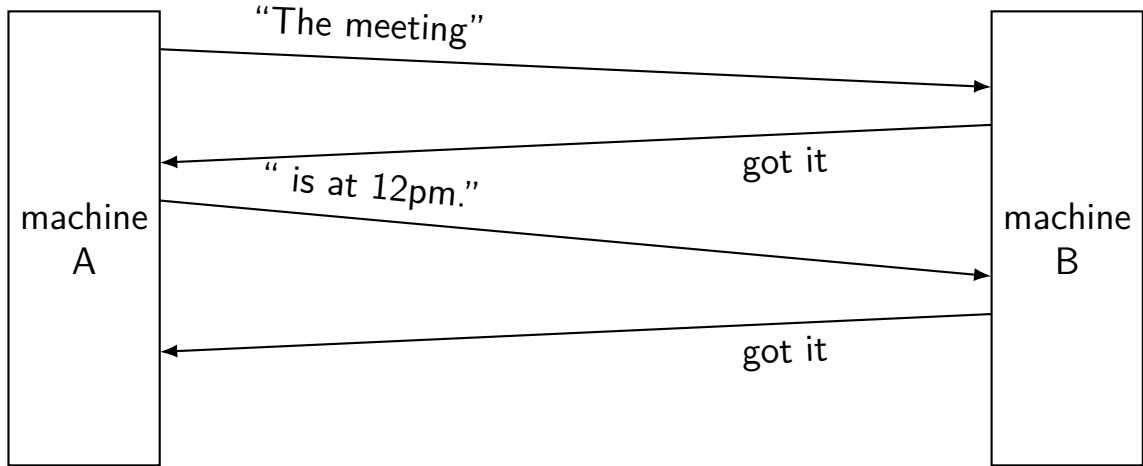
reconstructed message:
The meeting is at 12pm.

splitting messages: try 1 — problem 1

splitting messages: try 1 — problem 1

"The meeting"

"The meeting"

got it

"is at 12pm."

got it

got it
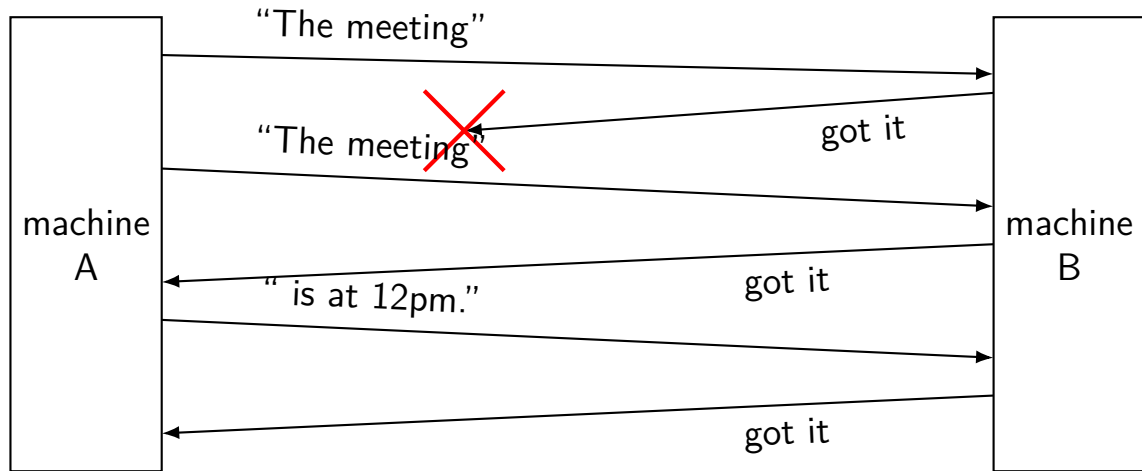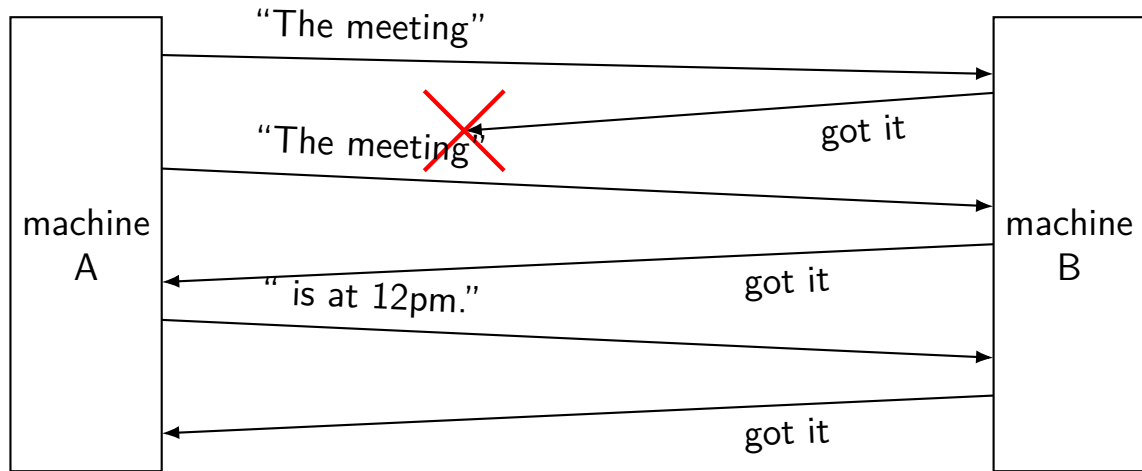
machine A

machine B

reconstructed message:
The meetingThe meeting is at 12pm.

exercise: other problems?

other scenarios where we'd also have problems?

1. message (instead of acknowledgment) is lost
2. first message from machine A is delayed a long time by network
3. acknowledgment of second message lost instead of first

splitting messages: try 2

machine A — part 1: "The meeting" → machine B

machine B — got it → machine A

machine A — part 2: " is at 12pm." → machine B

machine B — got it → machine A
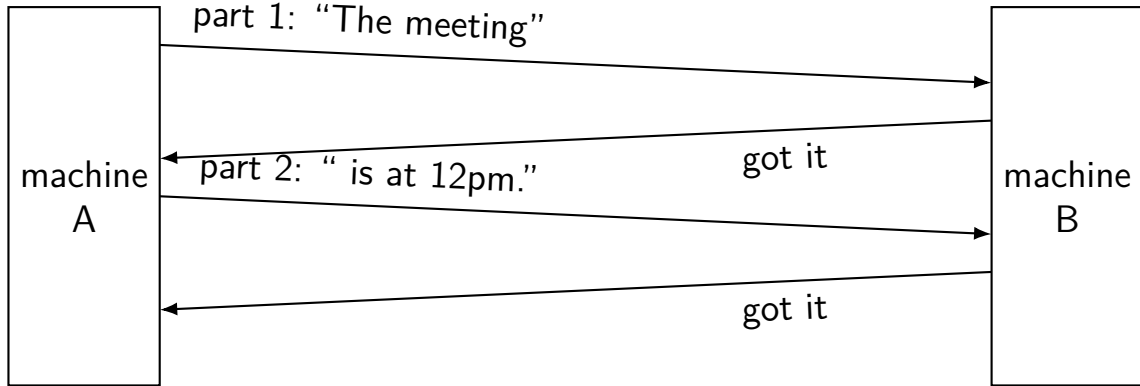
reconstructed message:
The meeting is at 12pm.

splitting messages: try 2 — missed ack

part 1: "The meeting"

got it

part 1: "The meeting"

machine A

machine B

part 2: " is at 12pm."

got it

got it
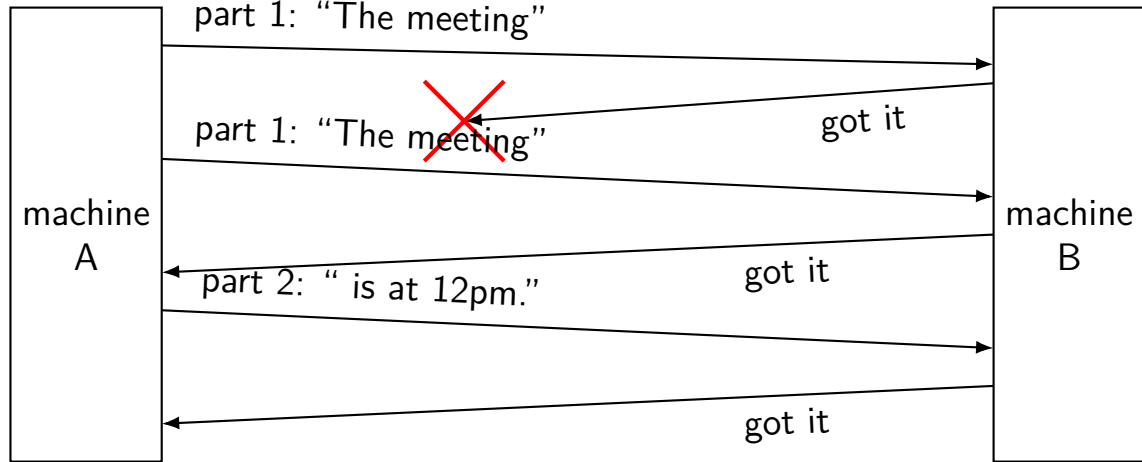
reconstructed message:
The meeting is at 12pm.

splitting messages: try 2 — problem
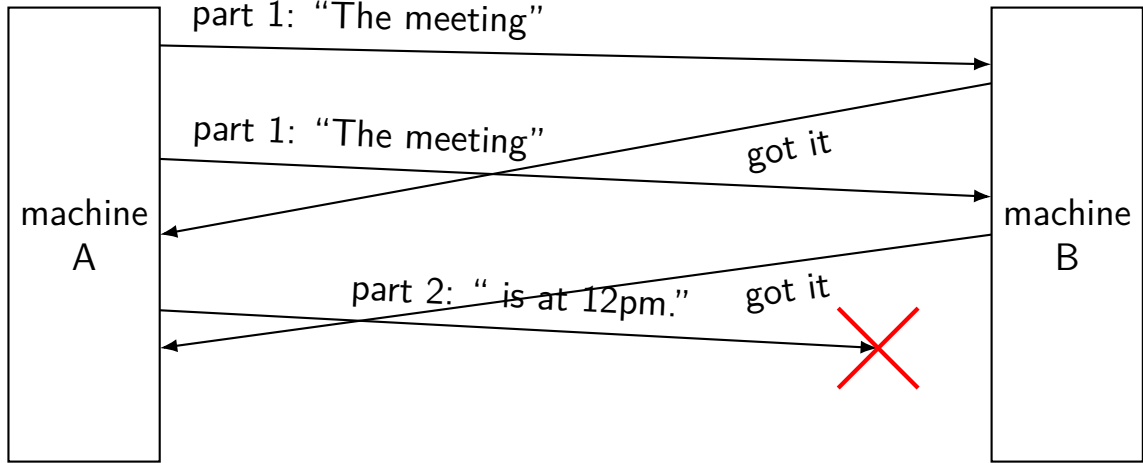
A thinks: part 1 + part 2 acknowleged!

splitting messages: version 3
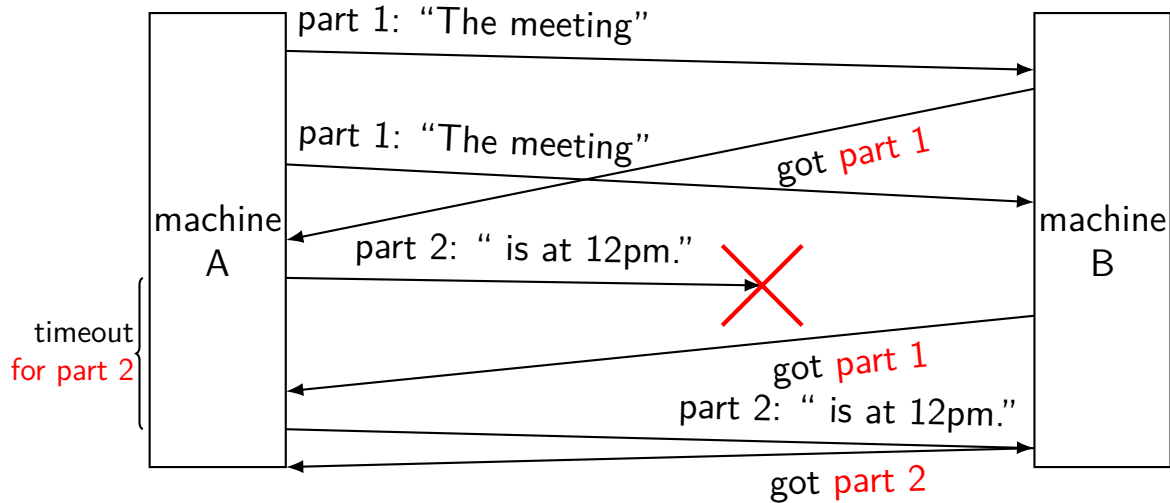
network limitations/failures

messages lost

messages delayed/reordered

messages limited in size

messages corrupted

message corrupted

instead of sending "message"

say Hash("message") = 0xABCDEF12

then send "0xABCDEF12,message"

when receiving, recompute hash

pretend message lost if does not match

"checksum"

these hashes commonly called "checksums"

in UDP/TCP, hash function: treat bytes of messages as array of integers; then add integers together

going faster

so far: send one message, get acknowledgments

pretty slow

instead, can send a bunch of parts and get them acknowledged together

need to do *congestion control* to avoid overloading network

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

more than four layers?

sometimes more layers above 'application'

e.g. HTTPS:
HTTP (app layer) on TLS (another app layer) on TCP (network) on …

e.g. DNS over HTTPS:
DNS (app layer) on HTTP on on TLS on TCP on …

e.g. SFTP:
SFTP (app layer??) on SSH (another app layer) on TCP on …

e.g. HTTP over OpenVPN:
HTTP on TCP on IP on OpenVPN on UDP on different IP on …

names and addresses

name	address
logical identifier	location/how to locate
variable `counter`	memory address `0x7FFF9430`
DNS name `www.virginia.edu`	IPv4 address `128.143.22.36`
DNS name `mail.google.com`	IPv4 address `216.58.217.69`
DNS name `mail.google.com`	IPv6 address `2607:f8b0:4004:80b::2005`
DNS name `reiss-t3620.cs.virginia.edu`	IPv4 address `128.143.67.91`
DNS name `reiss-t3620.cs.virginia.edu`	MAC address `18:66:da:2e:7f:da`
service name `https`	port number 443
service name `ssh`	port number 22

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

an Ethernet frame

	destination MAC address			source MAC address			frame type

4c cc 6a ba 1c b9 d8 07 b6 d9 ae 50 08 00

frame's data

45 00 00 60 db 89 40 00 f2 06 cf cd 34 60 e6 a2

c0 a8 01 95 01 bb aa c4 40 2b d6 46 7c 9d 15 e4

80 18 40 02 65 fe 00 00 01 01 08 0a 03 83 98 62

19 70 27 9e 17 03 03 00 27 00 00 00 00 00 00 00
c8 b9 ab 81 50 e0 ef 1a d8 97 73 76 9a ee 33 d4

an Ethernet frame

destination MAC address **source MAC address** **frame type**

4c cc 6a ba 1c b9 d8 07 b6 d9 ae 50 08 00

vers. **length** **protocol** **source IPv4 address**

45 00 00 60 db 89 40 00 f2 06 cf cd 34 60 e6 a2

destination IPv4 address **packet's data**

c0 a8 01 95 01 bb aa c4 40 2b d6 46 7c 9d 15 e4

80 18 40 02 65 fe 00 00 01 01 08 0a 03 83 98 62

19 70 27 9e 17 03 03 00 27 00 00 00 00 00 00 00
c8 b9 ab 81 50 e0 ef 1a d8 97 73 76 9a ee 33 d4

IP packet

35

an Ethernet frame

destination MAC address | **source MAC address** | **frame type**

4c cc 6a ba 1c b9 d8 07 b6 d9 ae 50 08 00

vers. | **length** | **protocol** | **source IPv4 address**

45 00 00 60 db 89 40 00 f2 06 cf cd 34 60 e6 a2

destination IPv4 address | **source port** | **dest. port** | **sequence num.**

c0 a8 01 95 01 bb aa c4 40 2b d6 46 7c 9d 15 e4

flags

80 18 40 02 65 fe 00 00 01 01 08 0a 03 83 98 62

segment's data

19 70 27 9e 17 03 03 00 27 00 00 00 00 00 00 00
c8 b9 ab 81 50 e0 ef 1a d8 97 73 76 9a ee 33 d4

IP packet

TCP segment

35

the link layer

Ethernet, Wi-Fi, Bluetooth, DOCSIS (cable modems), …

allows send/recv messages to machines on "same" network segment
> typically: wireless range+channel or connected to a single switch/router
> could be larger (if *bridging* multiple network segments)
> could be smaller (switch/router uses "virtual LANs")

typically: source+destination specified with MAC addresses
> MAC = media access control
> usually manufacturer assigned / hard-coded into device
> unique address per port/wifi transmitter/etc.

can specify destination of "anyone" (called *broadcast*)

link layer quality of service

if frame gets…

event	on Ethernet	on WiFi
collides with another	detected + may resend	resend
not received	lose silently	resent
header corrupted	usually discard silently	usually resend
data corrupted	usually discard silently	usually resend
too long	not allowed to send	not allowed to send
reordered (v. other messages)	received out of order	received out of order
destination unknown	lose silently	usually resend??
too much being sent	discard excess?	discard excess?

link layer reliablity?

Ethernet + Wifi have checksums

Q1: Why doesn't this give us uncorrupted messages?
   Why do we still have checksums at the higher layers?

Q2: What's a benefit of doing this if we're also doing it in the higher layer?

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

the network layer

the Internet Protocool (IP) version 4 or version 6
    there are also others, but quite uncommon today

allows send messages to/recv messages from other networks
    "internetwork"

messages usually called "packets"

network layer quality of service

if packet ...

event	on IPv4/v6
collides with another	out of scope — handled by link layer
not received	lost silently
header corrupted	usually discarded silently
data corrupted	received corrupted
too long	dropped with notice or "fragmented" + recombined
reordered (v. other messages)	received out of order
destination unknown	usually dropped with notice
too much being sent	discard excess

network layer quality of service

if packet …

event	on IPv4/v6
collides with another	out of scope — handled by link layer
not received	lost silently
header corrupted	usually discarded silently
data corrupted	received corrupted
too long	dropped with notice or "fragmented" + recombined
reordered (v. other messages)	received out of order
destination unknown	usually dropped with notice
too much being sent	discard excess

includes dropped by link layer
(e.g. if detected corrupted there)

IPv4 addresses

32-bit numbers

typically written like 128.143.67.11
> four 8-bit decimal values separated by dots
> first part is most significant
> same as $128 \cdot 256^3 + 143 \cdot 256^2 + 67 \cdot 256 + 11 = 2\,156\,782\,459$

organizations get blocks of IPs
> e.g. UVa has 128.143.0.0–128.143.255.255
> e.g. Google has 216.58.192.0–216.58.223.255 and
> 74.125.0.0–74.125.255.255 and 35.192.0.0–35.207.255.255

some IPs reserved for non-Internet use (127.*, 10.*, 192.168.*)

IPv6 addresses

IPv6 like IPv4, but with 128-bit numbers

written in hex, 16-bit parts, seperated by colons (:)

strings of 0s represented by double-colons (::)

typically given to users in blocks of 2^{80} or 2^{64} addresses
    no need for address translation?

2607:f8b0:400d:c00::6a =
2607:f8b0:400d:0c00:0000:0000:0000:006a
    $2607f8b0400d0c0000000000000006a_{\text{SIXTEEN}}$

selected special IPv6 addresses

`::1` = localhost

anything starting with `fe80` = link-local addresses
    never forwarded by routers

IPv4 addresses and routing tables
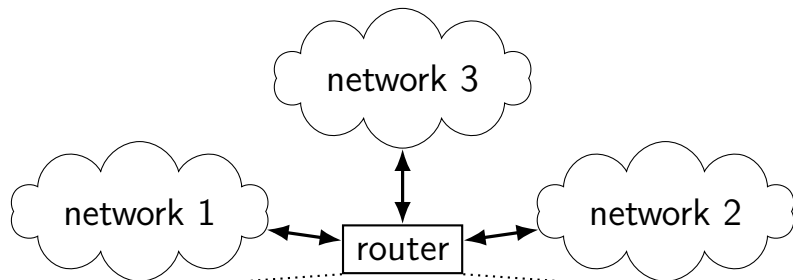
if I receive data for…	send it to…
128.143.0.0—128.143.255.255	network 1
192.107.102.0–192.107.102.255	network 1
…	…
4.0.0.0–7.255.255.255	network 2
64.8.0.0–64.15.255.255	network 2
…	…
anything else	network 3

selected special IPv4 addresses

127.0.0.0 — 127.255.255.255 — localhost
     AKA loopback
     the machine we're on
     typically only 127.0.0.1 is used

192.168.0.0–192.168.255.255 and
10.0.0.0–10.255.255.255 and
172.16.0.0–172.31.255.255
     "private" IP addresses
     not used on the Internet
     commonly connected to Internet with network address translation
     also 100.64.0.0–100.127.255.255 (but with restrictions)

169.254.0.0-169.254.255.255
     link-local addresses — 'never' forwarded by routers

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

port numbers

we run multiple programs on a machine
  IP addresses identifying machine — not enough

port numbers

we run multiple programs on a machine
>   IP addresses identifying machine — not enough

so, add 16-bit *port numbers*
>   think: multiple PO boxes at address

port numbers

we run multiple programs on a machine
>   IP addresses identifying machine — not enough

so, add 16-bit *port numbers*
>   think: multiple PO boxes at address

0–49151: typically assigned for particular services
>   80 = http, 443 = https, 22 = ssh, …

49152–65535: allocated on demand
>   default "return address" for client connecting to server

UDP v TCP

TCP: stream to other program
    reliable transmission of as much data as you want
    "connecting" fails if server not responding
    write(fd, "a", 1); write(fd, "b", 1) = write(fd, "ab", 2)
    (at least) one socket per remote program being talked to

UDP: messages sent to program, but no reliablity/streams
    unreliable transmission of short messages
    write(fd, "a", 1); write(fd, "b", 1) $\neq$ write(fd, "ab", 2)
    "connecting" just sets default destination
    can sendto()/recvfrom() multiple other programs with one socket
        (but don't have to)

'connected' UDP sockets

```
int fd = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in my_addr= ...;
/* set local IP address + port */
bind(fd, &my_addr, sizeof(my_addr))
struct sockaddr_in to_addr = ...;
connect(fd, &to_addr); /* set remote IP address + port */
    /* doesn't actually communicate with remote address yet */
...
int count = write(fd, data, data_size);
// OR
int count = send(fd, data, data_size, 0 /* flags */);
    /* single message -- sent ALL AT ONCE */

int count = read(fd, buffer, buffer_size);
// OR
int count = recv(fd, buffer, buffer_size, 0 /* flags */);
    /* receives whole single message ALL AT ONCE */
```

UDP sockets on IPv4

```c
int fd = socket(AF_INET, SOCK_DGRAM, 0);
struct sockaddr_in my_addr= ...;
/* set local IP address + port */
if (0 != bind(fd, &my_addr, sizeof(my_addr)))
    handle_error();
...
struct sockaddr_in to_addr = ...;
    /* send a message to specific address */
int bytes_sent = sendto(fd, data, data_size, 0 /* flags */,
    &to_addr, sizeof(to_addr));

struct sockaddr_in from_addr = ...;
    /* receive a message + learn where it came from */
int bytes_recvd = recvfrom(fd, &buffer[0], buffer_size, 0,
    &from_addr, sizeof(from_addr));
...
```

connections in TCP/IP

connection identified by *5-tuple*
    used by OS to lookup "where is the socket?"

(protocol=TCP/UDP, local IP addr., local port, remote IP addr., remote port)

local IP address, port number can be set with `bind()` function
    typically always done for servers, not done for clients
    system will choose default if you don't

connections on my desktop

```
cr4bd@reiss-t3620>/u/cr4bd
$ netstat --inet --inet6 --numeric
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 128.143.67.91:49202     128.143.63.34:22        ESTABLISHI
tcp        0      0 128.143.67.91:803       128.143.67.236:2049     ESTABLISHI
tcp        0      0 128.143.67.91:50292     128.143.67.226:22       TIME_WAIT
tcp        0      0 128.143.67.91:54722     128.143.67.236:2049     TIME_WAIT
tcp        0      0 128.143.67.91:52002     128.143.67.236:111      TIME_WAIT
tcp        0      0 128.143.67.91:732       128.143.67.236:63439    TIME_WAIT
tcp        0      0 128.143.67.91:40664     128.143.67.236:2049     TIME_WAIT
tcp        0      0 128.143.67.91:54098     128.143.67.236:111      TIME_WAIT
tcp        0      0 128.143.67.91:49302     128.143.67.236:63439    TIME_WAIT
tcp        0      0 128.143.67.91:50236     128.143.67.236:111      TIME_WAIT
tcp        0      0 128.143.67.91:22        172.27.98.20:49566      ESTABLISHI
tcp        0      0 128.143.67.91:51000     128.143.67.236:111      TIME_WAIT
tcp        0      0 127.0.0.1:50438         127.0.0.1:631           ESTABLISHI
tcp        0      0 127.0.0.1:631           127.0.0.1:50438         ESTABLISHI
```

non-connection sockets

TCP servers waiting for connections +
UDP sockets with no particular remote host

Linux: OS keeps 5-tuple with "wildcard" remote address

"listening" sockets on my desktop

```
cr4bd@reiss-t3620>/u/cr4bd
$ netstat --inet --inet6 --numeric --listen
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:38537         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:36777         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:41099           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:45291           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:51949         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:41071         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:32881         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:38673         0.0.0.0:*               LISTEN
 ....
tcp6       0      0 :::42689                :::*                    LISTEN
udp        0      0 128.143.67.91:60001     0.0.0.0:*
udp        0      0 128.143.67.91:60002     0.0.0.0:*
```

TCP state machine

TIME_WAIT, ESTABLISHED, …?

OS tracks "state" of TCP connection
    am I just starting the connection?
    is other end ready to get data?
    am I trying to close the connection?
    do I need to resend something?
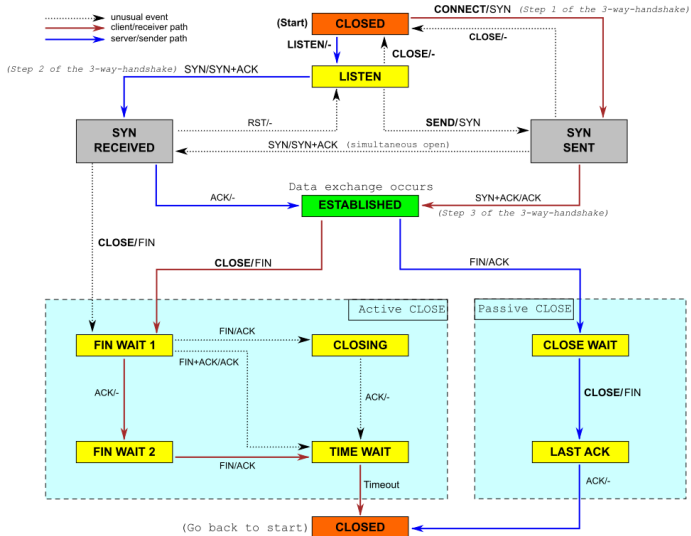
standardized set of state names

TIME_WAIT

remember delayed messages?

problem for TCP ports

if I reuse port number, I can get message from old connection

solution: TIME_WAIT to make sure connection really done
   done after sending last message in connection

TCP state machine picture

58

names and addresses

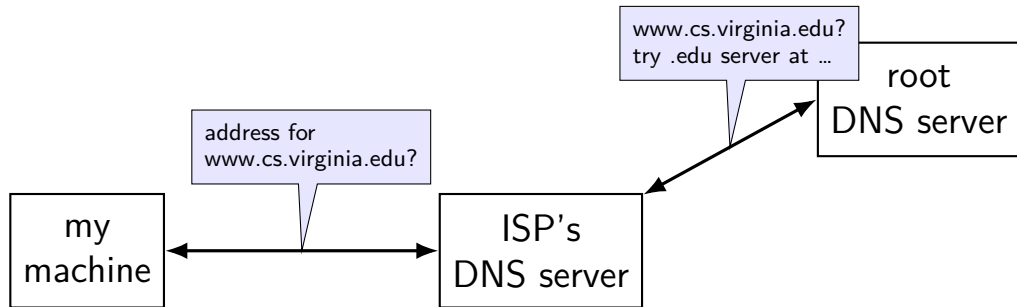| name	address
logical identifier	location/how to locate
variable counter	memory address 0x7FFF9430
DNS name www.virginia.edu	IPv4 address 128.143.22.36
DNS name mail.google.com	IPv4 address 216.58.217.69
DNS name mail.google.com	IPv6 address 2607:f8b0:4004:80b::2005
DNS name reiss-t3620.cs.virginia.edu	IPv4 address 128.143.67.91
DNS name reiss-t3620.cs.virginia.edu	MAC address 18:66:da:2e:7f:da
service name https	port number 443
service name ssh	port number 22

DNS: distributed database
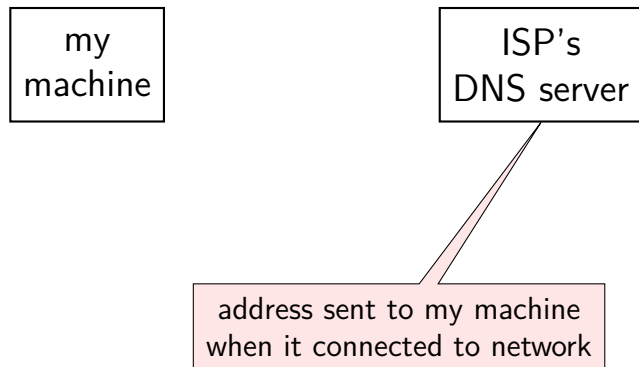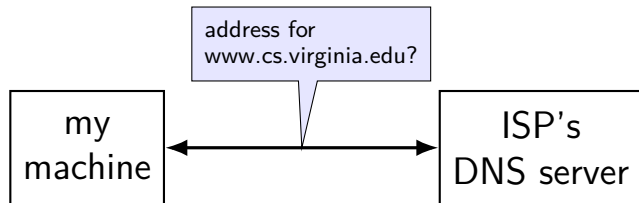
DNS: distributed database

DNS: distributed database

DNS: distributed database

DNS: distributed database

www.cs.virginia.edu?
try .edu server at …

root
DNS server

address for
www.cs.virginia.edu?

my
machine

ISP's
DNS server

.edu
DNS server

www.cs.virginia.edu =
128.143.67.11

.edu server doesn't change much
optimization: *cache* its address

check for updated version once in a while

DNS server
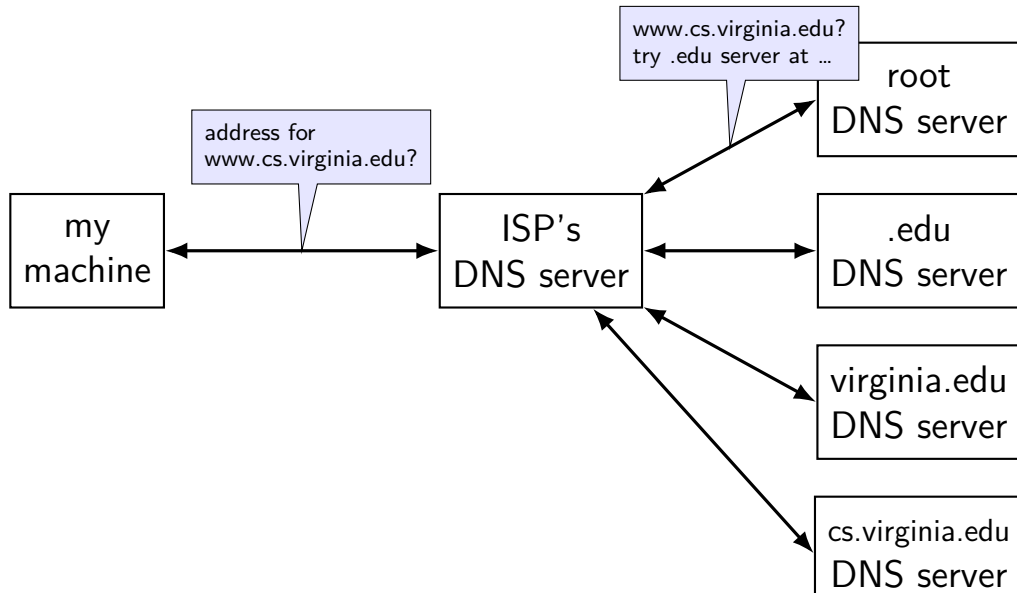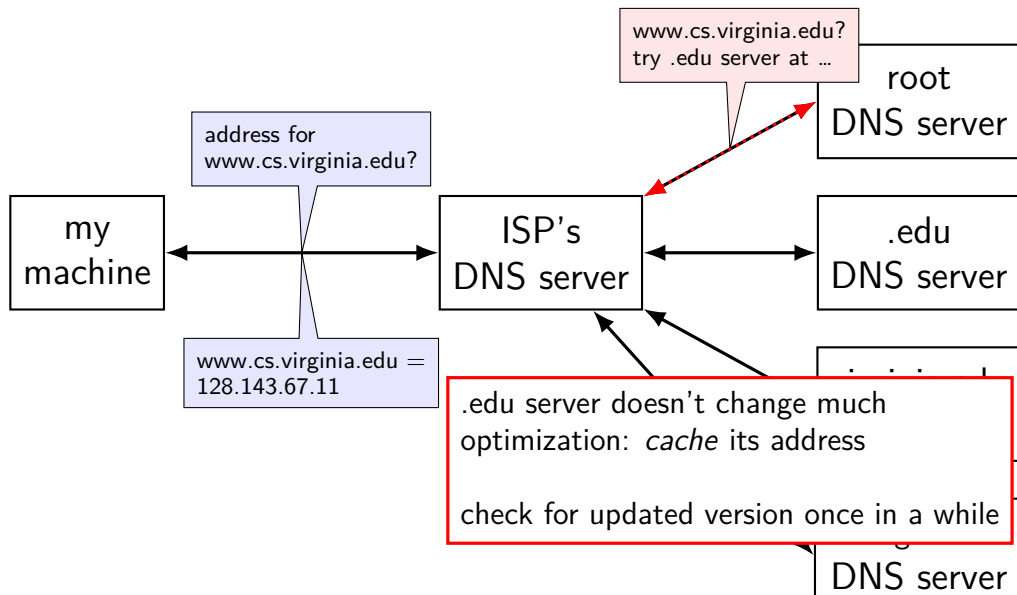
querying the root

```
$ dig +trace +all www.cs.virginia.edu
...
edu.                    172800      IN      NS      b.edu-servers.net.
edu.                    172800      IN      NS      f.edu-servers.net.
edu.                    172800      IN      NS      i.edu-servers.net.
edu.                    172800      IN      NS      a.edu-servers.net.
...
b.edu-servers.net.      172800      IN      A       191.33.14.30
b.edu-servers.net.      172800      IN      AAAA    2001:503:231d::2:30
f.edu-servers.net.      172800      IN      A       192.35.51.30
f.edu-servers.net.      172800      IN      AAAA    2001:503:d414::30
...
;; Received 843 bytes from 198.97.190.53#53(h.root-servers.net) in 8 ms
...
```

querying the edu

```
$ dig +trace +all www.cs.virginia.edu
...
virginia.edu.                   172800          IN      NS        nom.virginia.edu.
virginia.edu.                   172800          IN      NS        uvaarpa.virginia.edu.
virginia.edu.                   172800          IN      NS        eip-01-aws.net.virginia.edu.
nom.virginia.edu.        172800          IN         A        128.143.107.101
uvaarpa.virginia.edu.        172800          IN         A        128.143.107.117
eip-01-aws.net.virginia.edu. 172800 IN          A        44.234.207.10
;; Received 165 bytes from 192.26.92.30#53(c.edu-servers.net) in 40 ms
...
```

querying virginia.edu+cs.virginia.edu

```
$ dig +trace +all www.cs.virginia.edu
...
cs.virginia.edu.          3600          IN          NS          coresrv01.cs.virginia.edu.
coresrv01.cs.virginia.edu. 3600         IN          A           128.143.67.11
;; Received 116 bytes from 44.234.207.10#53(eip-01-aws.net.virginia.edu) in 72 ms

www.cs.Virginia.EDU.          172800          IN          A          128.143.67.11
cs.Virginia.EDU.          172800          IN          NS          coresrv01.cs.Virginia.EDU.
coresrv01.cs.Virginia.EDU. 172800 IN          A          128.143.67.11
;; Received 151 bytes from 128.143.67.11#53(coresrv01.cs.virginia.edu) in 4 ms
```

querying typical ISP's resolver

```
$ dig www.cs.virginia.edu
...
;; ANSWER SECTION:
www.cs.Virginia.EDU.        7183      IN        A        128.143.67.11
..
```

cached response

valid for 7183 more seconds

after that everyone needs to check again

names and addresses

name	address
logical identifier	location/how to locate
variable `counter`	memory address `0x7FFF9430`
DNS name `www.virginia.edu`	IPv4 address `128.143.22.36`
DNS name `mail.google.com`	IPv4 address `216.58.217.69`
DNS name `mail.google.com`	IPv6 address `2607:f8b0:4004:80b::2005`
DNS name `reiss-t3620.cs.virginia.edu`	IPv4 address `128.143.67.91`
DNS name `reiss-t3620.cs.virginia.edu`	MAC address `18:66:da:2e:7f:da`
service name `https`	port number 443
service name `ssh`	port number 22

two types of addresses?

MAC addreses: on link layer

IP addresses: on network layer

how do we know which MAC address to use?

a table on my desktop

my desktop:

```
$ arp -an
? (128.143.67.140) at 3c:e1:a1:18:bd:5f [ether] on enp0s31f6
? (128.143.67.236) at <incomplete> on enp0s31f6
? (128.143.67.11) at 30:e1:71:5f:39:10 [ether] on enp0s31f6
? (128.143.67.92) at <incomplete> on enp0s31f6
? (128.143.67.5) at d4:be:d9:b0:99:d1 [ether] on enp0s31f6
```

…

network address to link-layer address + interface

only tracks things directly connected to my local network

how is that table made?

ask all machines on local network (same switch)

"Who has 128.148.67.140"

the correct one replies

what about non-local machines?

when configuring network specify:

range of addresses to expect on local network
    128.148.67.0-128.148.67.255 on my desktop
    "netmask"

gateway machine to send to for things outside my local network
    128.143.67.1 on my desktop
    my desktop looks up the corresponding MAC address

routes on my desktop

```
$ /sbin/route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         128.143.67.1    0.0.0.0         UG    100    0        0 enp0s31f6
128.143.67.0    0.0.0.0         255.255.255.0   U     100    0        0 enp0s31f6
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 enp0s31f6
```

network configuration says:

(line 2) to get to 128.143.67.0–128.143.67.255, send directly on local network

    "genmask" is mask (for bitwise operations) to specify how big range is

(line 3) to get to 169.254.0.0–169.254.255.255, send directly on local network

(line 1) to get anywhere else, use "gateway" 128.143.67.1

URL / URIs

Uniform Resource Locators (URL)
>   tells how to find "resource" on network

Unifrom Resources Identifiers
>   superset of URLs

URI examples

```
https://kytos02.cs.virginia.edu:443/cs3130-spring2023/
                quizzes/quiz.php?qid=02#q2

https://kytos02.cs.virginia.edu/cs3130-spring2023/
                quizzes/quiz.php?qid=02

https://www.cs.virginia.edu/

sftp://cr4bd@portal.cs.virginia.edu/u/cr4bd/file.txt

tel:+1-434-982-2200

//www.cs.virginia.edu/~cr4bd/3130/S2023/
/~cr4bd/3130/S2023
```

scheme and/or host implied from context

URI generally

```
scheme://authority/path?query#fragment
```

scheme: — what protocol

//authority/
    authoirty = user@host:port OR host:port OR user@host OR host

path
    which resource

?query — usually key/value pairs

#fragment — place in resource

most components (sometimes) optional

URLs and HTTP (1)

```
http://www.foo.com:80/foo/bar?quux#q1
```

lookup IP address of www.foo.com

connect via TCP to port 80:
```
GET /foo/bar?quux HTTP/1.1
Host: www.foo.com:80
```

URLs and HTTP (1)

```
http://www.foo.com:80/foo/bar?quux#q1
```

lookup IP address of www.foo.com

connect via TCP to port 80:
```
GET /foo/bar?quux HTTP/1.1
Host: www.foo.com:80
```

URLs and HTTP (1)

```
http://www.foo.com:80/foo/bar?quux#q1
```

lookup IP address of www.foo.com

connect via TCP to port 80:
```
GET /foo/bar?quux HTTP/1.1
Host: www.foo.com:80
```

exercise: why include the Host there?

autoconfiguration

problem: how does my machine get IP address

otherwise:
    have sysadmin type one in?
    just choose one?
    ask machine on local network to assign it

autoconfiguration

problem: how does my machine get IP address

otherwise:
      have sysadmin type one in?
      just choose one?
      ask machine on local network to assign it

autoconfiguration

problem: how does my machine get IP address

otherwise:
>   have sysadmin type one in?
>   just choose one?
>   ask machine on local network to assign it

often local router machine runs service to assign IP addresses
>   knows what IP addresses are available
>   sysadmin might configure in mapping from MAC addresses to IP
>   addresses

DHCP high-level

protocol done over UDP

but since we don't have IP address yet, use `0.0.0.0`

and since we don't know server address, use `255.255.255.255`
    = "everyone on the local network"

local server replies to request with address + time limit

later: can send messages to local server to renew/give up address

DHCP high-level

protocol done over UDP

but since we don't have IP address yet, use `0.0.0.0`

and since we don't know server address, use `255.255.255.255`
     = "everyone on the local network"

local server replies to request with address + time limit

later: can send messages to local server to renew/give up address

exercise: why time limit?

DHCP "lease"

rather than getting address forever

but DHCP has way of releasing taken address

why impose a time limit

firewalls

don't want to expose network service to everyone?

solutions:
    service picky about who it accepts connections from
    filters in OS on machine with services
    filters on router

later two called "firewalls"

firewall rules examples?

ALLOW tcp port 443 (https) FROM everyone

ALLOW tcp port 22 (ssh) FROM my desktop's IP address

BLOCK tcp port 22 (ssh) FROM everyone else

ALLOW from address X to address Y

…

spoofing

if I only allow connections from my desktop's IP addresses, how would you attack this?

hint: how do we know what address messages come from?