UDP
Transport Layer Introduction

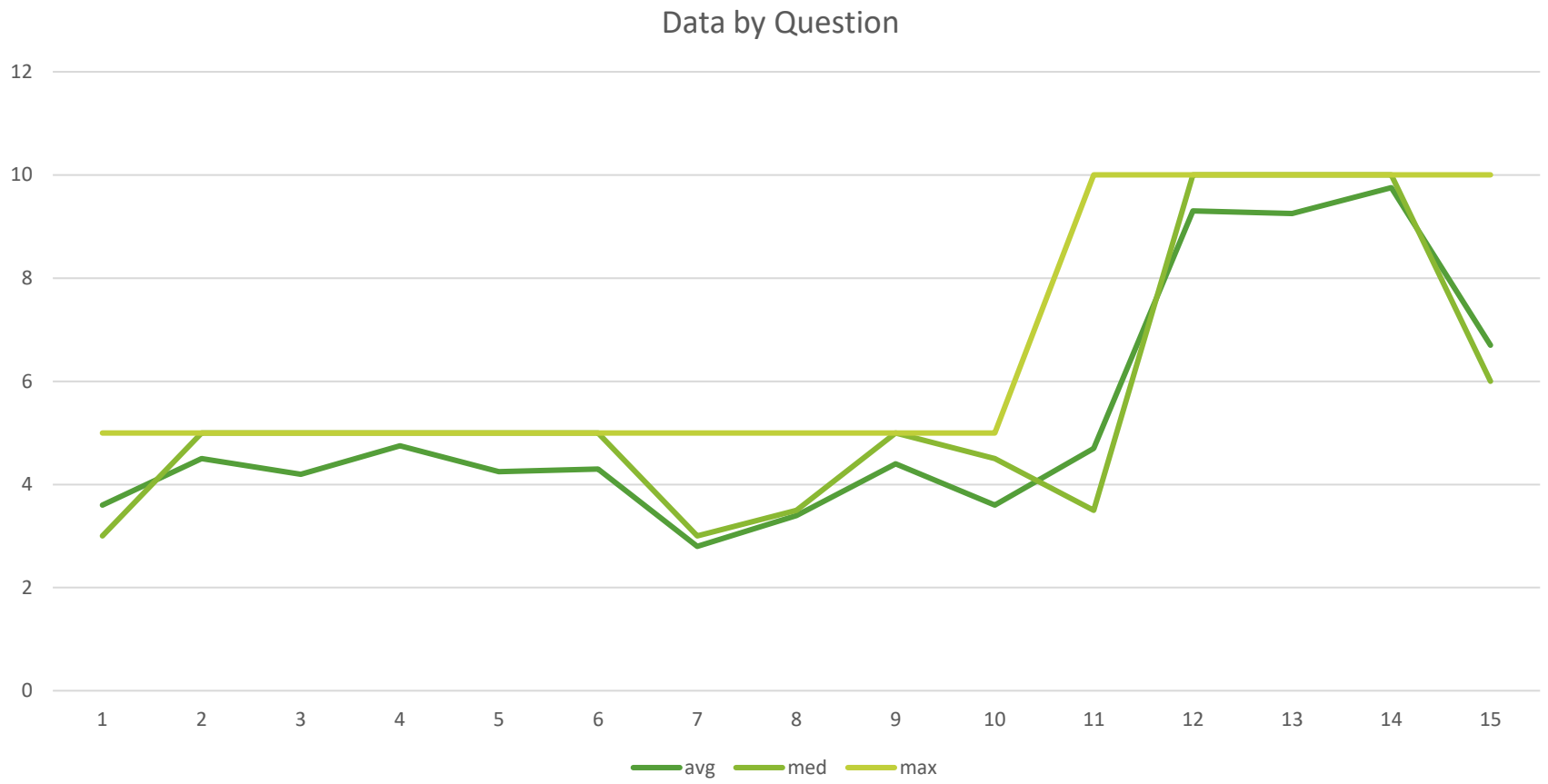
Kevin Scrivnor

COMP 429

Fall 2022

midterm review

	layers	error corr/detect	hex	bit stuffing	NAT	routing alg	routing tbl	timers	TFTP	DNS	NAT	troute	http	DNS	tftp	total
avg	3.6	4.5	4.2	4.75	4.25	4.3	2.8	3.4	4.4	3.6	4.7	9.3	9.25	9.75	6.7	79.5
med	3	5	5	5	5	5	3	3.5	5	4.5	3.5	10	10	10	6	78.5

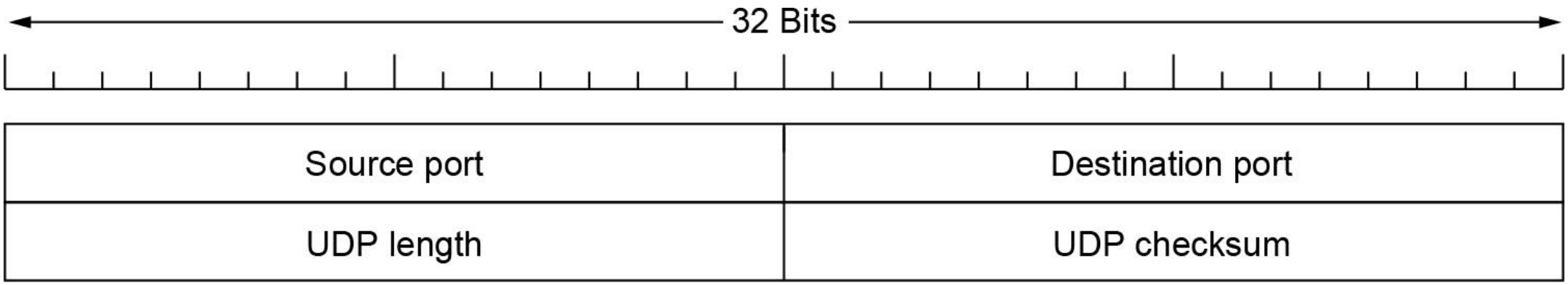


UDP

UDP

- Basically adds nothing on top of the network layer
- Connectionless, unreliable transport
- Basically adds source/dest ports and a checksum

- To quote the book, “it’s datagrams”



Transport Layer Introduction

Transport Layer Issues and Concepts

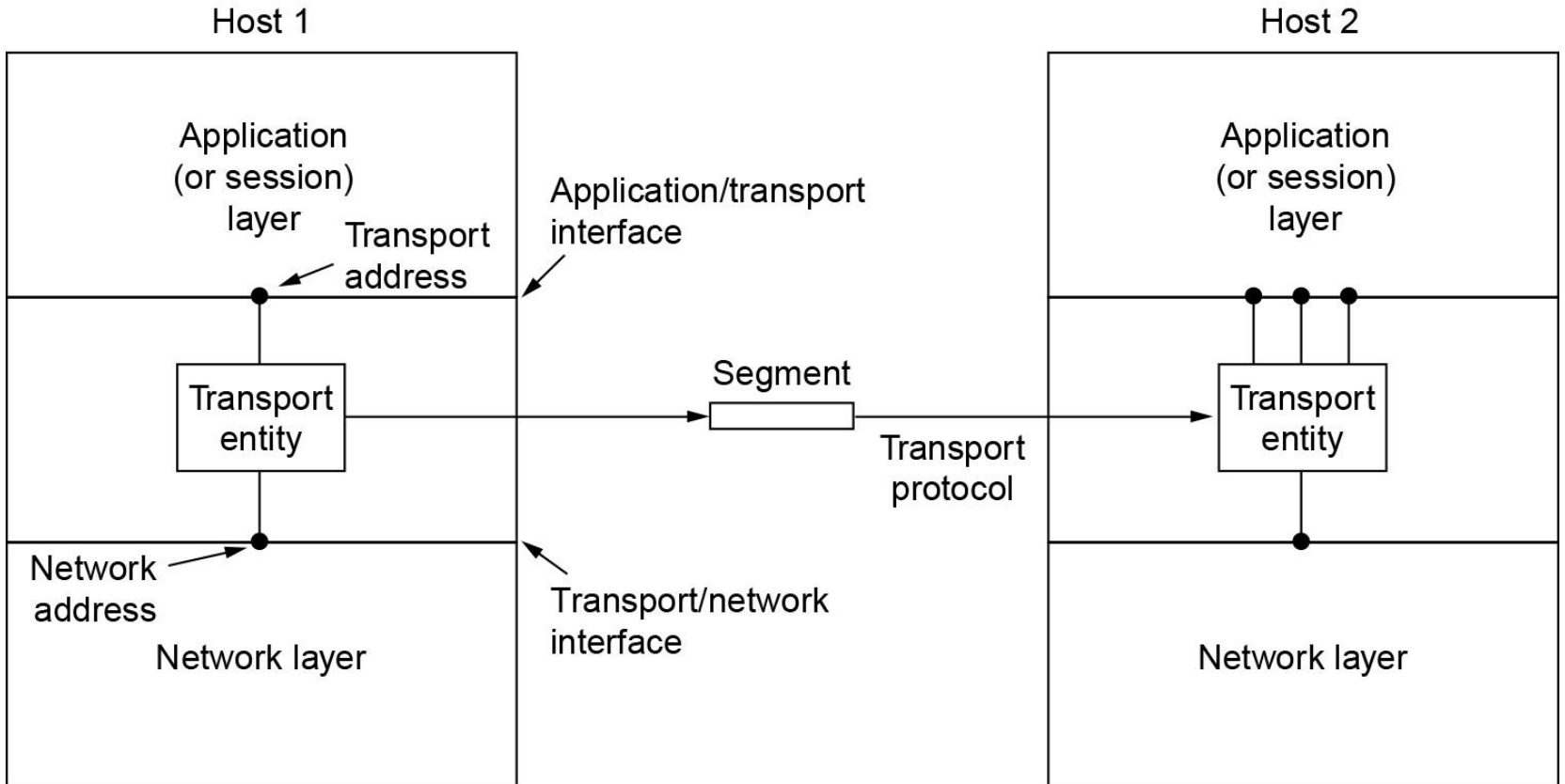
- This week we discuss concepts behind the transport layer
- The problems faced
- Design issues

- Transport layer “light”

- Next next week we dive into TCP and its various implementations
  - Which combines everything in different ways
  - Because real life is crazy

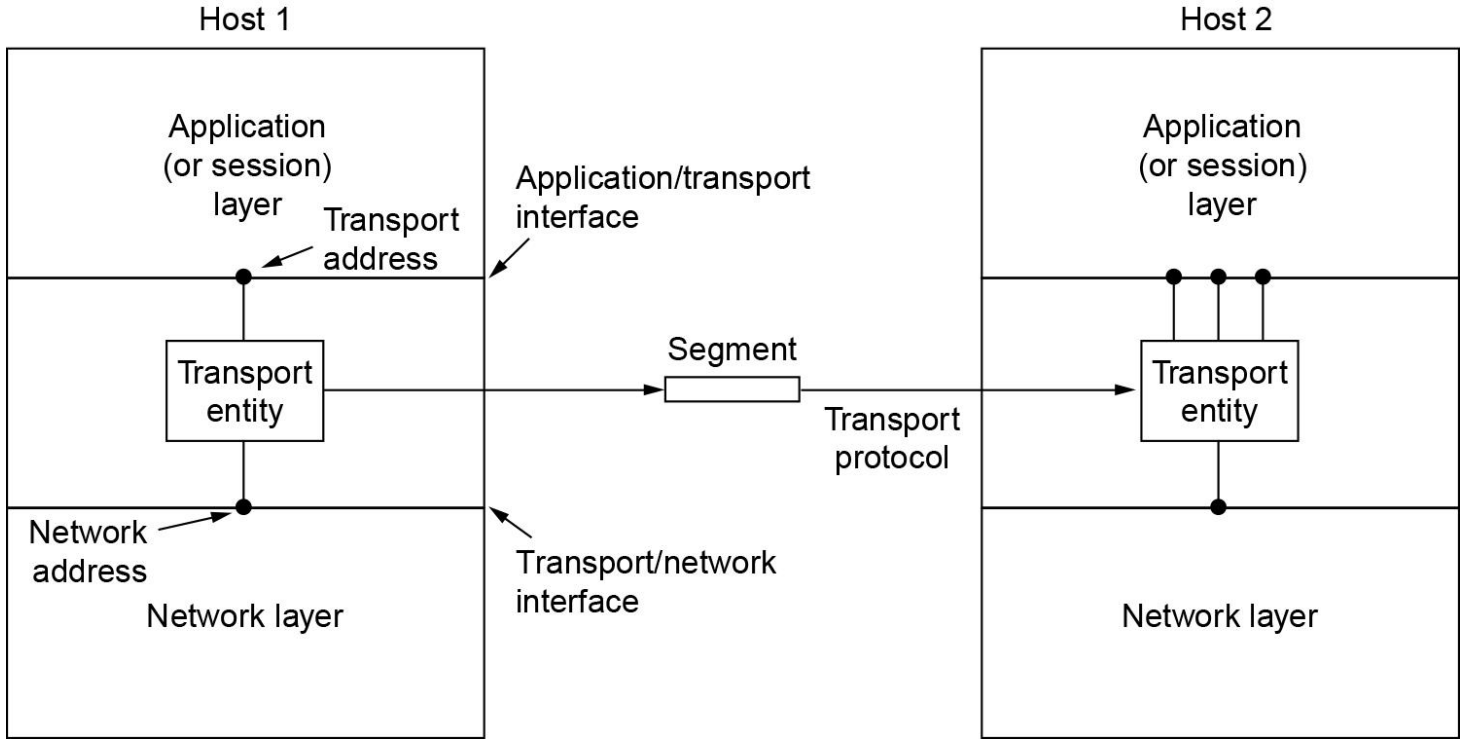
The ultimate goal of the transport layer is to

- provide efficient, reliable, cost-effective data transmission
- Another layer of abstraction, again think of the transport layer as *two processes communicating directly with each other*
- In this case, now we send *segments*



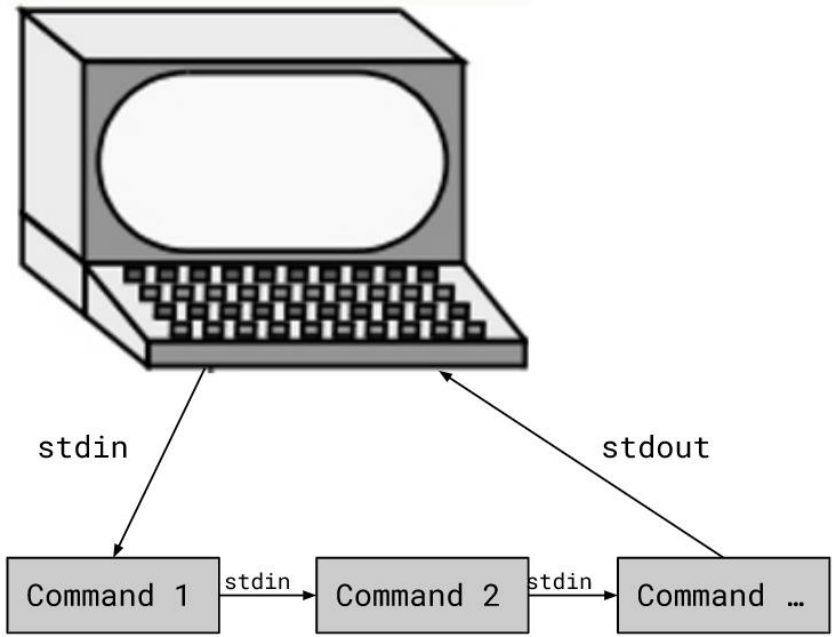
Transport Entity

- The *transport entity* is usually a process in the kernel
- The *application* sends data to the *transport entity* it wants to send across the network through API calls
- The application developer then does not have to worry too much about the network layer



Comparison to UNIX Pipes

- A pipe in UNIX is like the transport layer
- Pipes |
  - Take output from one process and communicate
  - the data to the input of another process
  - (IE, two processes talking to each other over a
  - perfect network (no data loss))

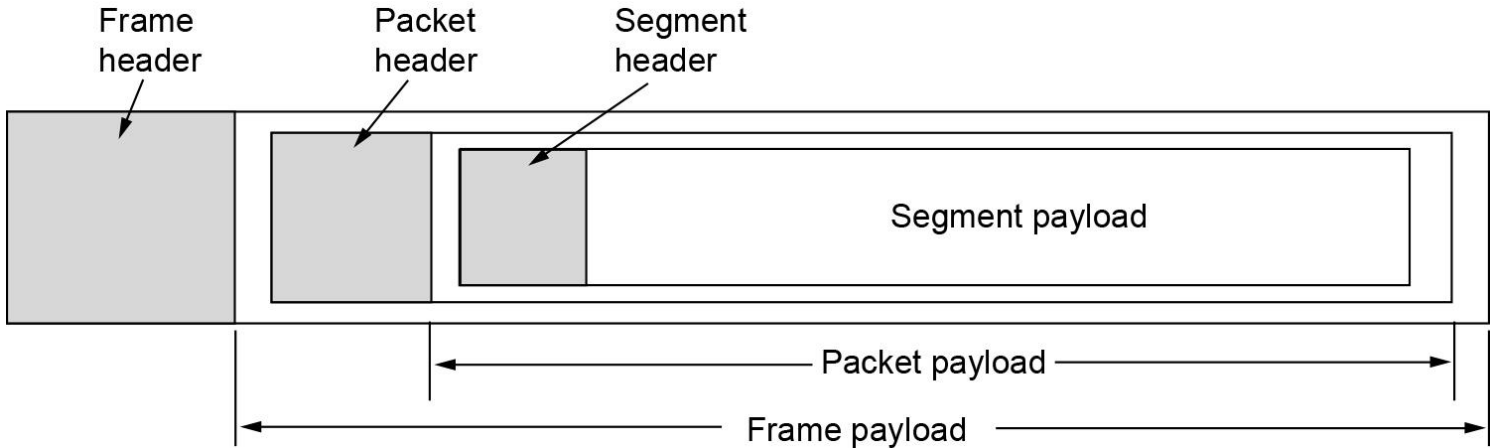


- Transport layer does the same thing
  - Take output from process on a host and communicate the data over a network to another process on another host
  - The key difference is the network is inherently unreliable

- *So the goal of the transport layer is to make it seem like the network layer is actually 100% reliable, when it is in fact not*

Bits, frames, packets, segments

- Bits on Wire
  - The physical bits represented by voltage through a wire or radio waves over WiFi
- Frames
  - The entire data "frame" sent by the data link layer encapsulating the network layer and above
- Packets
  - The network layer sends "packets" of information, encapsulating the transport layer and above
- Segments
  - The transport layer sends "segment" of data, which is encapsulating the data that the application layer wants to transmit over the network



There are two transport layer services

- in common practice and they are the same as the network layer
  - Connection oriented (TCP, QUIC)
  - Connectionless (UDP)

- We focus mostly on connection-oriented in this chapter

- Connection oriented has three phases
  - Connection establishment
  - Data transfer
  - Connection teardown

Why does the transport layer exist?

- if they both do very similar things? UDP doesn't add much on top of IP, and TCP is trying to reliably send data, isn't that what IP is doing?

- Generally, network layer is running only on routers
  - Largely concerned with providing adequate service
  - Monitors data loss
  - Routers can crash

- By separating the layers, transport layer can add/not add more features as needed to make the network layer *reliable as it needs to be*

Hypothetical Transport Layer

- A simple transport service would need to implement five *primitives*, or things that an API can do

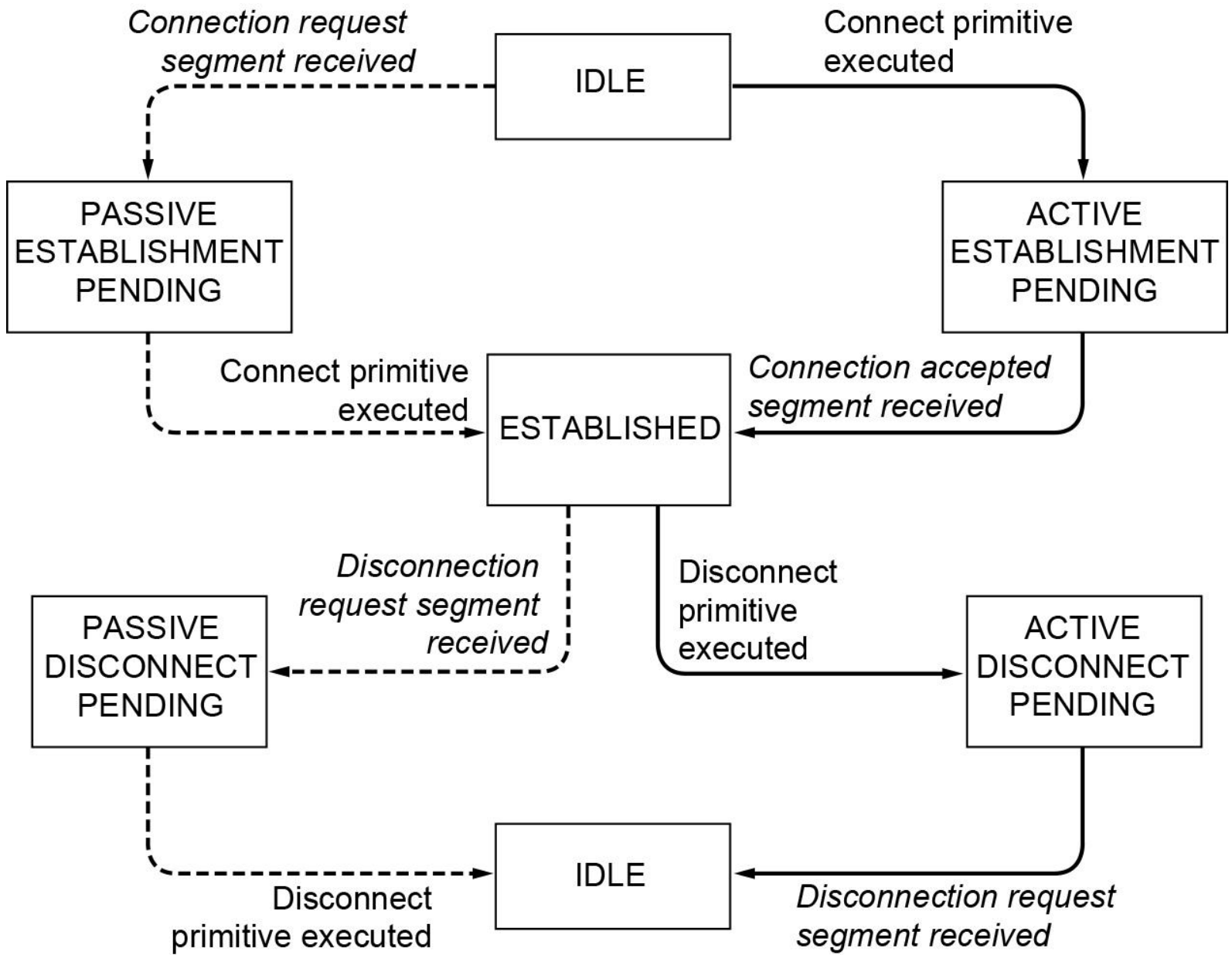
Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	Request a release of the connection

- *Block* simply means the CPU will wait until something happens, the program is essentially halted
- Just like when reading user input, th

State Diagram for a Transport Protocol

- Transitions labeled in *italics* are caused by packets arriving
- Solid lines show a client's state sequence
- Dotted lines show a server's state sequence

- *In almost all cases, a client connects to a server*



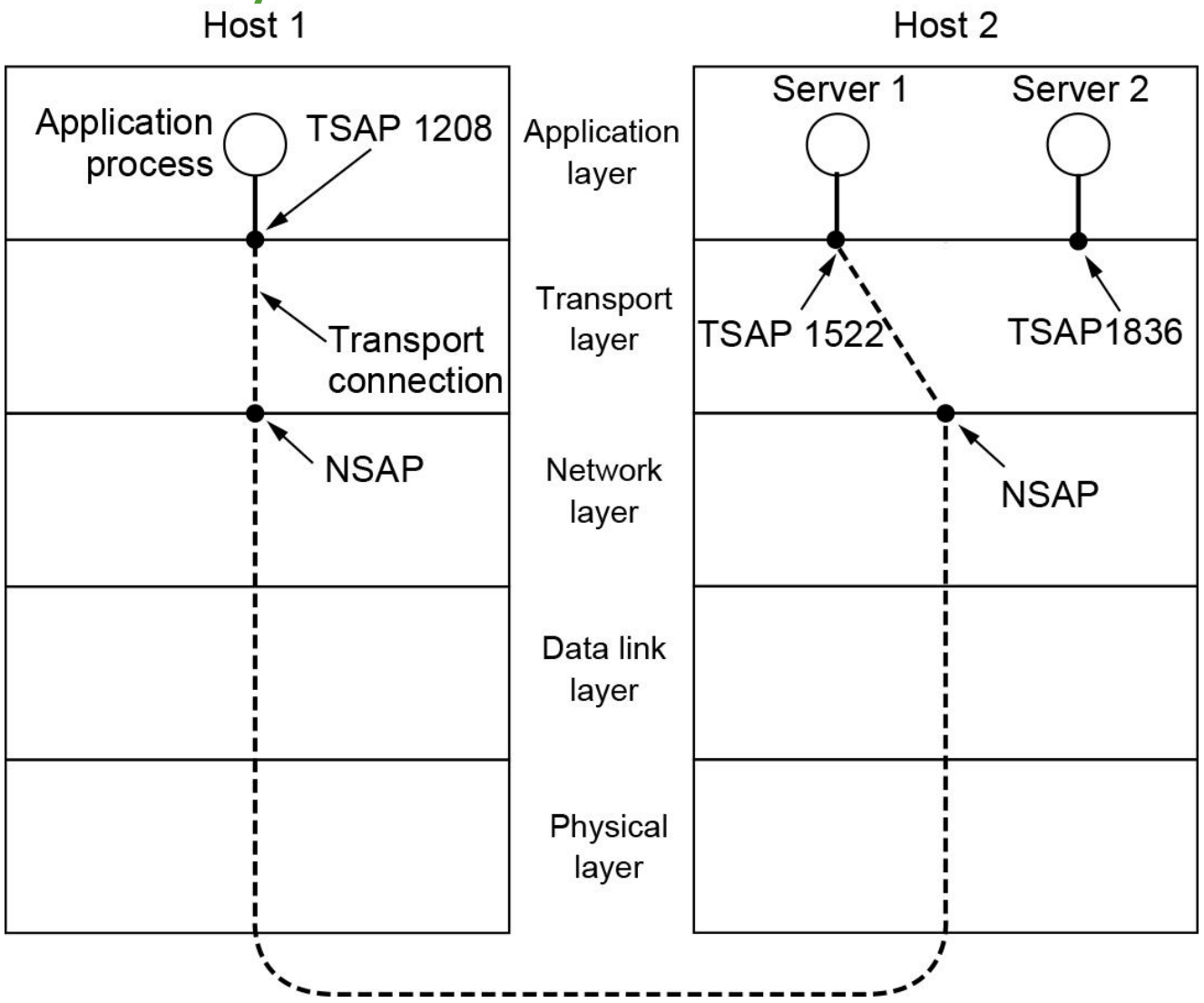
The network layer CANNOT

- Guarantee delivery
- Guarantee the order of packets arriving
- Guarantee that there will not be duplicates
- Guarantee latency

- The lack of these guarantees mean that the transport layer gets to deal with each one in some manner
  - ty network layer

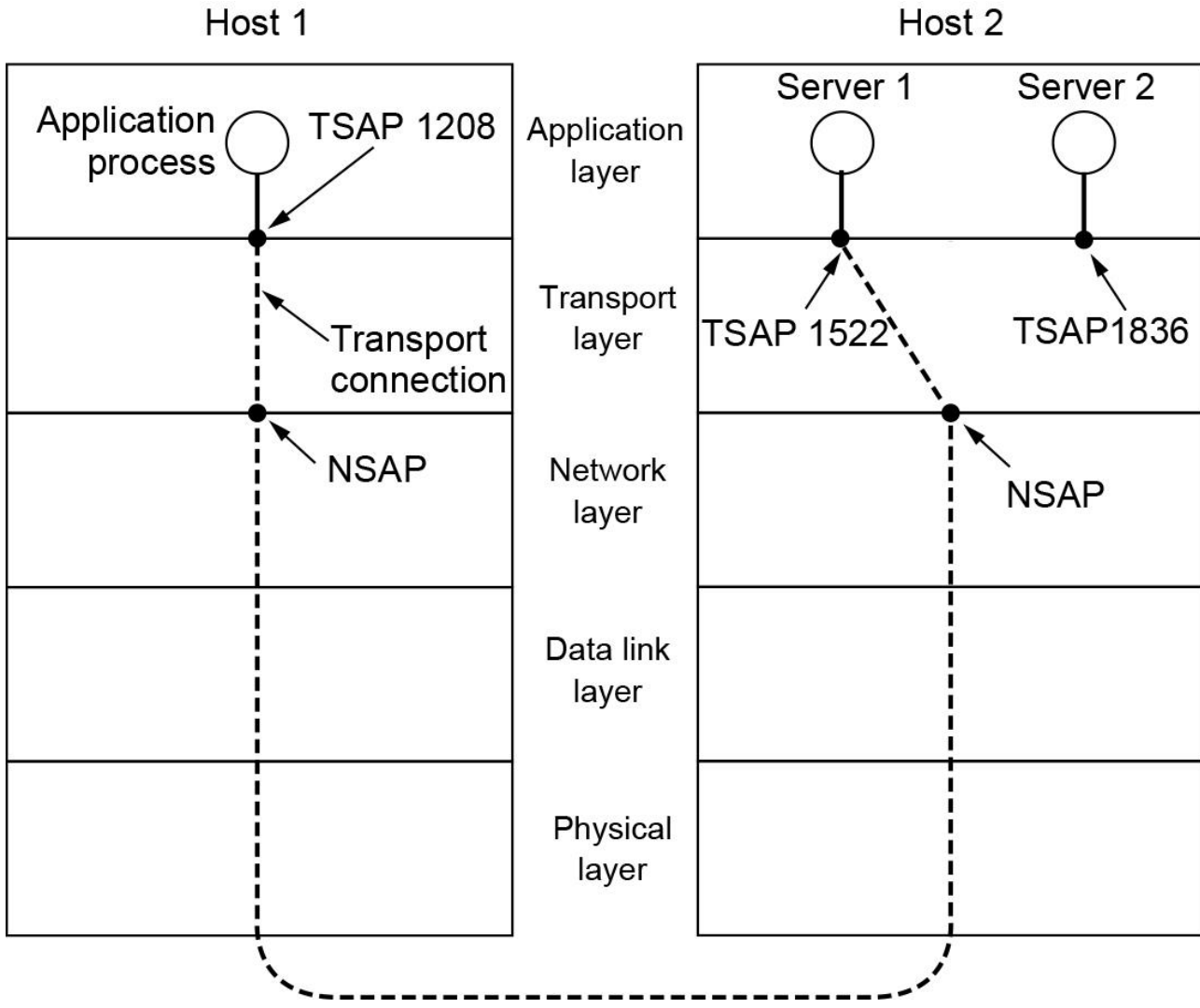
Addressing at the transport layer

- Every host has an IP address, how can multiple connections exist on a single IP?
- We define port numbers or (TSAP) Transport Layer Access Points
  - IP addresses get you to the host (NSAP)
  - Port numbers get you to the process on the host (TSAP)



Let's say a client wants to send an email through a mail server

1. Mail server process attaches to port 1522, calls LISTEN
2. Host 1 wants to send an email, calls CONNECT with source port 1208 and destination port 1522
3. Mail client calls SEND to transmit the email to the server
4. Mail server calls RECEIVE to read the message, sends an acknowledgement
5. Client calls DISCONNECT to terminate the connection



How does the client know to connect on TSAP 1522?

- Port numbers are defined in standards
- They should be chosen to avoid conflict
- Many decades ago Skype and MySQL used the same port numbers, causing issues

- Some common ports
  - 21 – FTP
  - 22 – SSH
  - 69 – TFTP
  - 443 – HTTPS
  - When managing systems, always look up what ports are needed for the firewall rules
  - https://support.skype.com/en/faq/FA148/which-ports-need-to-be-open-to-use-skype-on-desktop

/etc/services contains common port numbers

```
# /etc/services:
# $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
#
# Network services, Internet style
# IANA services version: last updated 2016-07-08
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, ``Assigned Numbers'' (October 1994).  Not all ports
# are included, only the more common ones.
#
# The latest IANA port assignments can be gotten from
#       http://www.iana.org/assignments/port-numbers
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name  port/protocol  [aliases ...]   [# comment]

tcpmux          1/tcp                           # TCP port service multiplexer
tcpmux          1/udp                           # TCP port service multiplexer
rje             5/tcp                           # Remote Job Entry
rje             5/udp                           # Remote Job Entry
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
/etc/services
```

Where did the source port 1208 come from?

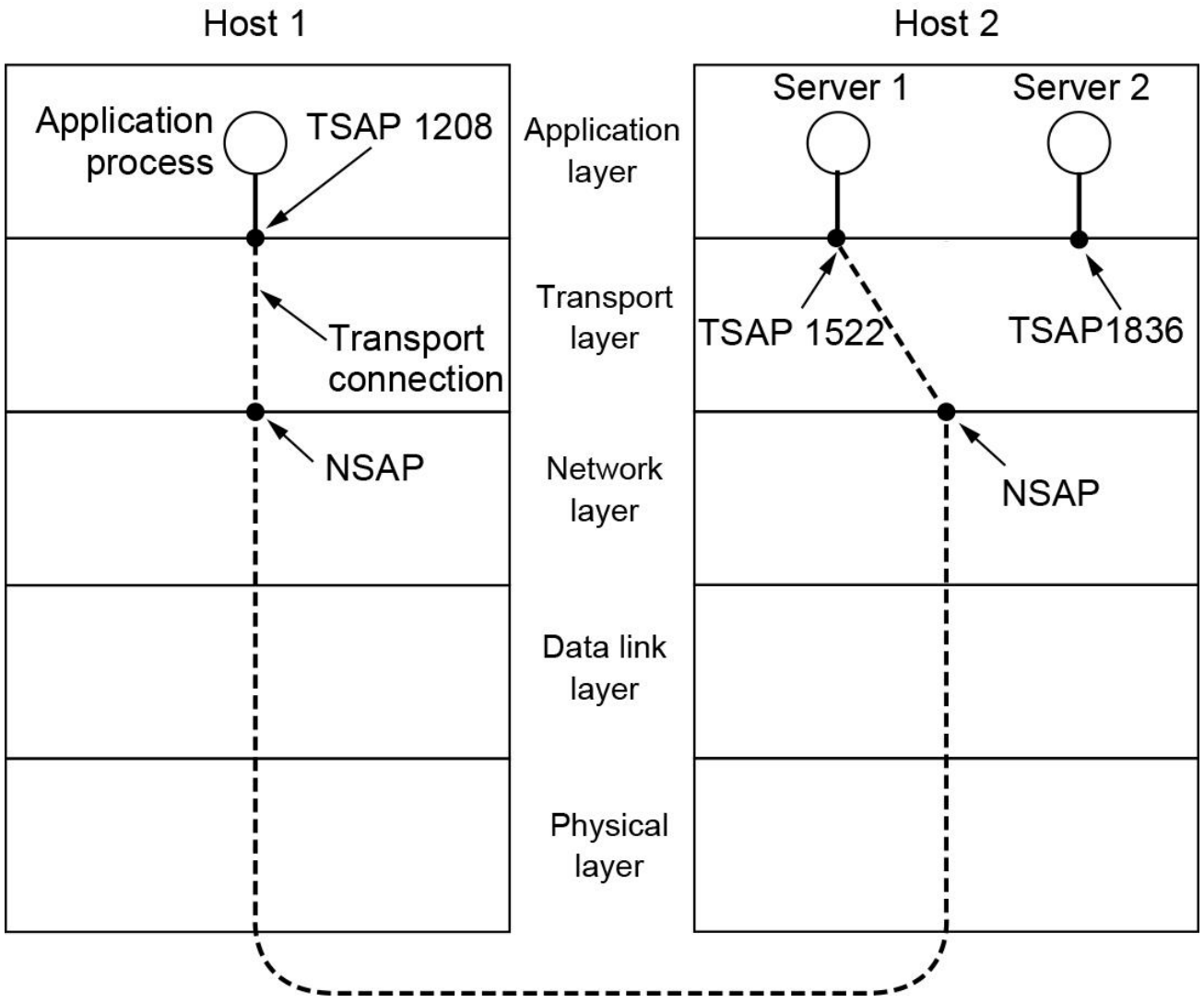
- It is generated by the kernel (transport entity)

- For a connection to be unique, there is a 5-tuple of information that defines a unique connection
  - Source IP
  - Source port
  - Destination IP
  - Destination port
  - Transport Protocol

- *The 5-tuple must be unique per process and per connection*

Multiple servers can be running on a single host

- Because we can have multiple port mappings
- So a mail server may be running on 1522
- And a web server may be running on 1836
- Network layer delivers to host, transport layer decides which application process to send to



An dated analogy for the difference between the network layer and transport layer

- “Snail mail”

- The network layer is like the post office
  - You write an address on an envelope (encapsulating data with address information)
  - Give the envelope to the post office (hand off to the network layer)
  - The post office routes the envelope to the destination address, potentially making multiple stops along the way
  - The envelope arrives at the destination

An dated analogy for the difference between the network layer and transport layer

- “Snail mail”

- The transport layer is the person who checks the mail (not me)
  - Envelopes have arrived from the post office at your house
  - Somebody gathers the mail, then distributes to the individuals in the house
  - Much like receiving packets, and distributing segments to individual processes

- In this analogy, we would say the name it was addressed to is the port number (TSAP) and the address is the IP address (NSAP)

Connection establishment seems simple

- The network layer can't guarantee order, delivery, or lack of duplicates

- How can we solve the issue of duplicate packets?
  - *IDEA: restrict the age of a packet, dropping any packet that is too "old"*
  - This means the transport layer needs to figure out how to deal with old packets, which it can do with

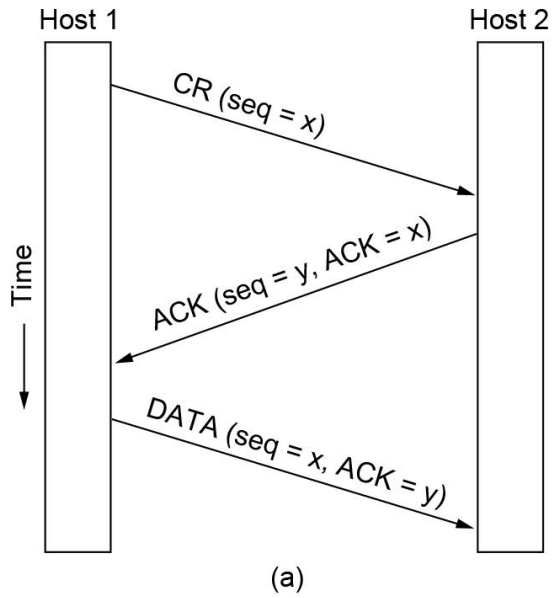
  - Restricting the network design
  - Add a hop counter to each packet
  - Time stamp each packet

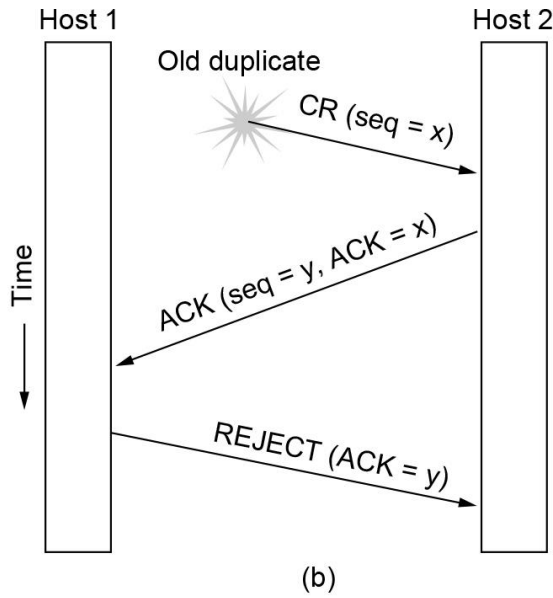
Limiting the packet lifetime

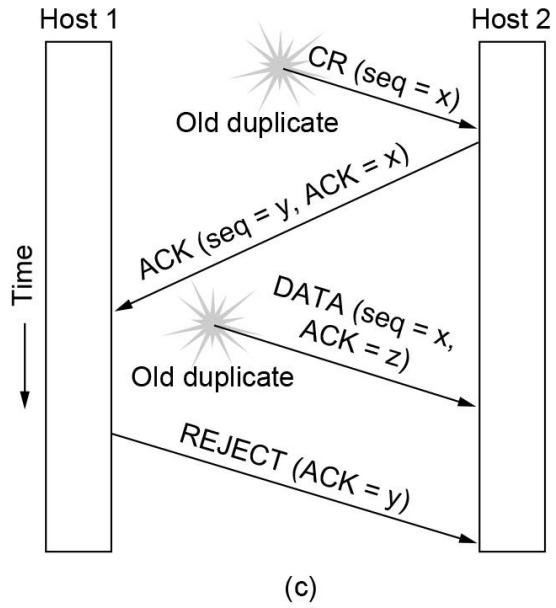
- Restricted network
  - Prevent loops
  - Use a bound congestion delay
- Hop counter
  - TTL on IP as we saw with traceroute
  - When TTL values goes to 0, router drops packet
- Time stamping
  - Would require routers to keep in sync with time, which is non-trivial
- In practice,
  - Hop counter works fine for the age of the packet
  - How long should the transport layer wait for an acknowledgement?

Sequence numbers to deal with packet re-ordering or packet loss

- Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST
- (a) Normal operation
- (b) Old duplicate CONNECTION REQUEST appearing out of nowhere
- (c) Duplicate CONNECTION REQUEST and duplicate ACK







TCP uses a three-way handshake

- Originally sequence numbers were based on a clock
- This clock would still be running even if the computer crashed
- Leads to predictable sequence numbers, which is not good as hackers can then potentially inject data

TCP uses three-way handshake

- To establish a 32-bit sequence number
- In addition, a timestamp is used to extend the sequence number so that it will not wrap within the max packet lifetime
  - This was added once gigabit-per-second links existed
  - PAWS (Protection Against Wrapped Sequence numbers)

- Sequence numbers are generated in a pseudorandom fashion
  - Which means they are not truly random, there is some method behind the madness
  - But to any outside observer, the data appears random

Initial Sequence Number (ISN)

- *The sequence number must be chosen with some care*
- Otherwise TCP will accept packets arriving if they match the 4-tuple and have carefully crafted rogue sequence number

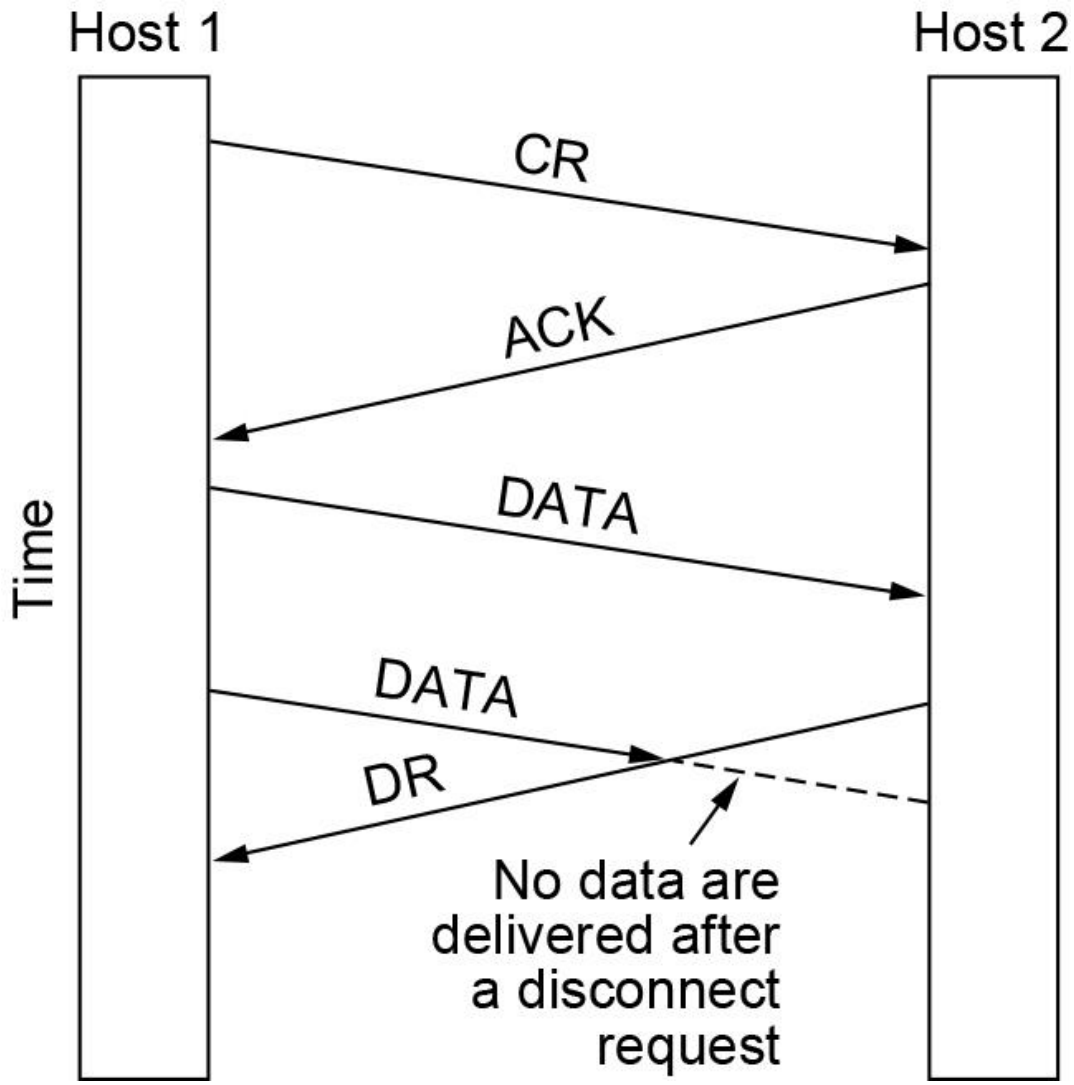
```
u32 secure_tcp_seq(__be32 saddr, __be32 daddr,
                   __be16 sport, __be16 dport)
{
        u32 hash;

        net_secret_init();
        hash = siphash_3u32((__force u32)saddr, (__force u32)daddr,
                            (__force u32)sport << 16 | (__force u32)dport,
                             &net_secret);
        return seq_scale(hash);
}
```

Releasing a connection

- Asymmetric release
  - Telephone system works this way
  - Ever hang up on someone?

- Symmetric release
  - Data is travelling in both directions, it is important that both sides agree to be done transmitting
  - (though in some cases one side can essentially hang up)

Abrupt disconnections cause data loss

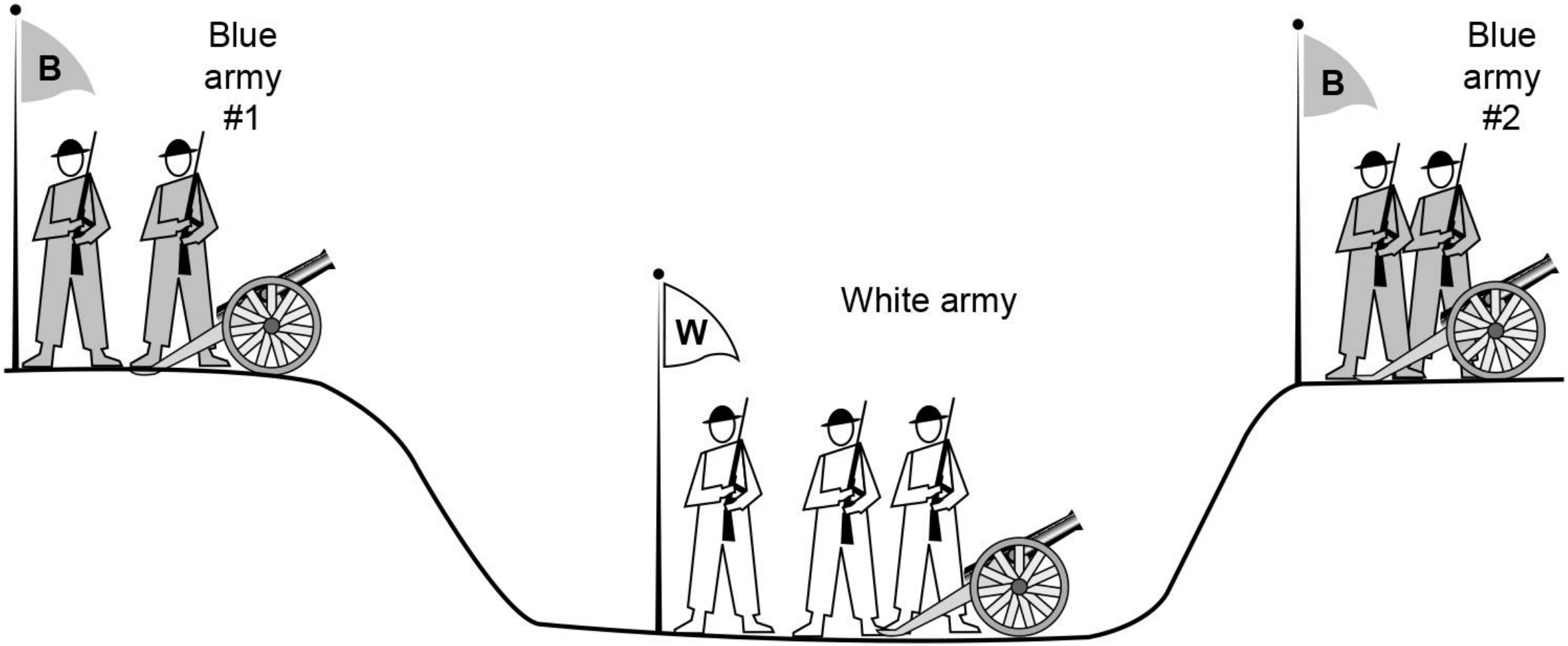


So how do we quit?

- “I’m done, are you done?”
- “Yes, I’m done, you can hang up now”
- “Okay, bye”
- “bye”
- Awkward silence…
- “you hang up first”
- “okay bye”
- “bye”
- ….

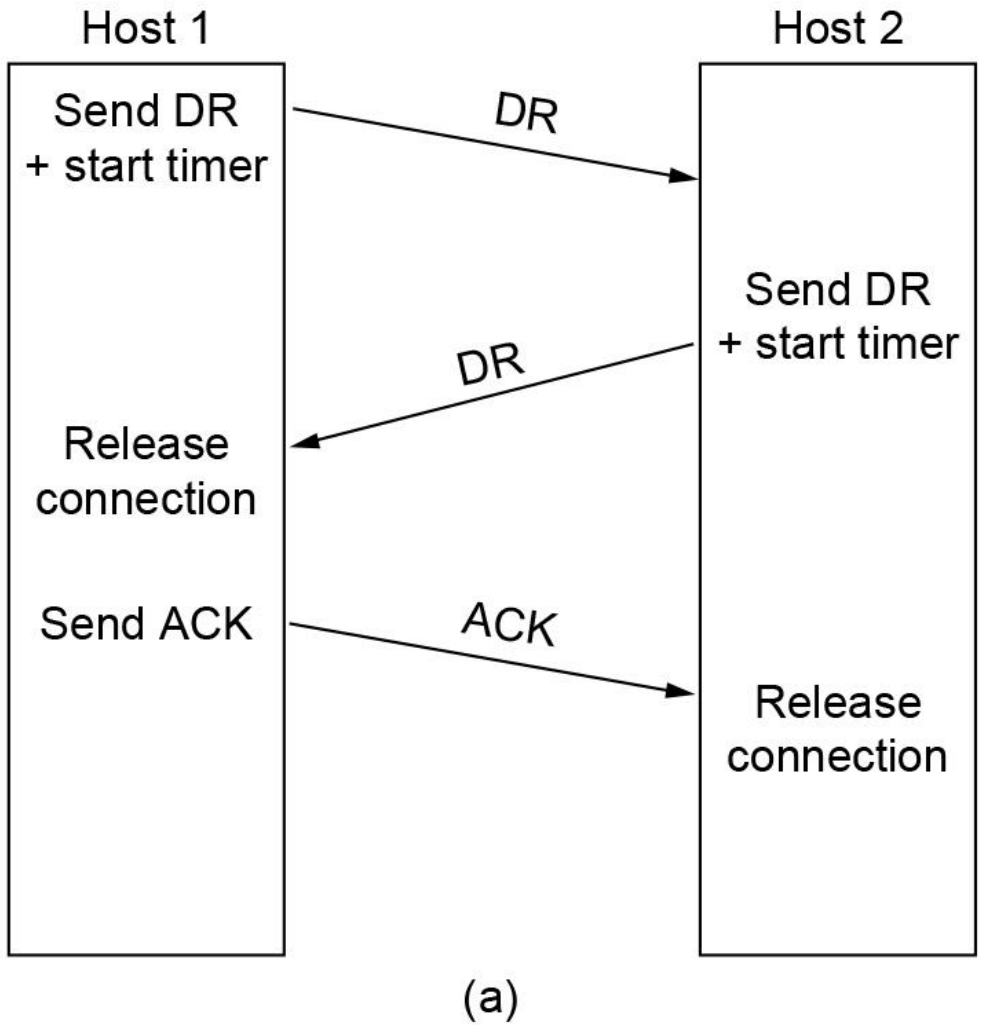
There is no theoretical solution to this problem

- Though in practice, the solution is good enough

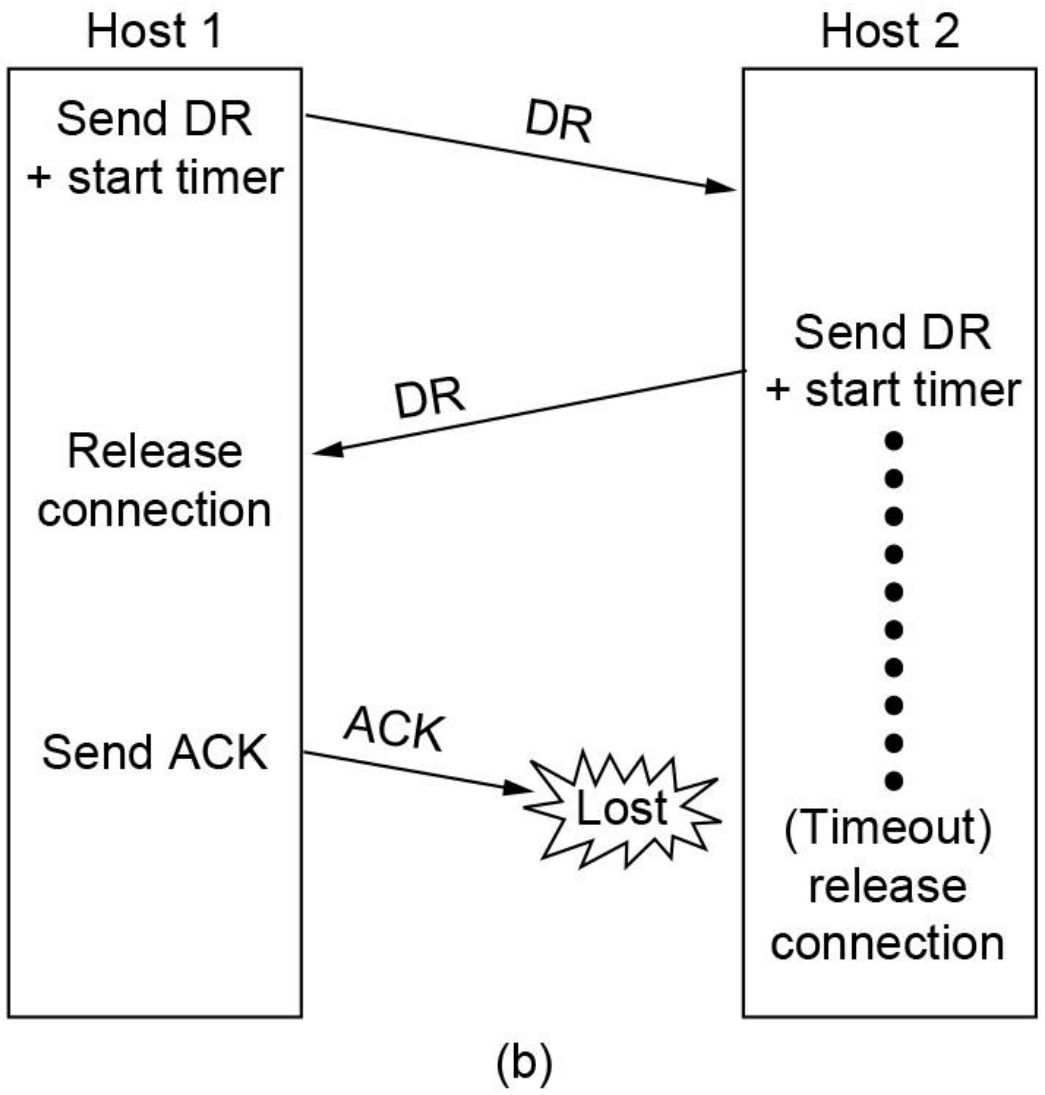


DISCONNECTION REQUEST

- The normal case, where a user sends a DR and starts a timer incase its DR is lost
- The server then sends its own DR and starts a timer
- The user sends an ACK to say it got the DR and the connection can be released on both sides

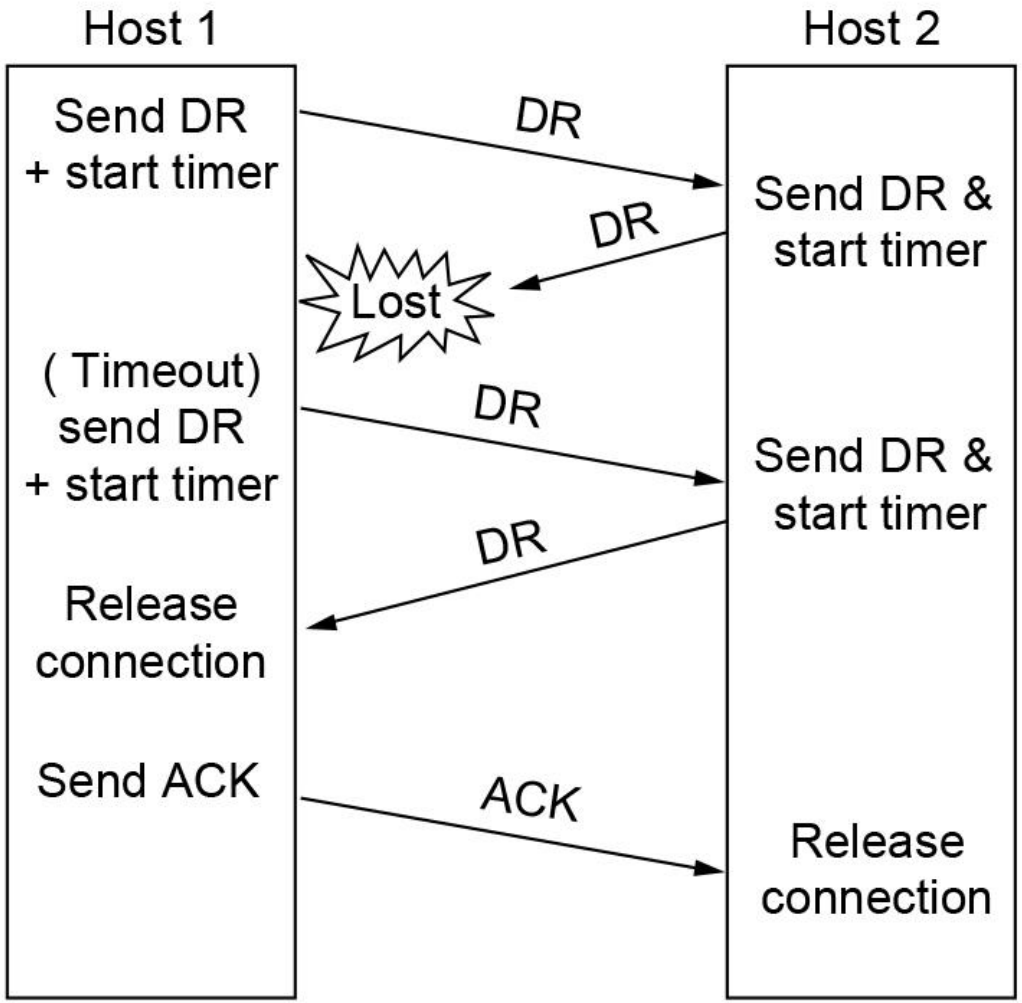


If the ACK is lost, the timer will timeout



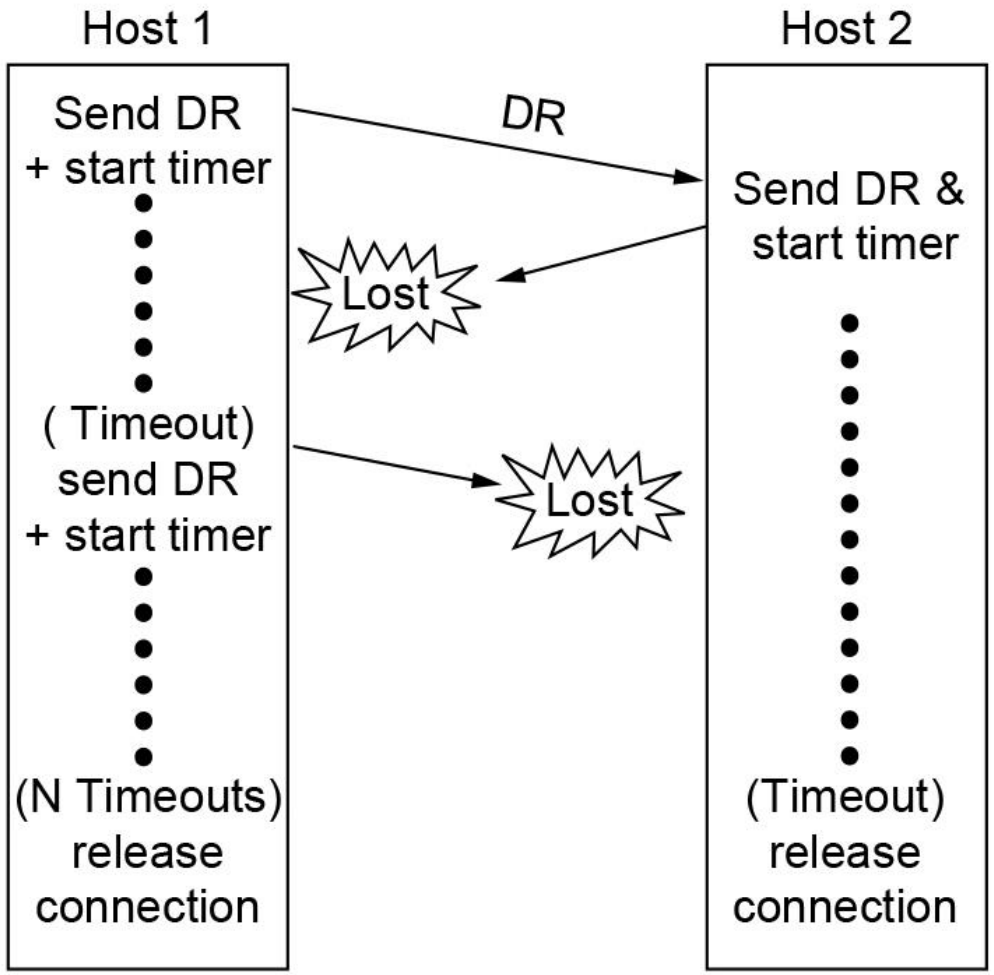
(b)

If the user DR is lost on the way or the DR from the server is lost on the way back, there is a timeout



(c)

If the server stops receiving packets after DR, user will keep trying N times, before quitting



(d)

Web Clients may perform an abrupt close

- While disconnecting seems more complicated than it first appears, in some cases it can be simple
- The application layer can sometimes handle situations where it makes sense to “hang up” on a client abruptly
- For instance, a web server can send the client an RST once all of the data has been transmitted to abruptly terminate the connection
  - This works because the web server knows the pattern of the data exchange