# A Quick Review of Computer Networking

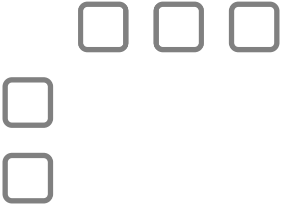
## Architecture, Applications, Transport (TCP), Routing

Antonio Carzaniga

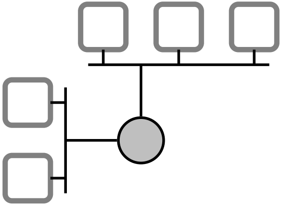
Faculty of Informatics
Università della Svizzera italiana

February 19, 2020

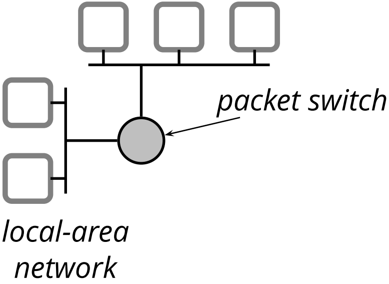
# What is *Inside* the Internet?



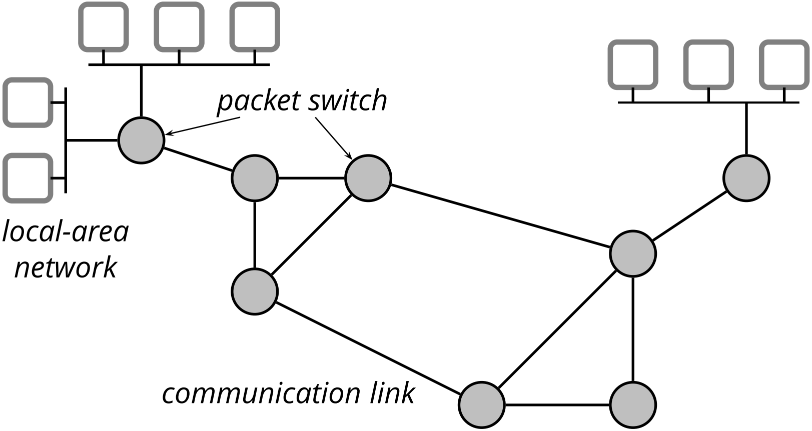
# What is *Inside* the Internet?



# What is *Inside* the Internet?



# What is *Inside* the Internet?



# Basic Concepts

- The Internet uses ***packet switching***

# Basic Concepts

- The Internet uses ***packet switching***

- ***Packet switch:*** a *link-layer switch* or a ***router***

# Basic Concepts

- The Internet uses ***packet switching***
- ***Packet switch:*** a *link-layer switch* or a ***router***
- ***Communication link:*** a connection between packet switches and/or end systems

# Basic Concepts

- The Internet uses ***packet switching***
- ***Packet switch:*** a *link-layer switch* or a ***router***
- ***Communication link:*** a connection between packet switches and/or end systems
- ***Route:*** sequence of switches that a packet goes through (a.k.a. *path*)

# Basic Concepts

- The Internet uses ***packet switching***
- ***Packet switch:*** a *link-layer switch* or a ***router***
- ***Communication link:*** a connection between packet switches and/or end systems
- ***Route:*** sequence of switches that a packet goes through (a.k.a. *path*)
- ***Protocol:*** control the sending and receiving of information to and from end systems and packet switches

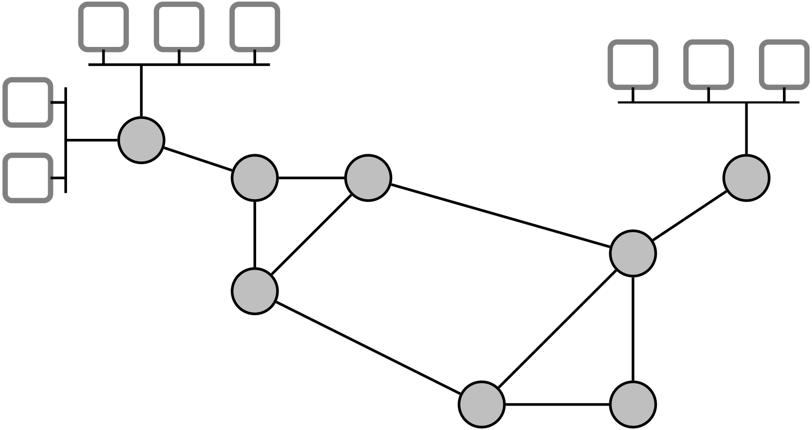
# Communication Links

- Various types and forms of medium

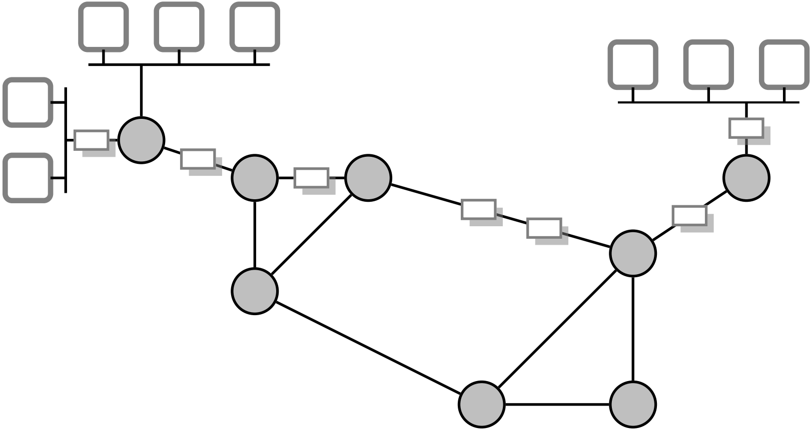
# Communication Links

- Various types and forms of medium
    - Fiber-optic cable
    - Twisted-pair copper wire
    - Coaxial cable
    - Wireless local-area links (e.g., 802.11, Bluetooth)
    - Satellite channel
    - . . .

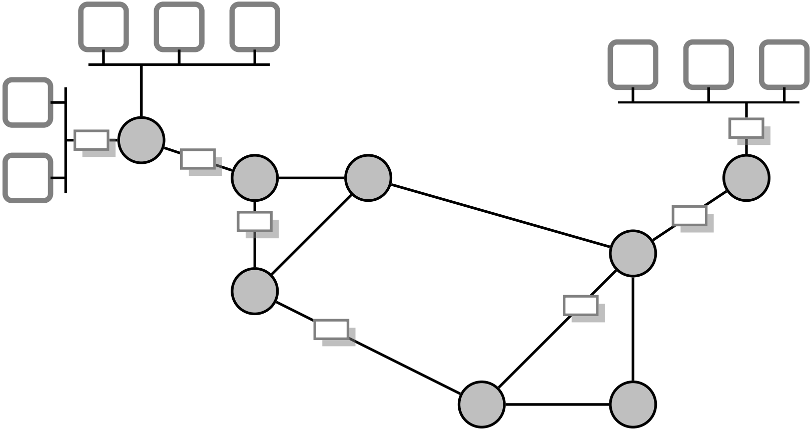
# Packet Switching



# Packet Switching



# Packet Switching



# Packet Switching

- The Internet is a *packet-switched* network

# Packet Switching

- The Internet is a *packet-switched* network
- Information is transmitted in *packets*

# Packet Switching

- The Internet is a *packet-switched* network
- Information is transmitted in *packets*
- Switches operate on individual packets

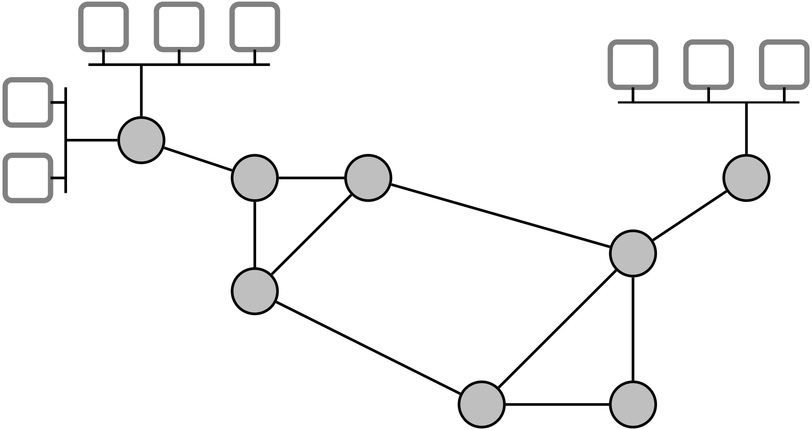
# Packet Switching

- The Internet is a *packet-switched* network
- Information is transmitted in *packets*
- Switches operate on individual packets
- A switch (router) receives packets and *forwards* them along to other switches or to end systems

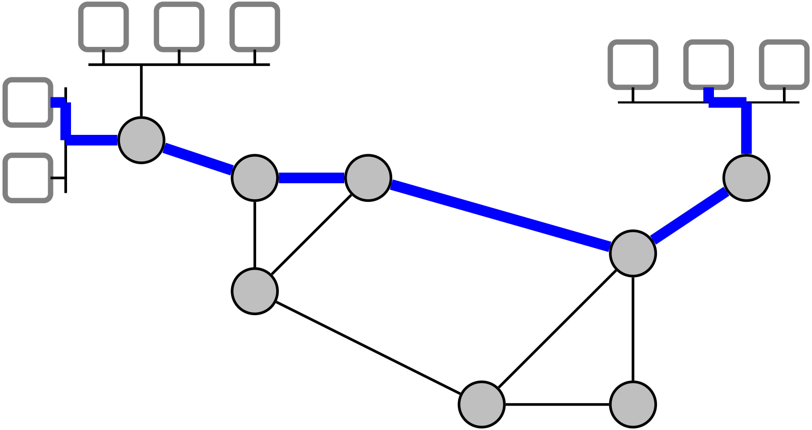
# Packet Switching

- The Internet is a *packet-switched* network
- Information is transmitted in *packets*
- Switches operate on individual packets
- A switch (router) receives packets and *forwards* them along to other switches or to end systems
- Every forwarding decision is taken on the basis of the information contained in the packet

# Circuit Switching



# Circuit Switching



# Circuit Switching

- The telephone network is a typical circuit-switched network
  - not any more, really, but still. . .

# Circuit Switching

- The telephone network is a typical circuit-switched network
    - not any more, really, but still. . .

- Communication requires a ***connection setup*** phase in which the network reserves all the necessary resources for that connection (links, buffers, switches, etc.)

# Circuit Switching

- The telephone network is a typical circuit-switched network
  - not any more, really, but still. . .

- Communication requires a ***connection setup*** phase in which the network reserves all the necessary resources for that connection (links, buffers, switches, etc.)

- After a successful setup, the communicating systems are connected by ***a set of links dedicated to the connection*** for the entire duration of their conversation

# Circuit Switching

- The telephone network is a typical circuit-switched network
    - not any more, really, but still. . .

- Communication requires a ***connection setup*** phase in which the network reserves all the necessary resources for that connection (links, buffers, switches, etc.)

- After a successful setup, the communicating systems are connected by ***a set of links dedicated to the connection*** for the entire duration of their conversation

- When the conversation ends, the network tears down the connection, freeing the corresponding resources (links, buffers, etc.) for other connections

# Circuit vs. Packet Switching

# Circuit vs. Packet Switching

- Circuit switching requires an expensive setup phase
  - however, once the connection is established, little or no processing is required

# Circuit vs. Packet Switching

- Circuit switching requires an expensive setup phase
    - however, once the connection is established, little or no processing is required

- Packet switching does not incur any setup cost
    - however, it always incurs a significant processing and space overhead, on a per-packet basis
        - *processing cost* for forwarding
        - *space overhead* because every packet must be self-contained

# Circuit vs. Packet Switching (2)

# Circuit vs. Packet Switching (2)

- Circuit switching admits a straightforward implementation of quality-of-service guarantees
  - network resources are reserved at connection setup time

# Circuit vs. Packet Switching (2)

- Circuit switching admits a straightforward implementation of quality-of-service guarantees
    - network resources are reserved at connection setup time

- Guaranteeing any quality of service with packet switching is very difficult
    - no concept of a “connection”
    - and again, processing, space overhead, etc.

# Circuit vs. Packet Switching (3)

- Circuit switching allows only a limited sharing of communication resources
  - once a connection is established, the resources are blocked even though there might be long silence periods
  - i.e., circuit switching is an inefficient way to use the network

# Circuit vs. Packet Switching (3)

- Circuit switching allows only a limited sharing of communication resources
  - once a connection is established, the resources are blocked even though there might be long silence periods
  - i.e., circuit switching is an inefficient way to use the network

- Packet switching achieves a much better utilization of network resources
  - it is designed specifically to share links
  - it is designed specifically to share links

# Virtual Circuits

- Idea: combine the advantages of circuit switching and packet switching

# Virtual Circuits

- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase

# Virtual Circuits

- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase
- The connection does not create a physical circuit, but rather a “virtual circuit”

# Virtual Circuits

- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase
- The connection does not create a physical circuit, but rather a “virtual circuit”
- Information is sent in packets, so links can be shared more effectively

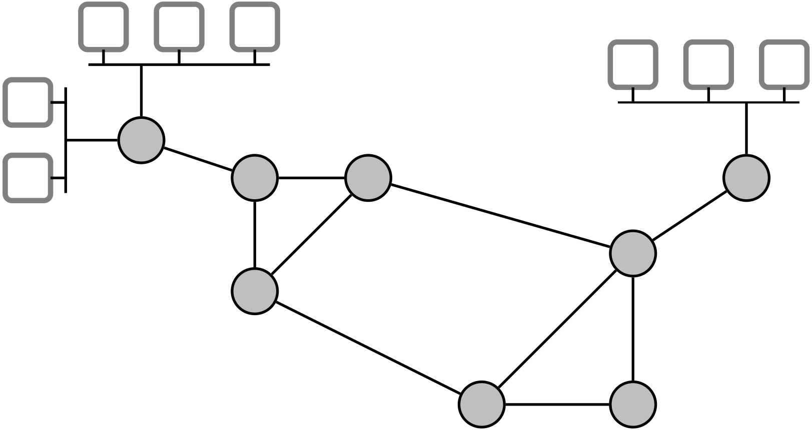
# Virtual Circuits

- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase
- The connection does not create a physical circuit, but rather a “virtual circuit”
- Information is sent in packets, so links can be shared more effectively
- Packets carry a *virtual circuit identifier* instead of the destination address

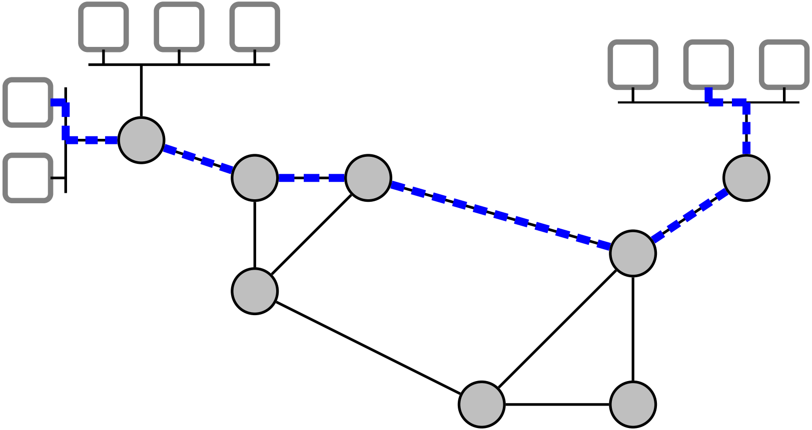
# Virtual Circuits

- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase
- The connection does not create a physical circuit, but rather a “virtual circuit”
- Information is sent in packets, so links can be shared more effectively
- Packets carry a *virtual circuit identifier* instead of the destination address
  - *Important observation:* at any given time there are much fewer *connections* than *destinations*
    - much faster per-packet processing (forwarding)
    - lower per-packet space overhead

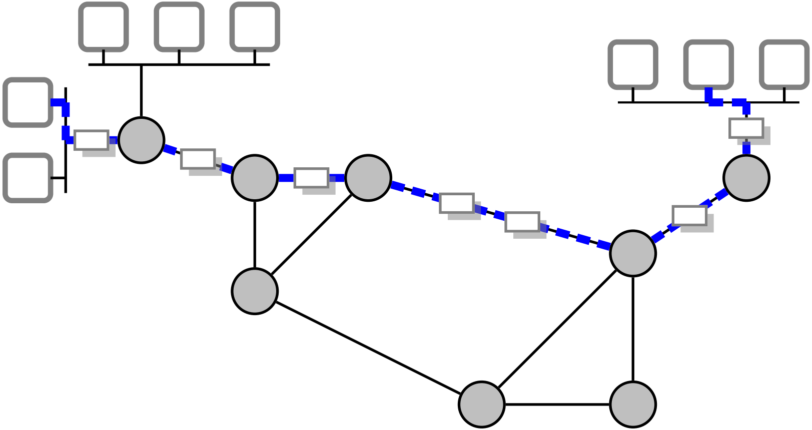
# Virtual Circuit



# Virtual Circuit



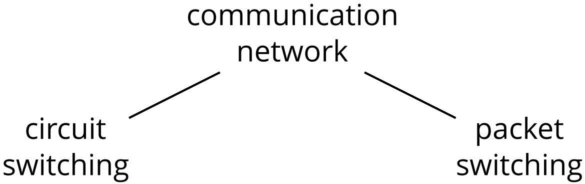
# Virtual Circuit



# Taxonomy of Networks

communication
network

# Taxonomy of Networks

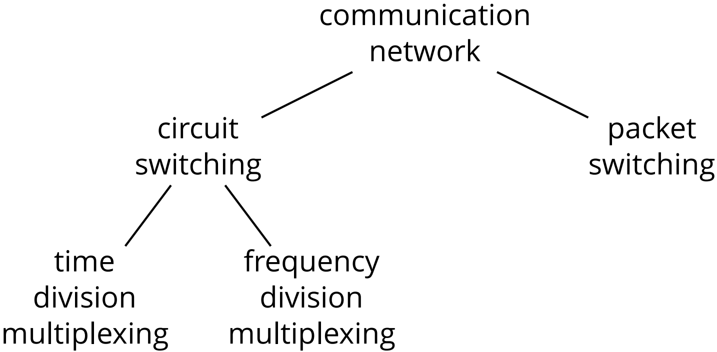


communication network

circuit switching

packet switching

# Taxonomy of Networks



communication network

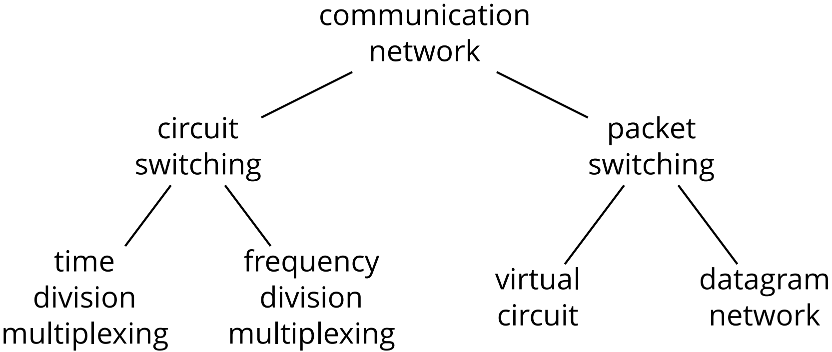
circuit switching

packet switching

time division multiplexing

frequency division multiplexing

# Taxonomy of Networks



communication network

circuit switching

packet switching

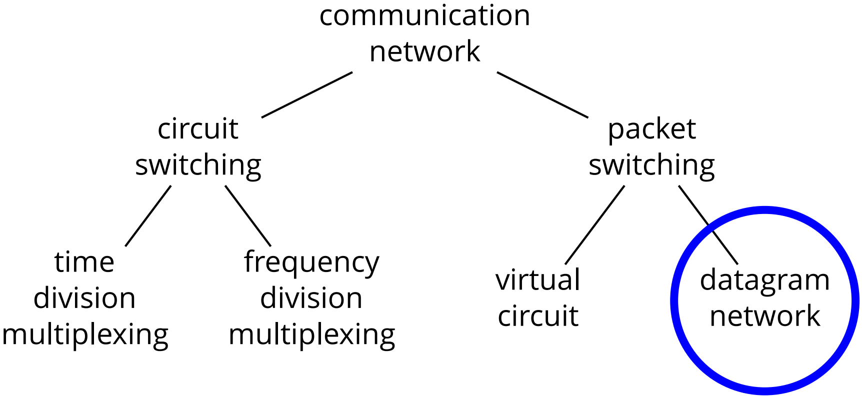
time division multiplexing

frequency division multiplexing

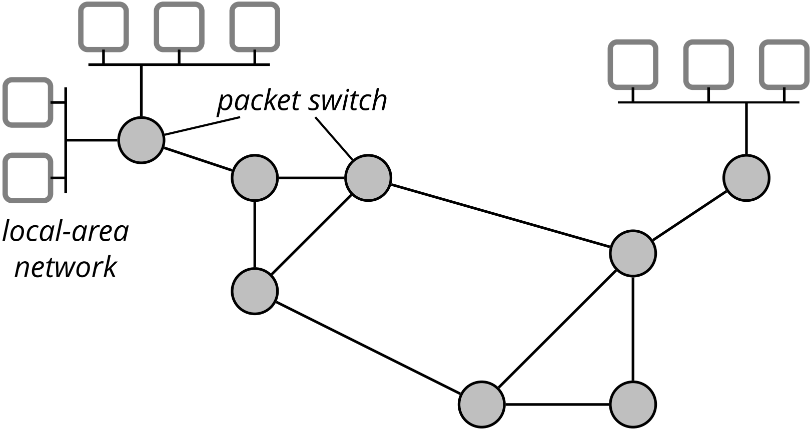
virtual circuit

datagram network

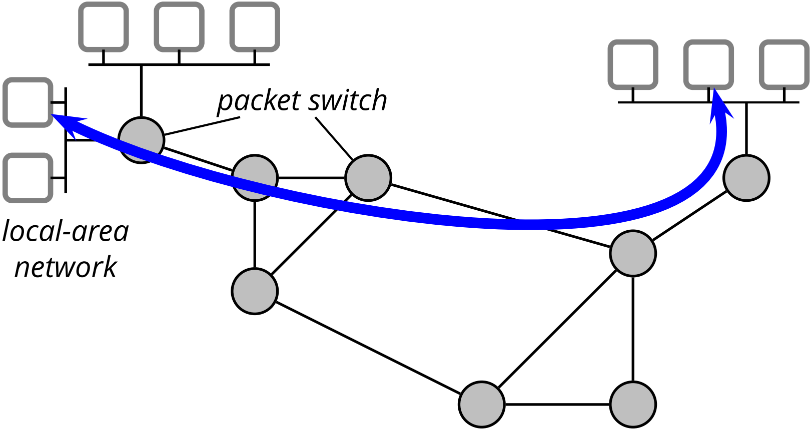
# Taxonomy of Networks



# Service Perspective



# Service Perspective



- What kind of ***service*** does the Internet offer to end systems?

# Type of Service

- Two end systems can communicate through the Internet, but exactly what kind of communication service is that of the Internet?

# Type of Service

- Two end systems can communicate through the Internet, but exactly what kind of communication service is that of the Internet?

- ***Connectionless, “best effort”***
  - the network accepts “datagrams” for delivery—this is conceptually similar to the postal service
  - “best effort” really means *unreliable* though not malicious

# Type of Service

- Two end systems can communicate through the Internet, but exactly what kind of communication service is that of the Internet?

- ***Connectionless, "best effort"***
  - the network accepts "datagrams" for delivery—this is conceptually similar to the postal service
  - "best effort" really means *unreliable* though not malicious

- ***Connection-oriented, reliable***
  - virtual duplex communication channel ($A \leftrightarrow B$)—conceptually similar to a telephone service
  - information is transmitted "reliably" and in order

# Type of Service (2)

- How reliable is a “reliable” service?

# Type of Service (2)

- How reliable is a “reliable” service?

- The term “reliable” means that information will eventually reach its destination if a route is viable within a certain amount of time

# Type of Service (2)

- How reliable is a “reliable” service?
- The term “reliable” means that information will eventually reach its destination if a route is viable within a certain amount of time
- The network makes absolutely no guarantees on *latency* (i.e., the time it takes to transmit some information from a source to a destination)

# Internet Protocol Stack

# Internet Protocol Stack

application

# Internet Protocol Stack

application

transport

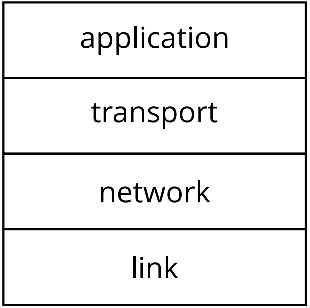
# Internet Protocol Stack

application

transport

network

# Internet Protocol Stack



application

transport

network

link

# Internet Protocol Stack

application
transport
network
link
physical

# Internet Protocol Stack (2)

- *Application* (e.g., HTTP, SMTP, and DNS)
  - application functionalities
  - application messages

# Internet Protocol Stack (2)

- *Application* (e.g., HTTP, SMTP, and DNS)
  - application functionalities
  - application messages

- *Transport* (e.g., TCP and UDP)
  - application multiplexing, reliable transfer (TCP), congestion control (TCP)
  - datagrams (UDP) or segments (TCP)

# Internet Protocol Stack (2)

- *Application* (e.g., HTTP, SMTP, and DNS)
  - application functionalities
  - application messages

- *Transport* (e.g., TCP and UDP)
  - application multiplexing, reliable transfer (TCP), congestion control (TCP)
  - datagrams (UDP) or segments (TCP)

- *Network* (IP)
  - end to end datagram, best-effort service, routing, fragmentation
  - packets (IP)

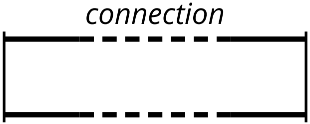
# Internet Protocol Stack (2)

- *Application* (e.g., HTTP, SMTP, and DNS)
    - application functionalities
    - application messages
- *Transport* (e.g., TCP and UDP)
    - application multiplexing, reliable transfer (TCP), congestion control (TCP)
    - datagrams (UDP) or segments (TCP)
- *Network* (IP)
    - end to end datagram, best-effort service, routing, fragmentation
    - packets (IP)
- *Link* (e.g., Ethernet and PPP)
    - point-to-point or local broadcast communication
    - frames (or packets)

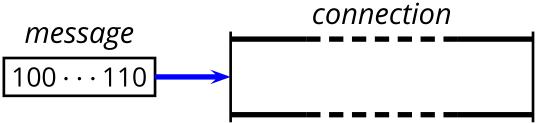
# Internet Protocol Stack (2)

- *Application* (e.g., HTTP, SMTP, and DNS)
  - application functionalities
  - application messages
- *Transport* (e.g., TCP and UDP)
  - application multiplexing, reliable transfer (TCP), congestion control (TCP)
  - datagrams (UDP) or segments (TCP)
- *Network* (IP)
  - end to end datagram, best-effort service, routing, fragmentation
  - packets (IP)
- *Link* (e.g., Ethernet and PPP)
  - point-to-point or local broadcast communication
  - frames (or packets)
- *Physical*

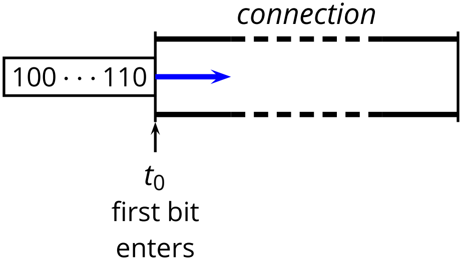
# Delay (Latency) and Rate (Throughput)



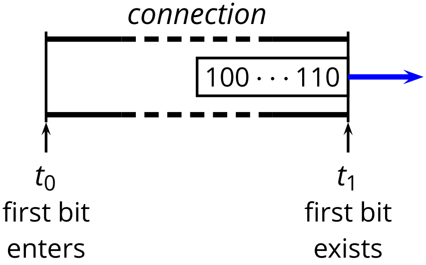
# Delay (Latency) and Rate (Throughput)



# Delay (Latency) and Rate (Throughput)



# Delay (Latency) and Rate (Throughput)



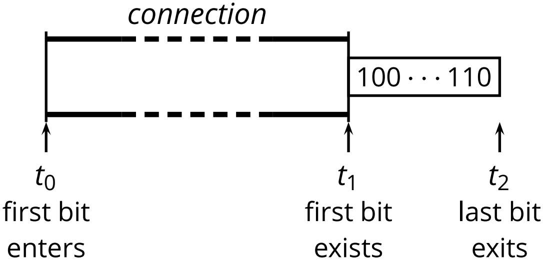
*connection*

100 · · · 110

$t_0$
first bit
enters

$t_1$
first bit
exists

# Delay (Latency) and Rate (Throughput)



*connection*

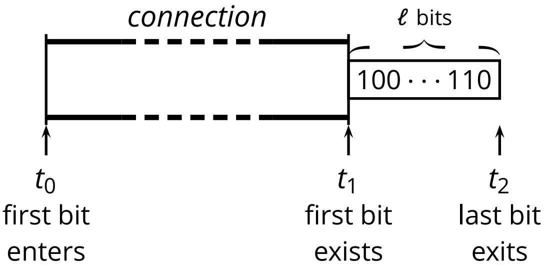
100 · · · 110

$t_0$
first bit
enters

$t_1$
first bit
exists

$t_2$
last bit
exits

# Delay (Latency) and Rate (Throughput)



*connection*

$\ell$ bits

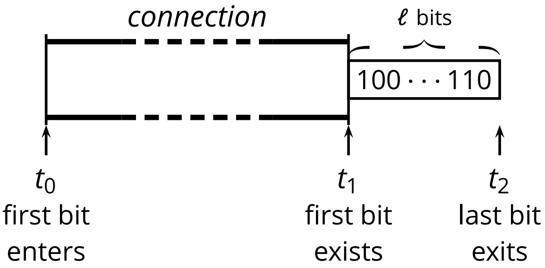
$100 \cdots 110$

$t_0$
first bit
enters

$t_1$
first bit
exists

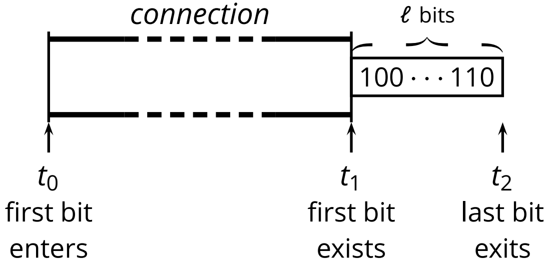
$t_2$
last bit
exits

# Delay (Latency) and Rate (Throughput)



*Propagation* ***Delay*** $d_{prop} = t_1 - t_0$ sec

# Delay (Latency) and Rate (Throughput)



*connection*

$\ell$ bits

$\underbrace{100 \cdots 110}$

$t_0$ first bit enters

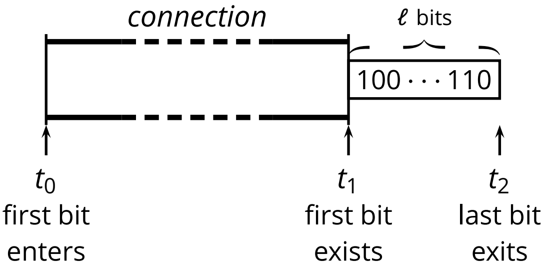
$t_1$ first bit exists

$t_2$ last bit exits

*Propagation* ***Delay*** $d_{prop} = t_1 - t_0$ sec

*Transmission* ***Rate*** $R = \dfrac{\ell}{t_2 - t_1}$ bits/sec

# Delay (Latency) and Rate (Throughput)



*connection*

$\ell$ bits

$100 \cdots 110$

$t_0$
first bit
enters

$t_1$
first bit
exists

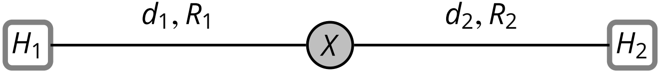
$t_2$
last bit
exits

| | | |
|---|---|---|
| *Propagation **Delay*** | $d_{prop} = t_1 - t_0$ | sec |
| *Transmission **Rate*** | $R = \dfrac{\ell}{t_2 - t_1}$ | bits/sec |
| *Total transfer time* | $d_{end\text{-}end} = d + \dfrac{\ell}{R}$ | sec |

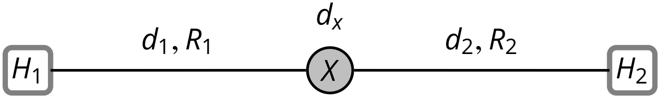
# Store-And-Forward Delay

# Store-And-Forward Delay

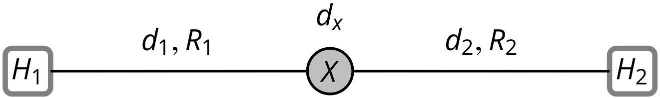
# Store-And-Forward Delay



# Store-And-Forward Delay

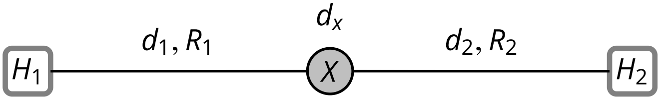


# Store-And-Forward Delay



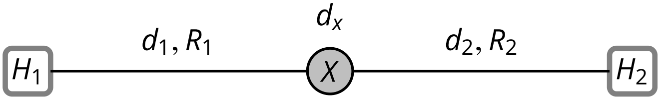
$$d_{\text{end-end}} = d_1 + \frac{\ell}{R_1}$$

# Store-And-Forward Delay



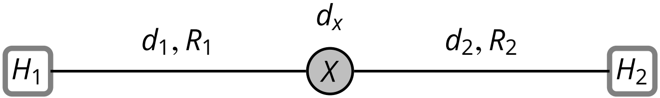
$$d_{end\text{-}end} = d_1 + \frac{\ell}{R_1} + d_x$$

# Store-And-Forward Delay



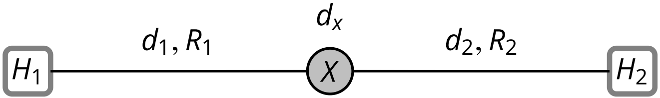
$$d_{end\text{-}end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2}$$

# Store-And-Forward Delay

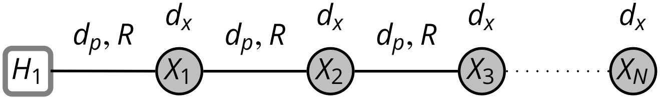


$$d_{\text{end-end}} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2} + d_2$$

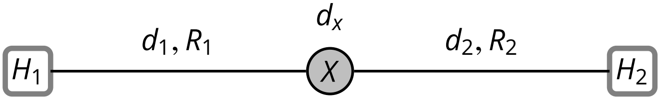
# Store-And-Forward Delay



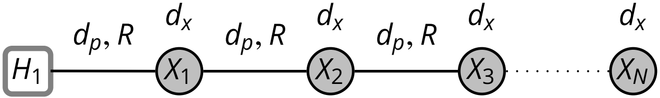
$$d_{end\text{-}end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2} + d_2$$



# Store-And-Forward Delay



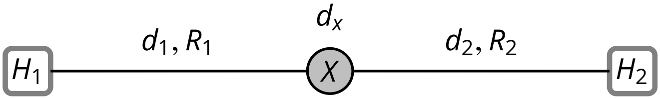
$$d_{end\text{-}end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2} + d_2$$



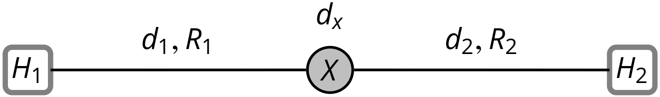
$$d_{end\text{-}end} = N\left(d_p + \frac{\ell}{R} + d_x\right)$$

# End-toEnd Throughput

# End-toEnd Throughput

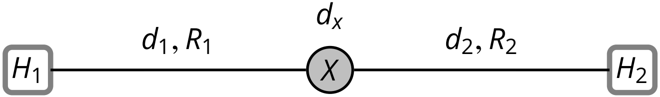


# End-toEnd Throughput



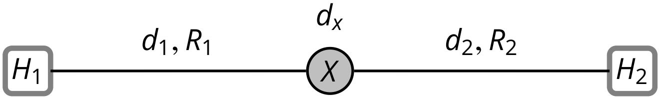
$$R_{\textit{end-end}} =$$

# End-toEnd Throughput

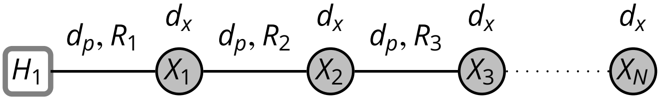


$$R_{end\text{-}end} = \min\{R_1, R_2\}$$

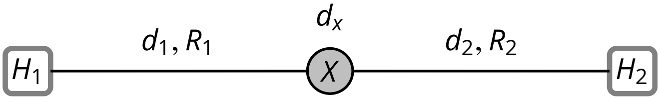
# End-toEnd Throughput



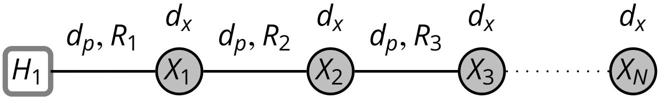
$$R_{\text{end-end}} = \min\{R_1, R_2\}$$



# End-toEnd Throughput



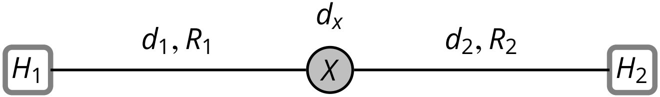
$$R_{end\text{-}end} = \min\{R_1, R_2\}$$



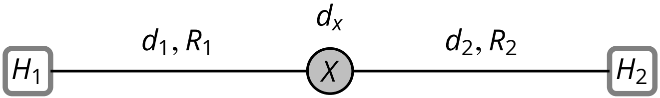
$$R_{end\text{-}end} = \min\{R_1, R_2, \ldots, R_N\}$$

# Processing and Queuing Delays

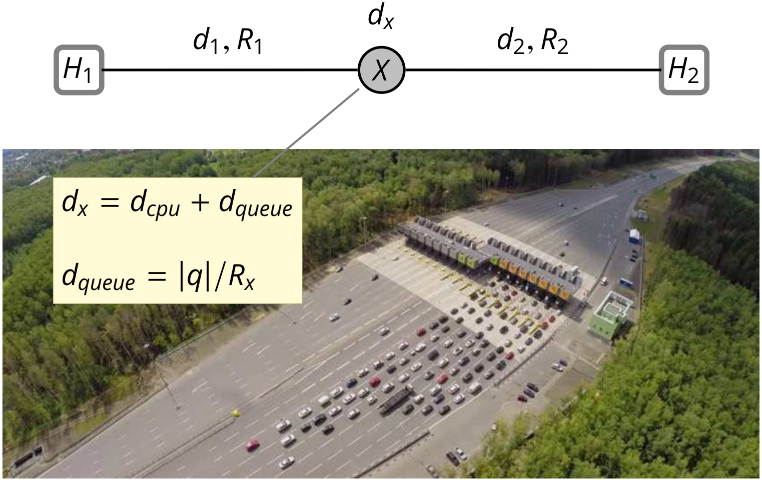
# Processing and Queuing Delays



# Processing and Queuing Delays



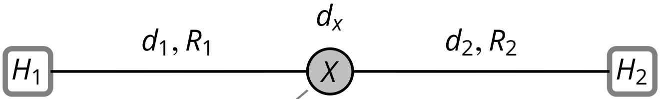
# Processing and Queuing Delays



$$d_x = d_{cpu} + d_{queue}$$

$$d_{queue} = |q|/R_x$$

# Processing and Queuing Delays



$$d_x = d_{cpu} + d_{queue}$$

$$d_{queue} = |q|/R_x$$

queue length

output rate

# Queuing Delay

$$d_x = d_{cpu} + d_{queue}$$

where

queue length

$$d_{queue} = |q|/R_x$$

output rate

# Queuing Delay

$$d_x = d_{cpu} + d_{queue}$$

where

$$d_{queue} = \overbrace{|q|}^{\text{queue length}} / \underbrace{R_x}_{\text{output rate}}$$

. . . $R_x$ is also the rate at which packets get out of the queue

# Application Protocols

# Application Protocols

HTTP

# Application Protocols

HTTP

SMTP

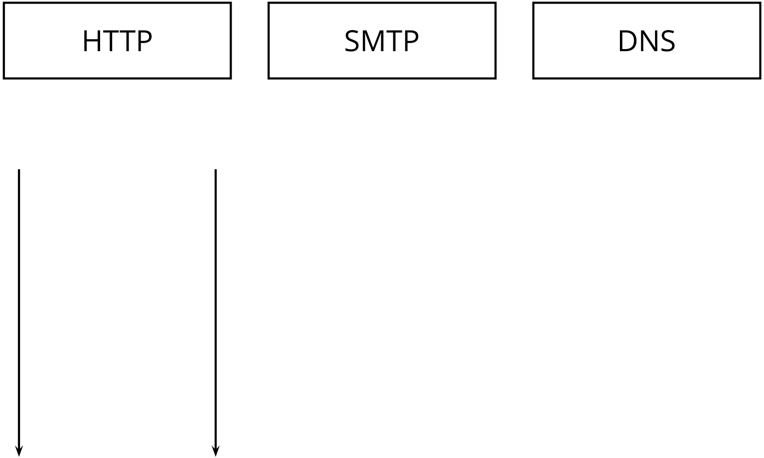
# Application Protocols

HTTP

SMTP

DNS

# Application Protocols



HTTP

SMTP

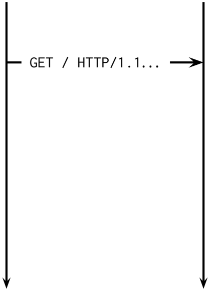
DNS

# Application Protocols

HTTP

SMTP

DNS

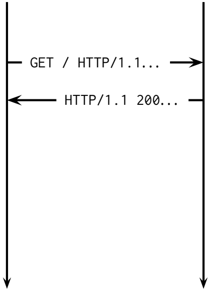


# Application Protocols

HTTP

SMTP

DNS

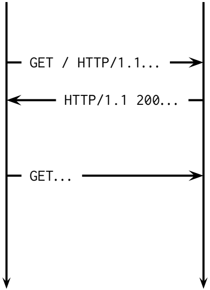


# Application Protocols

HTTP

SMTP

DNS

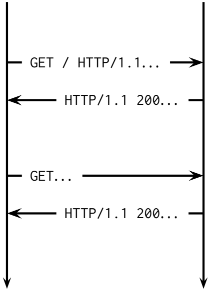


# Application Protocols

HTTP

SMTP

DNS



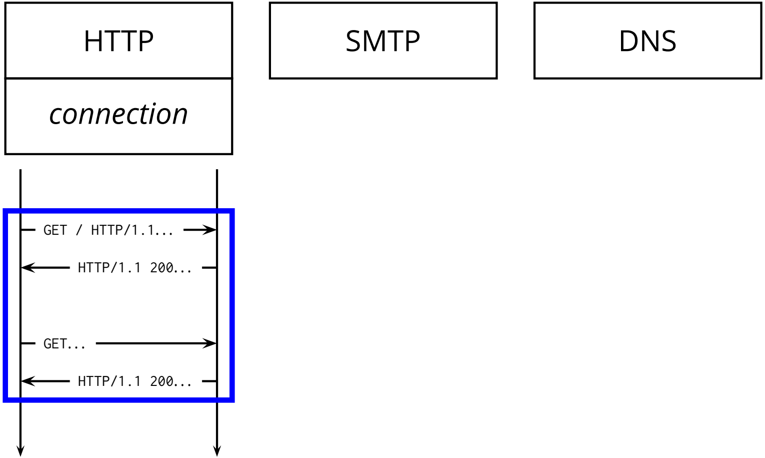
GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

# Application Protocols



HTTP

*connection*

SMTP

DNS

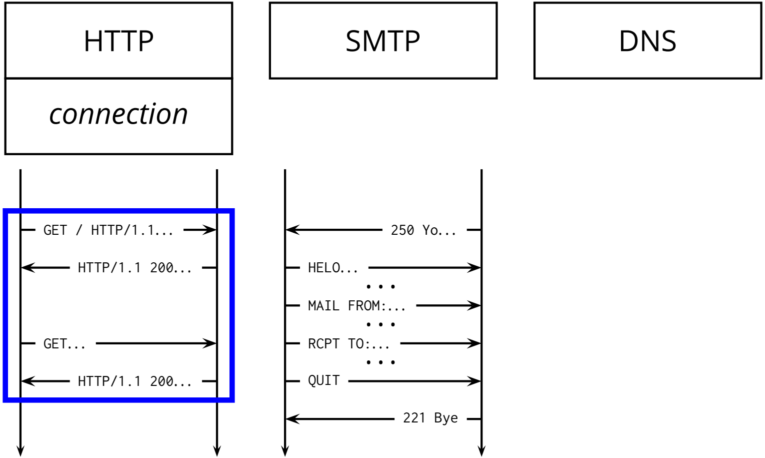
GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

# Application Protocols



HTTP

*connection*

SMTP

DNS

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

250 Yo...

HELO...

...

MAIL FROM:...

...

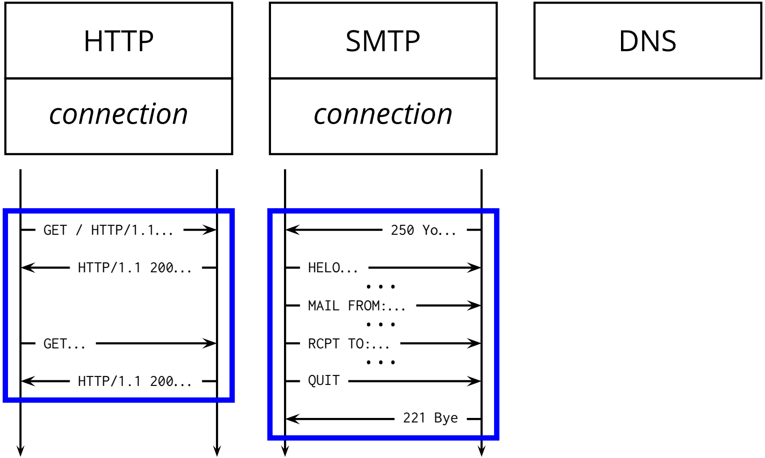
RCPT TO:...

...

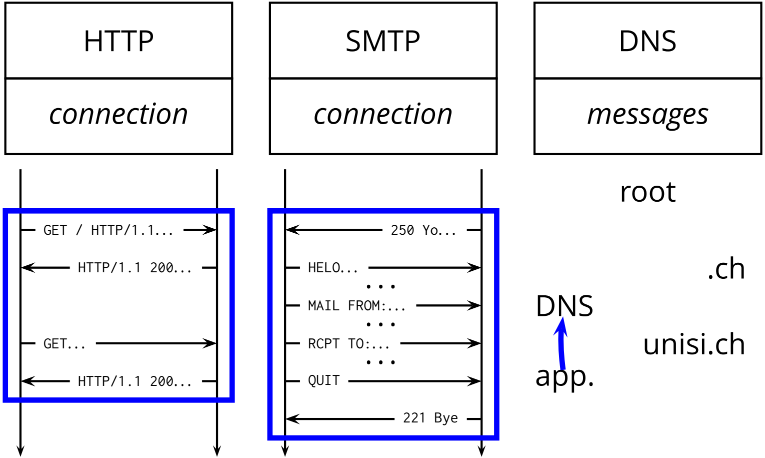
QUIT

221 Bye

# Application Protocols



# Application Protocols



HTTP

*connection*

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

SMTP

*connection*

250 Yo...

HELO...

...

MAIL FROM:...

...

RCPT TO:...

...

QUIT

221 Bye

DNS

*messages*

root

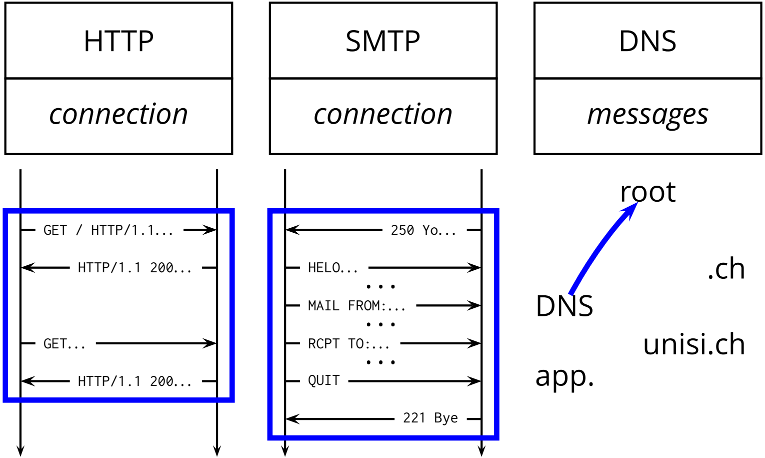
.ch

unisi.ch

DNS

app.

# Application Protocols



HTTP

*connection*

SMTP

*connection*

DNS

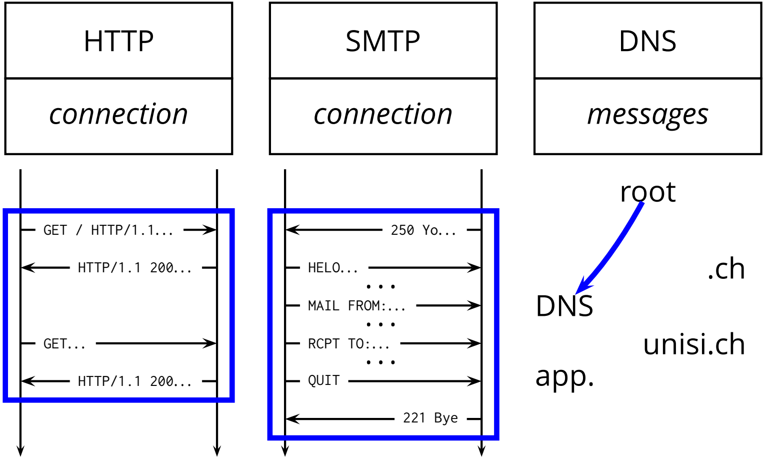
*messages*

GET / HTTP/1.1...
HTTP/1.1 200...
GET...
HTTP/1.1 200...

250 Yo...
HELO...
...
MAIL FROM:...
...
RCPT TO:...
...
QUIT
221 Bye

root
.ch
DNS
unisi.ch
app.

# Application Protocols



HTTP

*connection*

SMTP

*connection*

DNS

*messages*

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

250 Yo...

HELO...

...

MAIL FROM:...

...

RCPT TO:...

...

QUIT

221 Bye

root

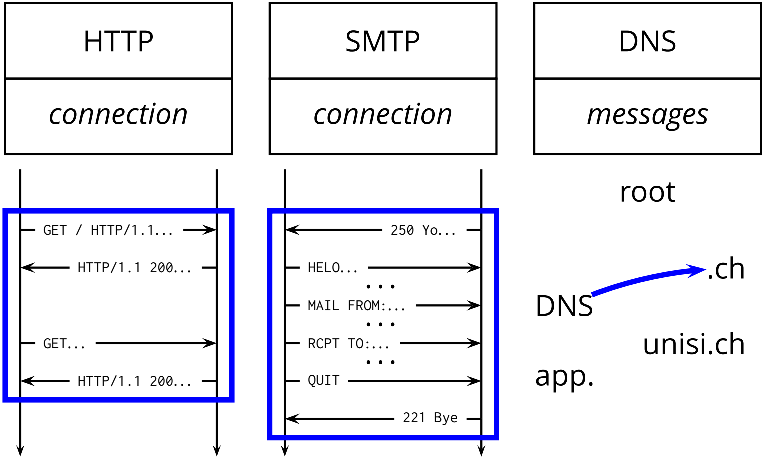
.ch

DNS

unisi.ch

app.

# Application Protocols



HTTP

*connection*

SMTP

*connection*

DNS

*messages*

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

250 Yo...

HELO...

...

MAIL FROM:...

...

RCPT TO:...

...

QUIT

221 Bye

root

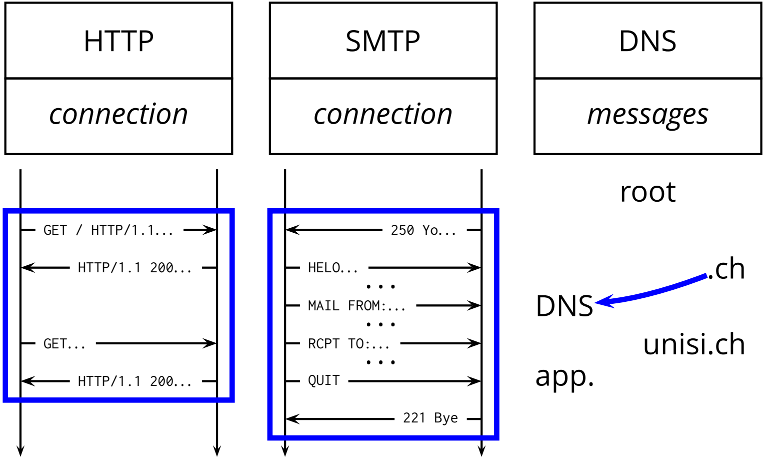
.ch

DNS

unisi.ch

app.

# Application Protocols



| HTTP | SMTP | DNS |
|---|---|---|
| *connection* | *connection* | *messages* |

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

250 Yo...

HELO...

...

MAIL FROM:...

...

RCPT TO:...

...

QUIT

221 Bye

root

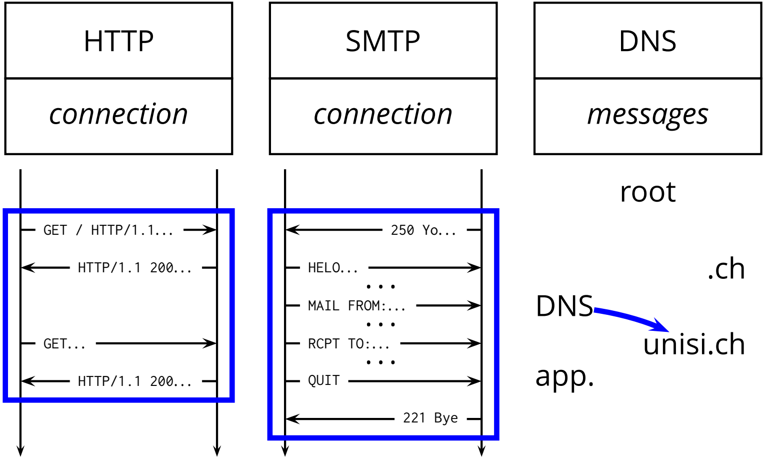
.ch

DNS

unisi.ch

app.

# Application Protocols



HTTP

*connection*

SMTP

*connection*

DNS

*messages*

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

250 Yo...

HELO...

...

MAIL FROM:...

...

RCPT TO:...

...

QUIT

221 Bye

root

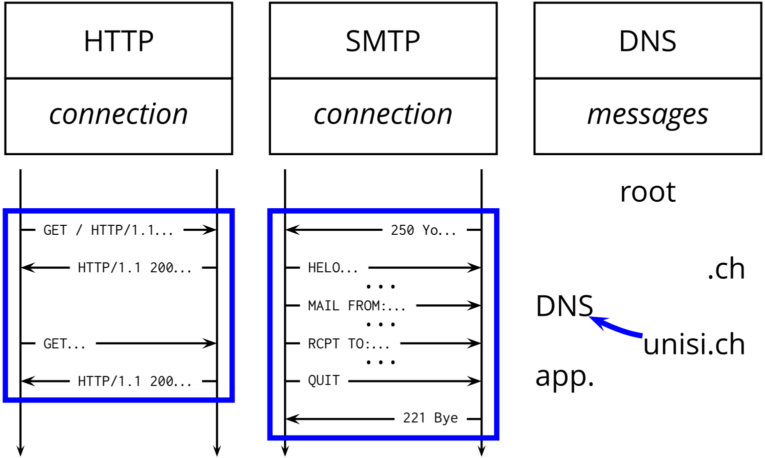
.ch

DNS

unisi.ch

app.

# Application Protocols



HTTP

*connection*

SMTP

*connection*

DNS

*messages*

`GET / HTTP/1.1...`

`HTTP/1.1 200...`

`GET...`

`HTTP/1.1 200...`

`250 Yo...`

`HELO...`

`...`

`MAIL FROM:...`

`...`

`RCPT TO:...`

`...`

`QUIT`

`221 Bye`

root

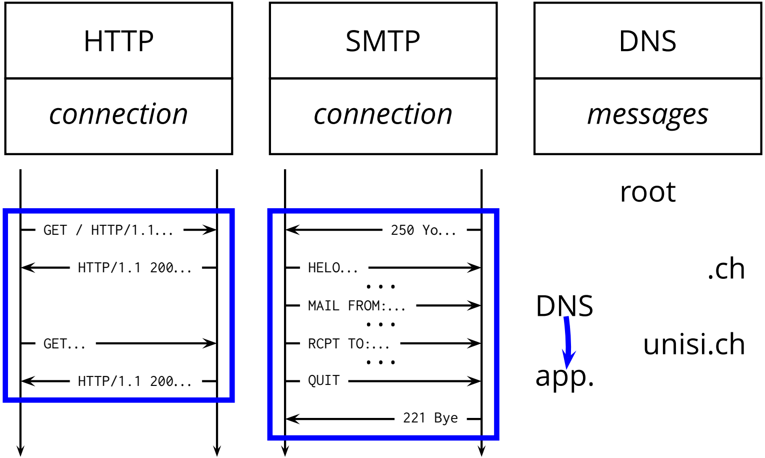
.ch

DNS

unisi.ch

app.

# Application Protocols



HTTP

*connection*

SMTP

*connection*

DNS

*messages*

GET / HTTP/1.1...

HTTP/1.1 200...

GET...

HTTP/1.1 200...

250 Yo...

HELO...

...

MAIL FROM:...

...

RCPT TO:...

...

QUIT

221 Bye

root

.ch

unisi.ch

DNS

app.

# Transport Layer in the Internet

# Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***
  - conntection-oriented (i.e., “connections”)

# Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***
  - conntection-oriented (i.e., “connections”)
- ***User Datagram Protocol (UDP)***
  - connectionless (i.e., “messages”)

# Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***
  - conntection-oriented (i.e., “connections”)
- ***User Datagram Protocol (UDP)***
  - connectionless (i.e., “messages”)
- Terminology
  - transport-layer packets are called ***segments***

# Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***
  - conntection-oriented (i.e., “connections”)
- ***User Datagram Protocol (UDP)***
  - connectionless (i.e., “messages”)
- Terminology
  - transport-layer packets are called ***segments***
- Basic assumptions on the underlying network layer

# Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***
  - conntection-oriented (i.e., “connections”)
- ***User Datagram Protocol (UDP)***
  - connectionless (i.e., “messages”)
- Terminology
  - transport-layer packets are called ***segments***
- Basic assumptions on the underlying network layer
  - every host has one unique *IP address*

# Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***
  - conntection-oriented (i.e., “connections”)

- ***User Datagram Protocol (UDP)***
  - connectionless (i.e., “messages”)

- Terminology
  - transport-layer packets are called ***segments***

- Basic assumptions on the underlying network layer
  - every host has one unique *IP address*
  - best-effort delivery service
    - no guarantees on the integrity of segments
    - no guarantees on the order in which segments are delivered

# Transport-Layer Value-Added Service

# Transport-Layer Value-Added Service

- ***Transport-layer multiplexing/demultiplexing***
    - i.e., connecting applications as opposed to hosts

# Transport-Layer Value-Added Service

- ***Transport-layer multiplexing/demultiplexing***
  - i.e., connecting applications as opposed to hosts

- ***Reliable data transfer***
  - i.e., integrity and possibly ordered delivery

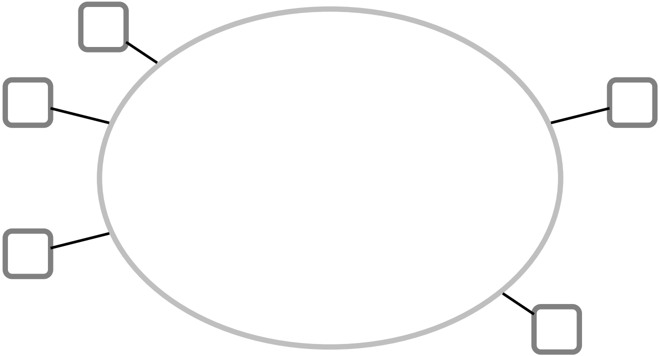
# Transport-Layer Value-Added Service

- ***Transport-layer multiplexing/demultiplexing***
  - i.e., connecting applications as opposed to hosts
- ***Reliable data transfer***
  - i.e., integrity and possibly ordered delivery
- ***Connections***
  - i.e., streams
  - can be seen as the same as ordered delivery

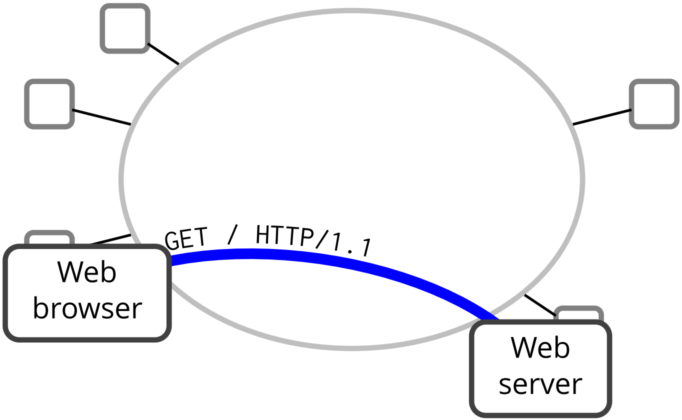
# Transport-Layer Value-Added Service

- ***Transport-layer multiplexing/demultiplexing***
  - i.e., connecting applications as opposed to hosts
- ***Reliable data transfer***
  - i.e., integrity and possibly ordered delivery
- ***Connections***
  - i.e., streams
  - can be seen as the same as ordered delivery
- ***Congestion control***
  - i.e., end-to-end traffic (admission) control so as to avoid destructive congestions within the network

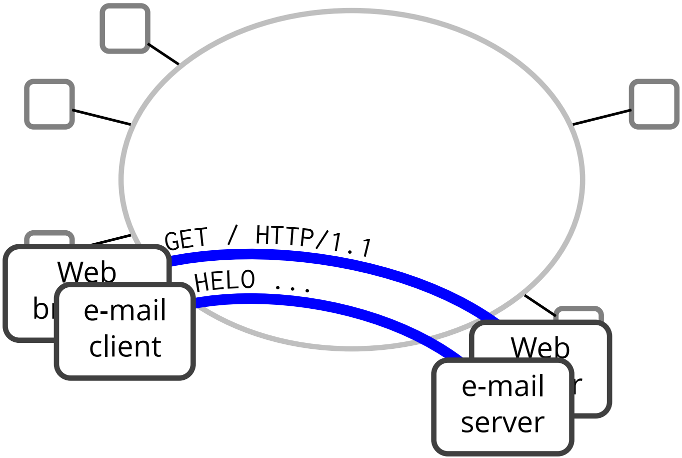
# Multiplexing/Demultiplexing



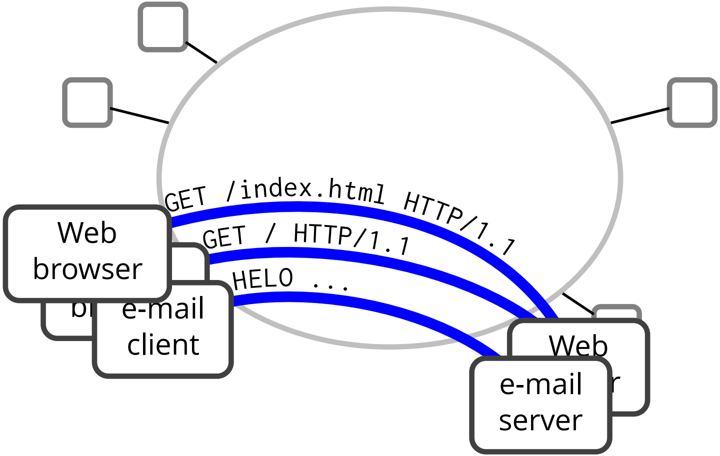
# Multiplexing/Demultiplexing



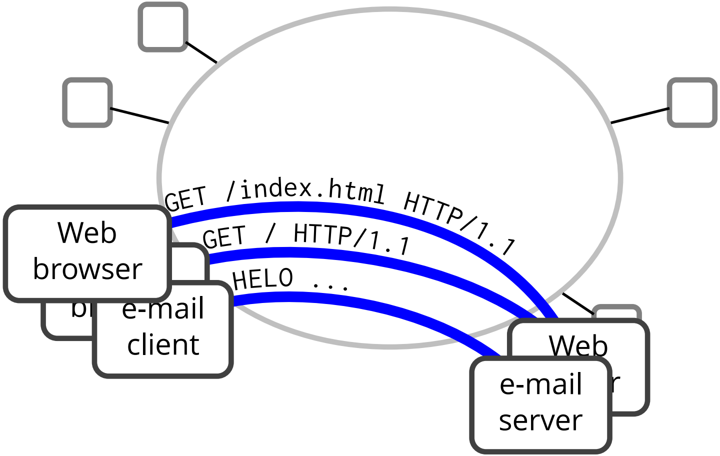
# Multiplexing/Demultiplexing



# Multiplexing/Demultiplexing

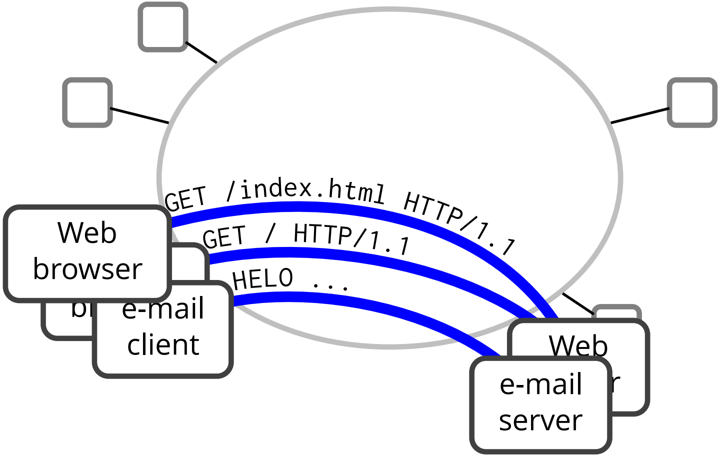


# Multiplexing/Demultiplexing



How do we distinguish all these “connections”?

# Multiplexing/Demultiplexing



How do we distinguish all these “connections”?
(in this case, connections between the same two hosts)

# Ports

# Ports

- Each application running on a host is identified (within that host) by a unique ***port number***
    - port numbers are simply cross-platform process identifiers

# Ports

- Each application running on a host is identified (within that host) by a unique ***port number***
    - port numbers are simply cross-platform process identifiers

- How do we identify a “connection”?

# Ports

- Each application running on a host is identified (within that host) by a unique ***port number***
    - port numbers are simply cross-platform process identifiers

- How do we identify a “connection”?
    - two pairs of *host* and *application* identifiers
    - i.e., two pairs *(IP-address, port)*

# Ports

- Each application running on a host is identified (within that host) by a unique ***port number***
  - port numbers are simply cross-platform process identifiers
- How do we identify a “connection”?
  - two pairs of *host* and *application* identifiers
  - i.e., two pairs *(IP-address, port)*
- How do we find out which application (host and port number) to connect to?

# Ports

- Each application running on a host is identified (within that host) by a unique ***port number***
  - port numbers are simply cross-platform process identifiers

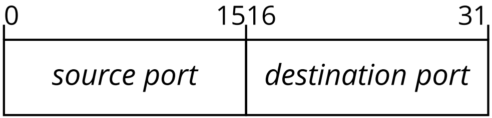
- How do we identify a “connection”?
  - two pairs of *host* and *application* identifiers
  - i.e., two pairs *(IP-address, port)*

- How do we find out which application (host and port number) to connect to?
  - outside the scope of the definition of the transport layer
  - but of course we can have “well-known” service numbers

# Ports

# Ports

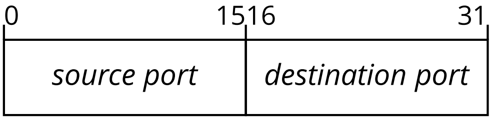
- The message format of both UDP and TCP starts with the source and destination port numbers



. . .

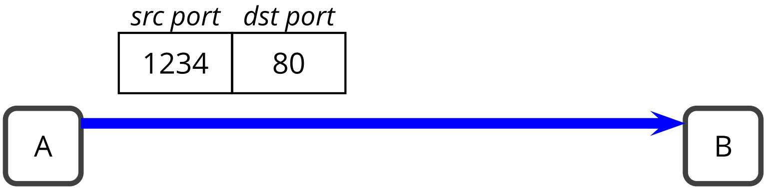
# Ports

- The message format of both UDP and TCP starts with the source and destination port numbers



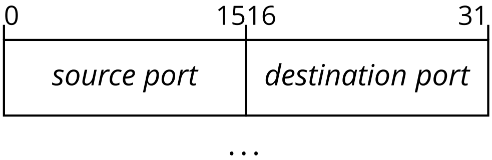
. . .

- E.g.,

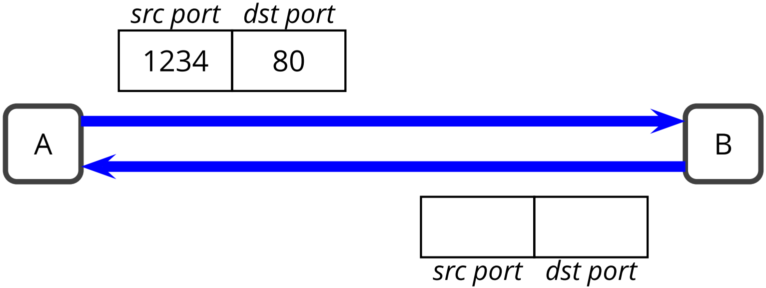


# Ports

- The message format of both UDP and TCP starts with the source and destination port numbers

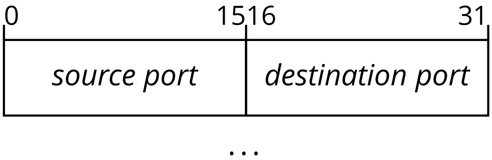


- E.g.,

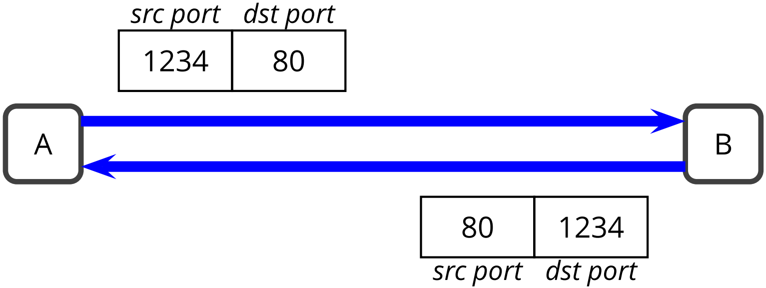


# Ports

- The message format of both UDP and TCP starts with the source and destination port numbers



- E.g.,



# Transmission Control Protocol

- The Internet’s primary transport protocol
    - defined in RFC 793, RFC 1122, RFC 1323, RFC 2018, and RFC 2581

# Transmission Control Protocol

- The Internet's primary transport protocol
  - defined in RFC 793, RFC 1122, RFC 1323, RFC 2018, and RFC 2581

- Connection-oriented service
  - endpoints "shake hands" to establish a connection
  - not a circuit-switched connection, nor a virtual circuit

# Transmission Control Protocol

- The Internet’s primary transport protocol
  - defined in RFC 793, RFC 1122, RFC 1323, RFC 2018, and RFC 2581

- Connection-oriented service
  - endpoints “shake hands” to establish a connection
  - not a circuit-switched connection, nor a virtual circuit

- Full-duplex service
  - both endpoints can both send and receive, at the same time

# Preliminary Definitions

# Preliminary Definitions

- ***TCP segment:*** envelope for TCP data
  - TCP data are sent within TCP segments
  - TCP segments are usually sent within an IP packet

# Preliminary Definitions

- ***TCP segment:*** envelope for TCP data
  - TCP data are sent within TCP segments
  - TCP segments are usually sent within an IP packet

- ***Maximum segment size (MSS):*** maximum amount of application data transmitted in a single segment
  - typically related to the MTU of the connection, to avoid network-level fragmentation (we'll talk about all of this later)

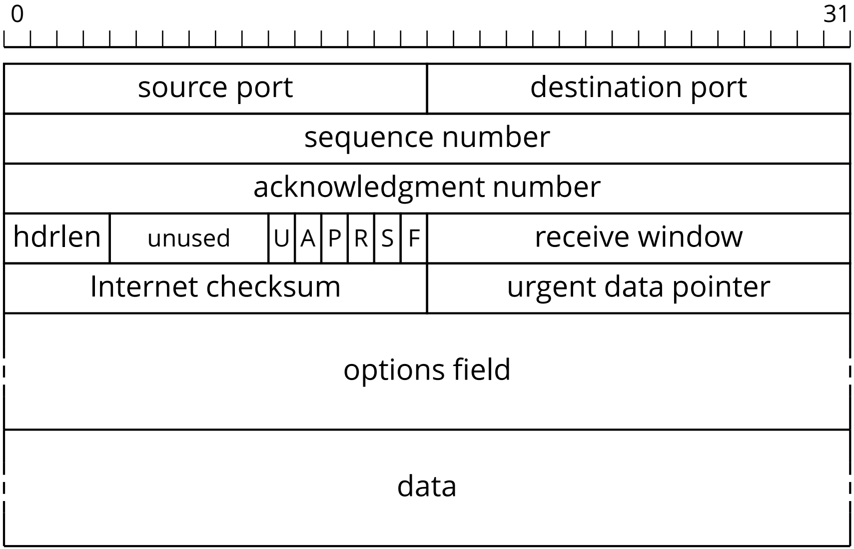
# Preliminary Definitions

- ***TCP segment:*** envelope for TCP data
  - TCP data are sent within TCP segments
  - TCP segments are usually sent within an IP packet

- ***Maximum segment size (MSS):*** maximum amount of application data transmitted in a single segment
  - typically related to the MTU of the connection, to avoid network-level fragmentation (we'll talk about all of this later)

- ***Maximum transmission unit (MTU):*** largest link-layer frame available to the sender host
  - *path MTU:* largest link-layer frame that can be sent on all links from the sender host to the receiver host

# TCP Segment Format



0

31

source port

destination port

sequence number

acknowledgment number

hdrlen

unused

U

A

P

R

S

F

receive window

Internet checksum

urgent data pointer

options field

data

# TCP Header Fields

# TCP Header Fields

- *Source and destination ports:* (16-bit each) application identifiers

# TCP Header Fields

- *Source and destination ports:* (16-bit each) application identifiers
- *Sequence number:* (32-bit) used to implement reliable data transfer
- *Acknowledgment number:* (32-bit) used to implement reliable data transfer

# TCP Header Fields

- *Source and destination ports:* (16-bit each) application identifiers
- *Sequence number:* (32-bit) used to implement reliable data transfer
- *Acknowledgment number:* (32-bit) used to implement reliable data transfer
- *Receive window:* (16-bit) size of the “window” on the receiver end

# TCP Header Fields

- *Source and destination ports:* (16-bit each) application identifiers
- *Sequence number:* (32-bit) used to implement reliable data transfer
- *Acknowledgment number:* (32-bit) used to implement reliable data transfer
- *Receive window:* (16-bit) size of the “window” on the receiver end
- *Header length:* (4-bit) size of the TCP header in 32-bit words

# TCP Header Fields

- *Source and destination ports:* (16-bit each) application identifiers
- *Sequence number:* (32-bit) used to implement reliable data transfer
- *Acknowledgment number:* (32-bit) used to implement reliable data transfer
- *Receive window:* (16-bit) size of the “window” on the receiver end
- *Header length:* (4-bit) size of the TCP header in 32-bit words
- *Optional and variable-length options field:* may be used to negotiate protocol parameters

# TCP Header Fields

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment
- *SYN flag:* (1-bit) used during connection setup and shutdown

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment
- *SYN flag:* (1-bit) used during connection setup and shutdown
- *RST flag:* (1-bit) used during connection setup and shutdown

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment
- *SYN flag:* (1-bit) used during connection setup and shutdown
- *RST flag:* (1-bit) used during connection setup and shutdown
- *FIN flag:* (1-bit) used during connection shutdown

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment
- *SYN flag:* (1-bit) used during connection setup and shutdown
- *RST flag:* (1-bit) used during connection setup and shutdown
- *FIN flag:* (1-bit) used during connection shutdown
- *PSH flag:* (1-bit) “push” flag, used to solicit the receiver to pass the data to the application immediately

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment
- *SYN flag:* (1-bit) used during connection setup and shutdown
- *RST flag:* (1-bit) used during connection setup and shutdown
- *FIN flag:* (1-bit) used during connection shutdown
- *PSH flag:* (1-bit) “push” flag, used to solicit the receiver to pass the data to the application immediately
- *URG flag:* (1-bit) “urgent” flag, used to inform the receiver that the sender has marked some data as “urgent”. The location of this urgent data is marked by the *urgent data pointer* field

# TCP Header Fields

- *ACK flag:* (1-bit) signals that the value contained in the *acknowledgment number* represents a valid acknowledgment
- *SYN flag:* (1-bit) used during connection setup and shutdown
- *RST flag:* (1-bit) used during connection setup and shutdown
- *FIN flag:* (1-bit) used during connection shutdown
- *PSH flag:* (1-bit) "push" flag, used to solicit the receiver to pass the data to the application immediately
- *URG flag:* (1-bit) "urgent" flag, used to inform the receiver that the sender has marked some data as "urgent". The location of this urgent data is marked by the *urgent data pointer* field
- *Checksum:* (16-bit) used to detect transmission errors

# Sequence Numbers

# Sequence Numbers

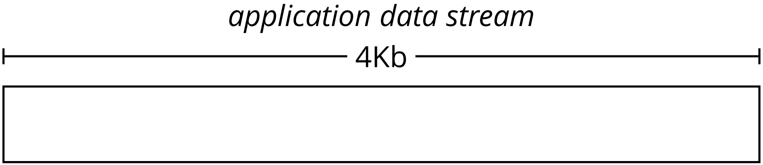
- Sequence numbers are associated with *bytes* in the data stream
    - not with segments, as we have used them before

# Sequence Numbers

- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before
- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***

# Sequence Numbers

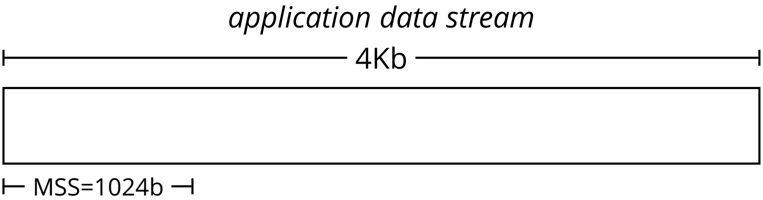
- Sequence numbers are associated with *bytes* in the data stream
    - not with segments, as we have used them before
- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



# Sequence Numbers

- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before

- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



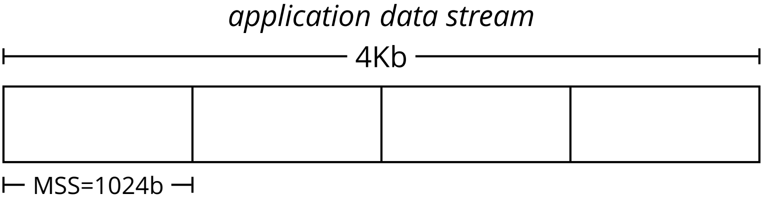
*application data stream*

4Kb

MSS=1024b

# Sequence Numbers

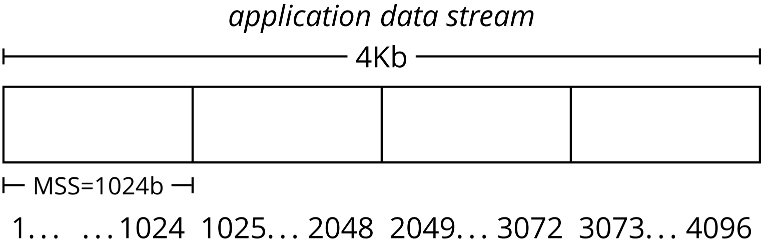
- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before
- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



# Sequence Numbers

- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before

- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



*application data stream*

4Kb

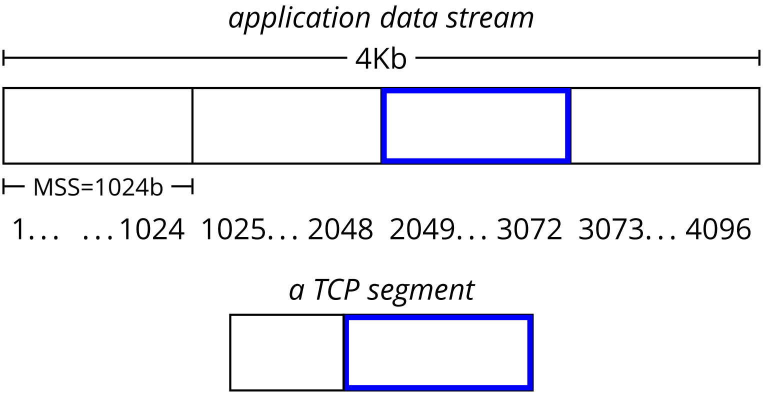
MSS=1024b

1. . .   . . .1024   1025. . . 2048   2049. . . 3072   3073. . . 4096

# Sequence Numbers

- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before

- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



*application data stream*

4Kb

MSS=1024b

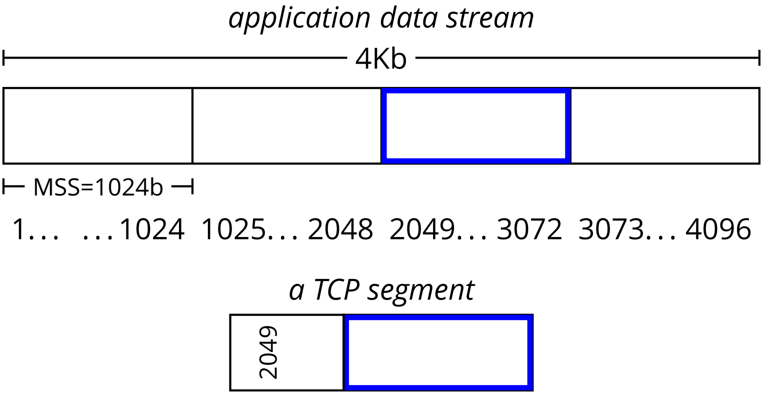
1...  ...1024  1025...2048  2049...3072  3073...4096

*a TCP segment*

# Sequence Numbers

- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before

- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



*application data stream*

4Kb

MSS=1024b

1. . .  . . .1024  1025. . . 2048  2049. . . 3072  3073. . . 4096

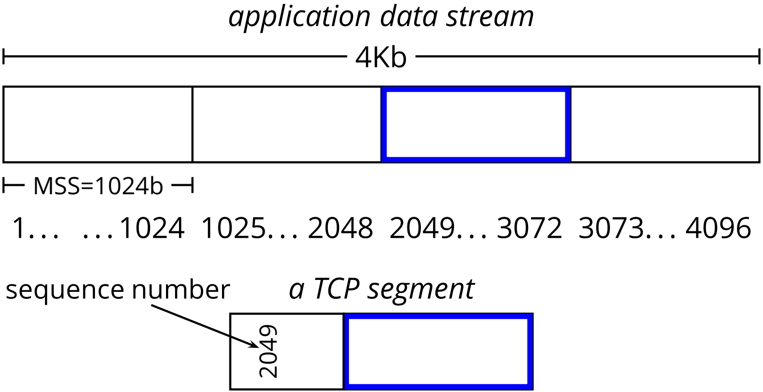
*a TCP segment*

2049

# Sequence Numbers

- Sequence numbers are associated with *bytes* in the data stream
  - not with segments, as we have used them before

- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***



*application data stream*

4Kb

MSS=1024b

1. . . . . . 1024 1025. . . 2048 2049. . . 3072 3073. . . 4096

sequence number *a TCP segment*

2049

# Acknowledgment Numbers

# Acknowledgment Numbers

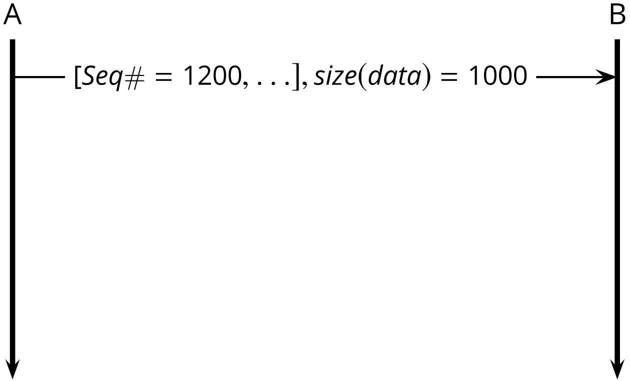
- An ***acknowledgment number*** represents the ***first sequence number not yet seen by the receiver***
  - TCP acknowledgments are *cumulative*

# Acknowledgment Numbers

- An ***acknowledgment number*** represents the ***first sequence number not yet seen by the receiver***
  - TCP acknowledgments are *cumulative*

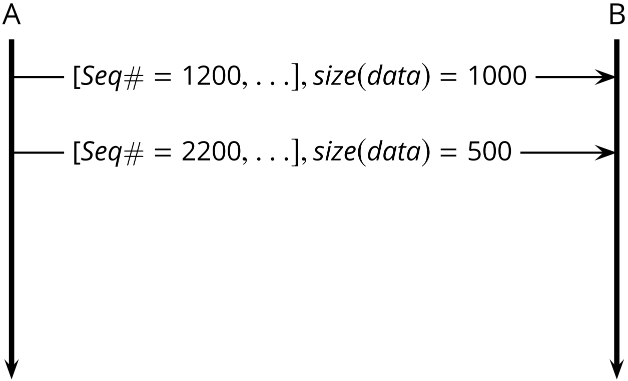
# Acknowledgment Numbers

- An ***acknowledgment number*** represents the ***first sequence number not yet seen by the receiver***
  - TCP acknowledgments are *cumulative*



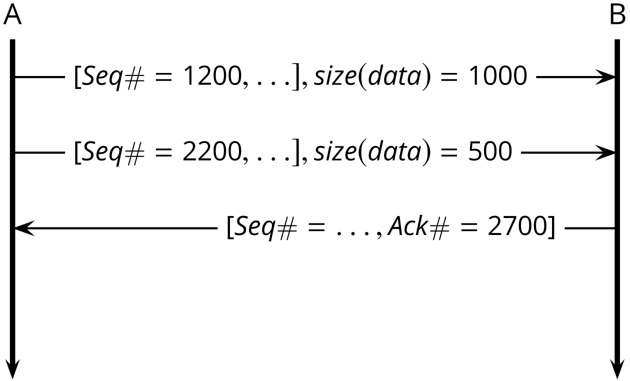
# Acknowledgment Numbers

- An ***acknowledgment number*** represents the ***first sequence number not yet seen by the receiver***
  - TCP acknowledgments are *cumulative*



# Acknowledgment Numbers

- An ***acknowledgment number*** represents the ***first sequence number not yet seen by the receiver***
  - TCP acknowledgments are *cumulative*



# Sequence Numbers and ACK Numbers

# Sequence Numbers and ACK Numbers

- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

# Sequence Numbers and ACK Numbers

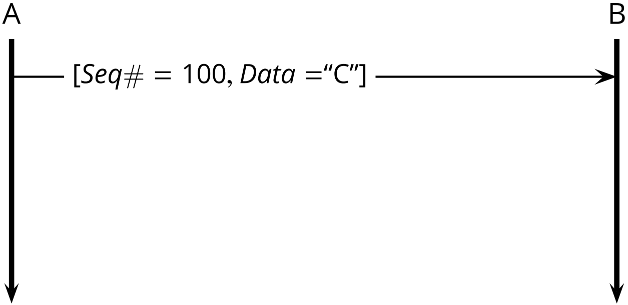
- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

E.g., consider a simple “Echo” application:

# Sequence Numbers and ACK Numbers

- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

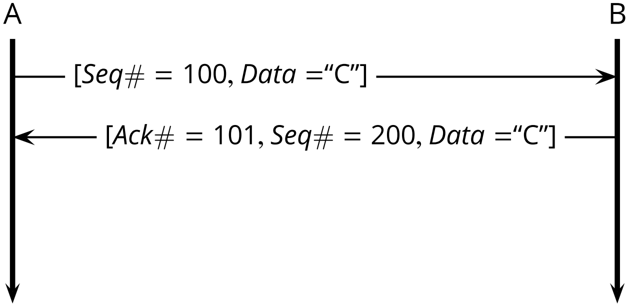
E.g., consider a simple "Echo" application:



# Sequence Numbers and ACK Numbers

- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

E.g., consider a simple “Echo” application:



A

B

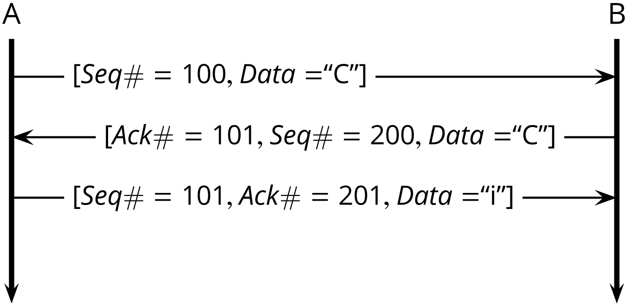
[$Seq\# = 100$, $Data$ =“C”]

[$Ack\# = 101$, $Seq\# = 200$, $Data$ =“C”]

# Sequence Numbers and ACK Numbers

- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

E.g., consider a simple “Echo” application:



A

B

$[Seq\# = 100, Data = \text{“C”}]$

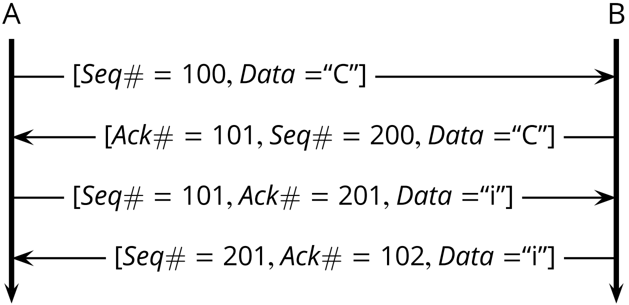
$[Ack\# = 101, Seq\# = 200, Data = \text{“C”}]$

$[Seq\# = 101, Ack\# = 201, Data = \text{“i”}]$

# Sequence Numbers and ACK Numbers

- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

E.g., consider a simple “Echo” application:



A

B

$[Seq\# = 100, Data = \text{“C”}]$

$[Ack\# = 101, Seq\# = 200, Data = \text{“C”}]$

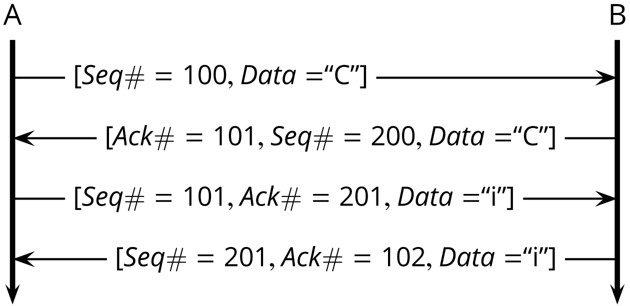
$[Seq\# = 101, Ack\# = 201, Data = \text{“i”}]$

$[Seq\# = 201, Ack\# = 102, Data = \text{“i”}]$

# Sequence Numbers and ACK Numbers

- Notice that a TCP connection is a *full-duplex* link
  - therefore, there are ***two streams***
  - two different sequence numbers

  E.g., consider a simple “Echo” application:



$[Seq\# = 100, Data = \text{“C”}]$

$[Ack\# = 101, Seq\# = 200, Data = \text{“C”}]$

$[Seq\# = 101, Ack\# = 201, Data = \text{“i”}]$

$[Seq\# = 201, Ack\# = 102, Data = \text{“i”}]$

- Acknowledgments are “piggybacked” on data segments

# Reliability and Timeout

- TCP provides reliable data transfer using a *timer* to detect lost segments
  - timeout without an ACK ⟶ lost packet ⟶ retransmission

# Reliability and Timeout

- TCP provides reliable data transfer using a *timer* to detect lost segments
  - timeout without an ACK → lost packet → retransmission

- How long to wait for acknowledgments?

# Reliability and Timeout

- TCP provides reliable data transfer using a *timer* to detect lost segments
    - timeout without an ACK $\longrightarrow$ lost packet $\longrightarrow$ retransmission

- How long to wait for acknowledgments?

- Retransmission timeouts should be larger than the round-trip time $RTT = 2L$
    - as close as possible to the $RTT$

# Reliability and Timeout

- TCP provides reliable data transfer using a *timer* to detect lost segments
  - timeout without an ACK $\longrightarrow$ lost packet $\longrightarrow$ retransmission

- How long to wait for acknowledgments?

- Retransmission timeouts should be larger than the round-trip time $RTT = 2L$
  - as close as possible to the $RTT$

- TCP controls its timeout by continuously *estimating the current RTT*

# Round-Trip Time Estimation

# Round-Trip Time Estimation

- RTT is measured using ACKs
    - only for packets transmitted once

- Given a single sample $S$ at any given time

- *Exponential weighted moving average (EWMA)*

$$\overline{RTT} = (1 - \alpha)\overline{RTT}' + \alpha S$$

# Round-Trip Time Estimation

- RTT is measured using ACKs
  - only for packets transmitted once

- Given a single sample *S* at any given time

- *Exponential weighted moving average (EWMA)*

$$\overline{RTT} = (1 - \alpha)\overline{RTT}' + \alpha S$$

  - RFC 2988 recommends $\alpha = 0.125$

# Round-Trip Time Estimation

- RTT is measured using ACKs
    - only for packets transmitted once

- Given a single sample $S$ at any given time

- *Exponential weighted moving average (EWMA)*

$$\overline{RTT} = (1-\alpha)\overline{RTT}' + \alpha S$$

    - RFC 2988 recommends $\alpha = 0.125$

- TCP also measures the *variability of RTT*

$$\overline{DevRTT} = (1-\beta)\overline{DevRTT}' + \beta|\overline{RTT}' - S|$$

# Round-Trip Time Estimation

- RTT is measured using ACKs
  - only for packets transmitted once

- Given a single sample $S$ at any given time

- *Exponential weighted moving average (EWMA)*

$$\overline{RTT} = (1 - \alpha)\overline{RTT}' + \alpha S$$

  - RFC 2988 recommends $\alpha = 0.125$

- TCP also measures the *variability of RTT*

$$\overline{DevRTT} = (1 - \beta)\overline{DevRTT}' + \beta|\overline{RTT}' - S|$$

  - RFC 2988 recommends $\beta = 0.25$

# Timeout Value

# Timeout Value

- The timeout interval *T* must be larger than the RTT
  - so as to avoid unnecessary retransmission

- However, *T* should not be too far from RTT
  - so as to detect (and retransmit) lost segments as quickly as possible

# Timeout Value

- The timeout interval $T$ must be larger than the RTT
  - so as to avoid unnecessary retransmission
- However, $T$ should not be too far from RTT
  - so as to detect (and retransmit) lost segments as quickly as possible
- TCP sets its timeouts using the estimated RTT ($\overline{RTT}$) and the variability estimate $\overline{DevRTT}$:

$$T = \overline{RTT} + 4\overline{DevRTT}$$

# Reliable Data Transfer (Sender)

A simplified TCP sender

- r_send(*data*)

```
if (timer not running)
    start_timer()
u_send([data,next_seq_num])
$next\_seq\_num \leftarrow next\_seq\_num + length(data)$
```

# Reliable Data Transfer (Sender)

A simplified TCP sender

- r_send(*data*)

```
if (timer not running)
    start_timer()
u_send([data,next_seq_num])
$next\_seq\_num \leftarrow next\_seq\_num + length(data)$
```

- timeout

```
u_send(pending segment with smallest sequence number)
start_timer()
```

# Reliable Data Transfer (Sender)

A simplified TCP sender

- r_send(*data*)

```
if (timer not running)
    start_timer()
u_send([data,next_seq_num])
$next\_seq\_num \leftarrow next\_seq\_num + length(data)$
```

- timeout

```
u_send(pending segment with smallest sequence number)
start_timer()
```

- u_recv([ACK,*y*])

```
if ($y > base$)
    $base \leftarrow y$
    if (there are pending segments)
        start_timer()
else ...
```

# Acknowledgment Generation (Receiver)

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
    - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
    - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK

- Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK (see above)

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
  - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK

- Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK (see above)
  - *Cumulative ACK:* immediately send cumulative ACK (for both segments)

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
  - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK

- Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK (see above)
  - *Cumulative ACK:* immediately send cumulative ACK (for both segments)

- Arrival of out of order segment with higher-than-expected sequence number (gap detected)

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
    - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK
- Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK (see above)
    - *Cumulative ACK:* immediately send cumulative ACK (for both segments)
- Arrival of out of order segment with higher-than-expected sequence number (gap detected)
    - *Duplicate ACK:* immediately send duplicate ACK

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
  - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK
- Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK (see above)
  - *Cumulative ACK:* immediately send cumulative ACK (for both segments)
- Arrival of out of order segment with higher-than-expected sequence number (gap detected)
  - *Duplicate ACK:* immediately send duplicate ACK
- Arrival of segment that (partially or completely) fills a gap in the received data

# Acknowledgment Generation (Receiver)

- Arrival of in-order segment with expected sequence number; all data up to expected sequence number already acknowledged
  - *Delayed ACK:* wait 500ms for another in-order segment; If that does not arrive, send ACK
- Arrival of in-order segment with expected sequence number. One other in-order segment waiting for ACK (see above)
  - *Cumulative ACK:* immediately send cumulative ACK (for both segments)
- Arrival of out of order segment with higher-than-expected sequence number (gap detected)
  - *Duplicate ACK:* immediately send duplicate ACK
- Arrival of segment that (partially or completely) fills a gap in the received data
  - *Immediate ACK:* immediately send ACK if the packet start at the lower end of the gap

# Reaction to ACKs (Sender)

# Reaction to ACKs (Sender)

- u_recv([ACK,*y*])

```
if ($y > base$)
    $base \leftarrow y$
    if (there are pending segments)
        start_timer()
```

## Reaction to ACKs (Sender)

- u_recv([ACK,*y*])

```
if ($y > base$)
    $base \leftarrow y$
    if (there are pending segments)
        start_timer()
else
    $ack\_counter[y] \leftarrow ack\_counter[y] + 1$
    if ($ack\_counter[y] = 3$)
        u_send(segment with sequence number $y$)
```

# Connection Setup

# Connection Setup

Three-way handshake

# Connection Setup

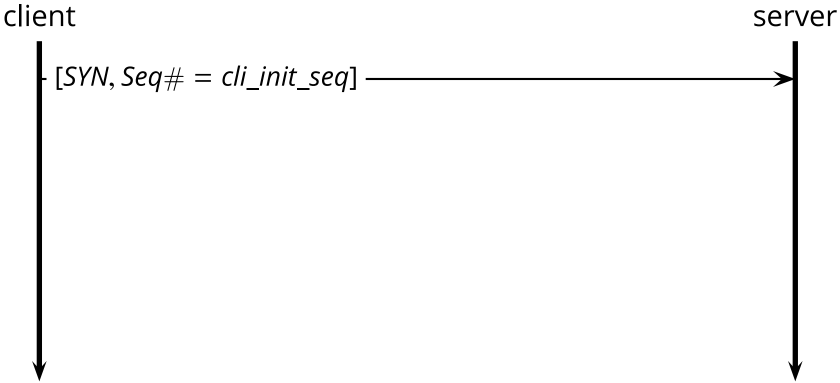
Three-way handshake

client

server

# Connection Setup

Three-way handshake



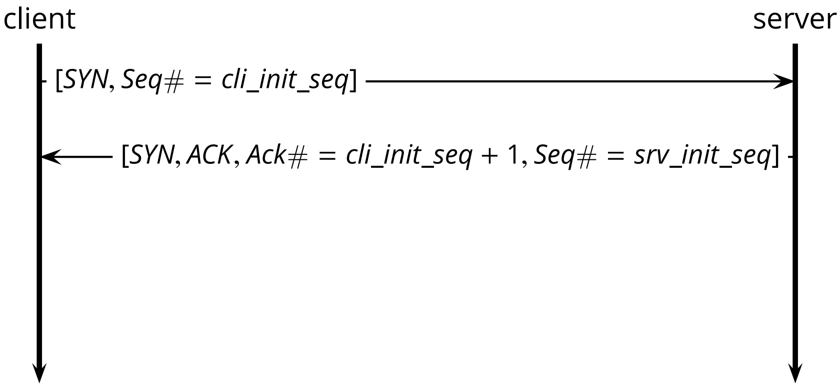
client

server

[$SYN, Seq\# = cli\_init\_seq$]

# Connection Setup

Three-way handshake



client

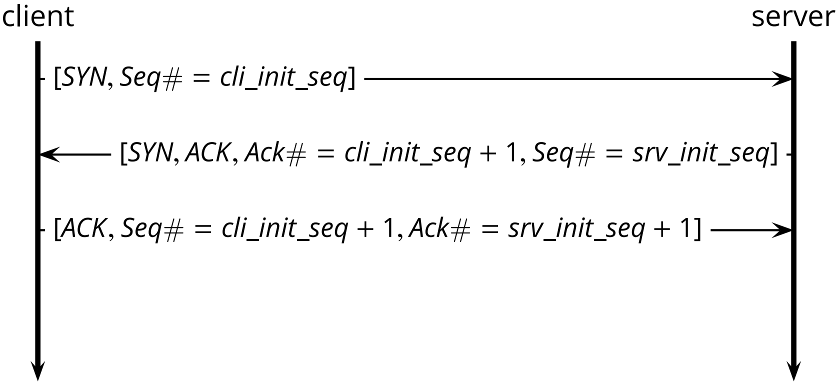
server

$[SYN, Seq\# = cli\_init\_seq]$

$[SYN, ACK, Ack\# = cli\_init\_seq + 1, Seq\# = srv\_init\_seq]$

# Connection Setup

Three-way handshake



client

server

$[SYN, Seq\# = cli\_init\_seq]$

$[SYN, ACK, Ack\# = cli\_init\_seq + 1, Seq\# = srv\_init\_seq]$

$[ACK, Seq\# = cli\_init\_seq + 1, Ack\# = srv\_init\_seq + 1]$

# Connection Shutdown

“This is it.”
“Okay, Bye now.”
“Bye.”

# Connection Shutdown

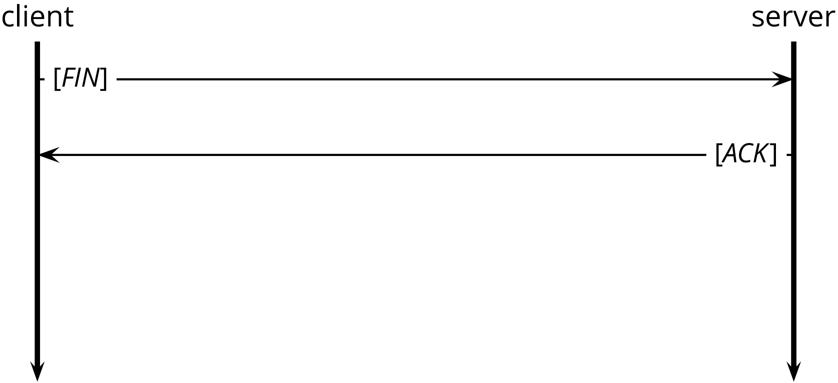
"This is it."
"Okay, Bye now."
"Bye."

# Connection Shutdown

“This is it.”
“Okay, Bye now.”
“Bye.”

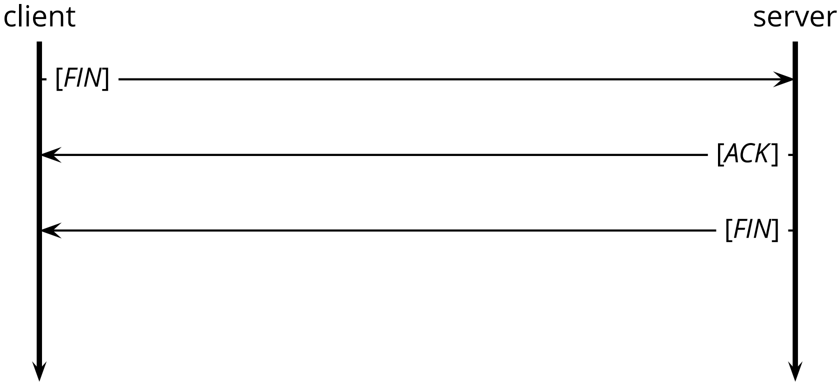
# Connection Shutdown

"This is it."
"Okay, Bye now."
"Bye."



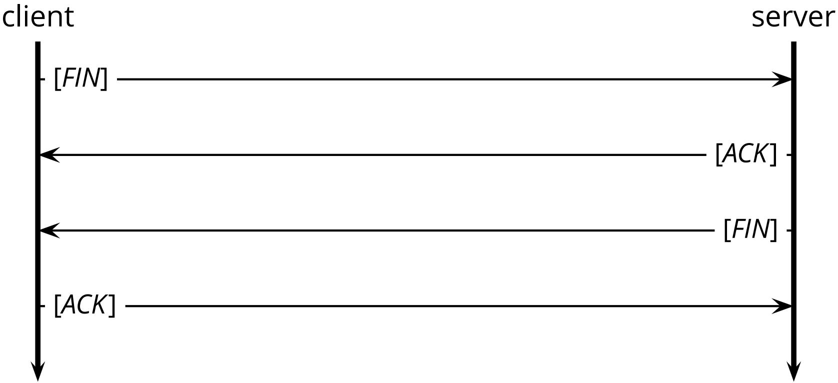
# Connection Shutdown

“This is it.”
“Okay, Bye now.”
“Bye.”



# Connection Shutdown

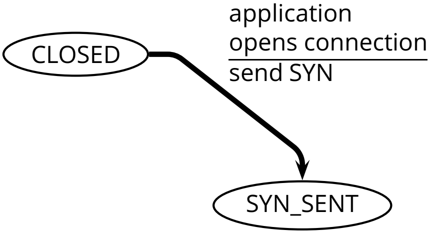
“This is it.”
“Okay, Bye now.”
“Bye.”



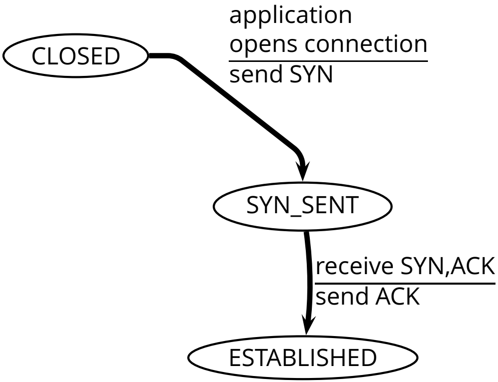
# The TCP State Machine (Client)

CLOSED

# The TCP State Machine (Client)



# The TCP State Machine (Client)



CLOSED

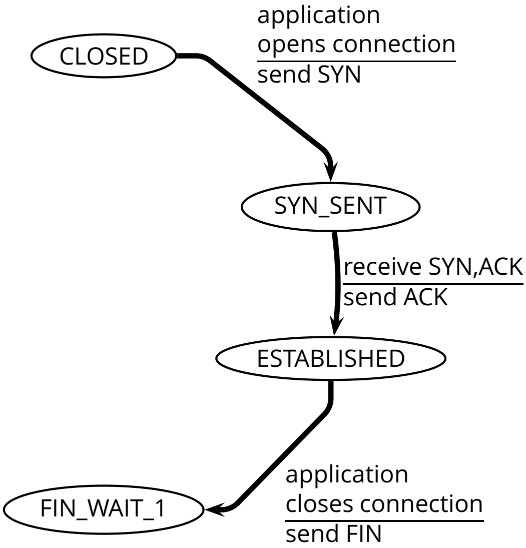
application
opens connection
send SYN

SYN_SENT

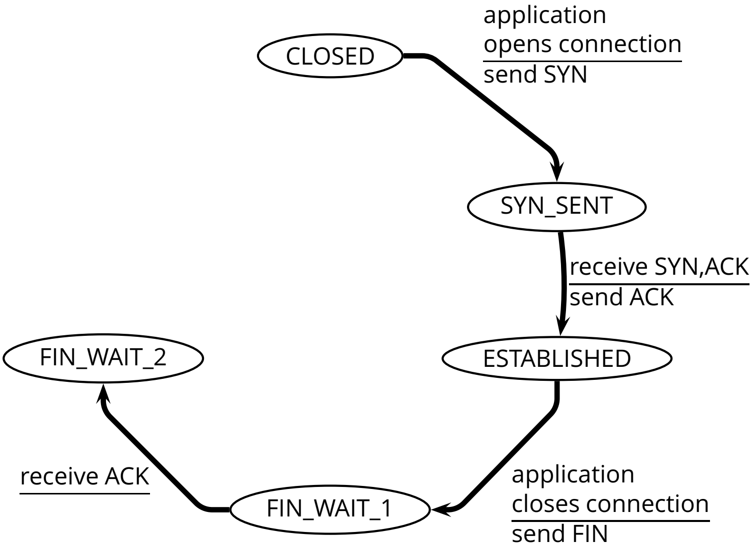
receive SYN,ACK
send ACK

ESTABLISHED

# The TCP State Machine (Client)



# The TCP State Machine (Client)



CLOSED

application
opens connection
send SYN

SYN_SENT

receive SYN,ACK
send ACK

ESTABLISHED

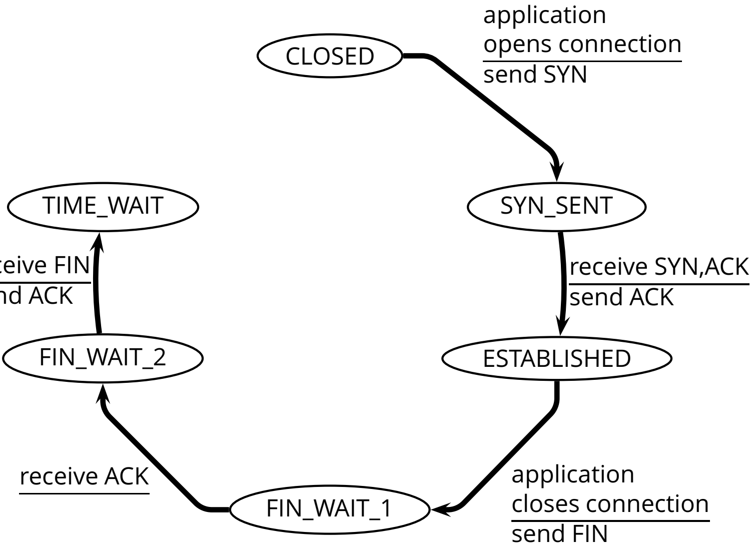
application
closes connection
send FIN

FIN_WAIT_1

receive ACK

FIN_WAIT_2

# The TCP State Machine (Client)



CLOSED

application
opens connection
send SYN

SYN_SENT

receive SYN,ACK
send ACK

ESTABLISHED

application
closes connection
send FIN

FIN_WAIT_1

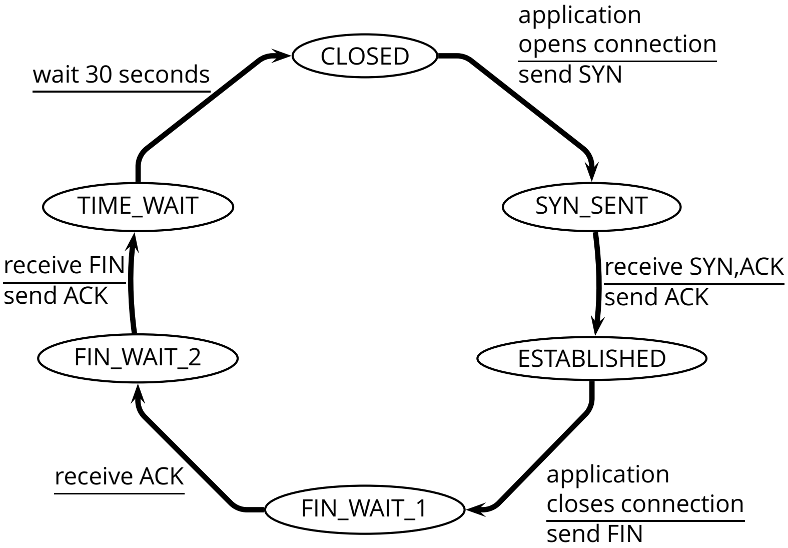
receive ACK

FIN_WAIT_2

receive FIN
send ACK

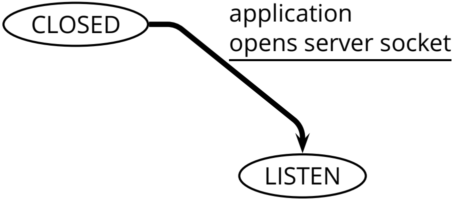
TIME_WAIT

# The TCP State Machine (Client)

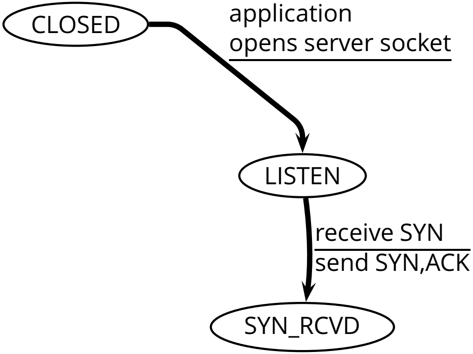


# The TCP State Machine (Server)

# The TCP State Machine (Server)



# The TCP State Machine (Server)



CLOSED

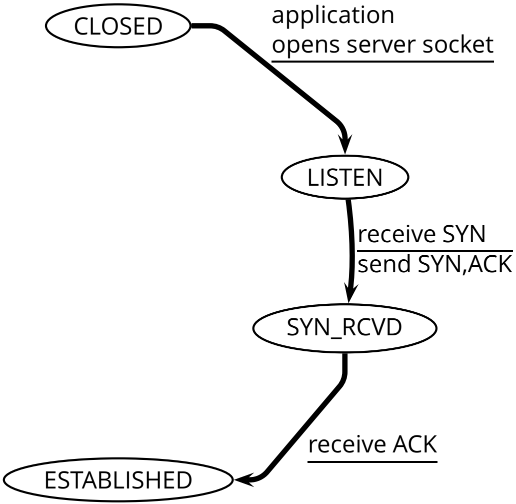
application
opens server socket

LISTEN

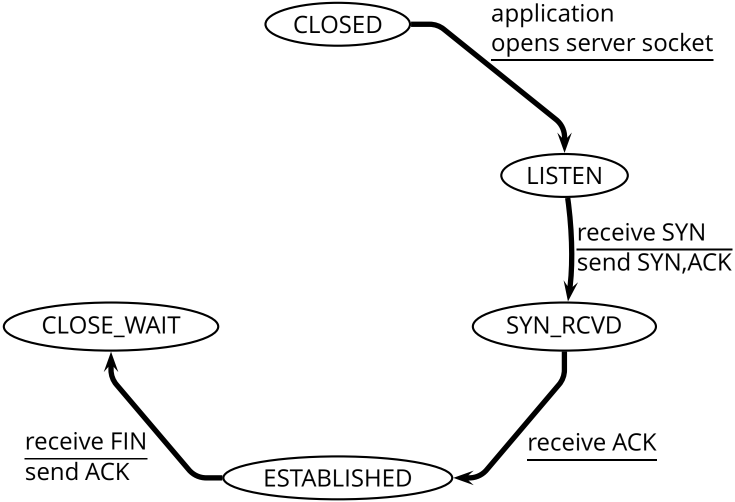
receive SYN
send SYN,ACK

SYN_RCVD

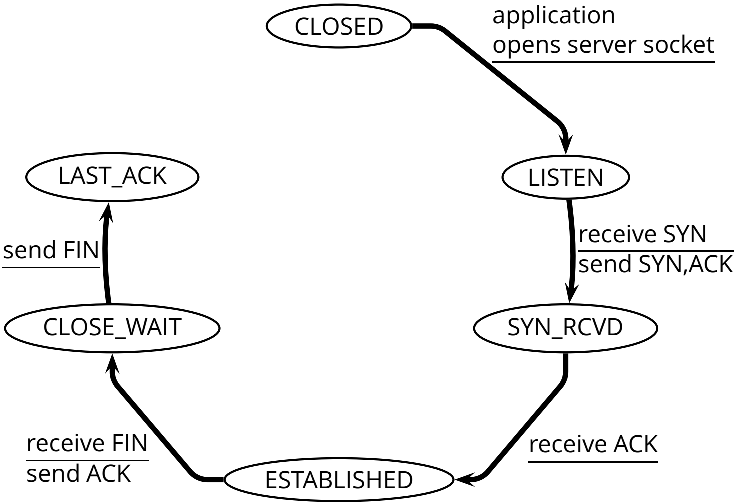
# The TCP State Machine (Server)



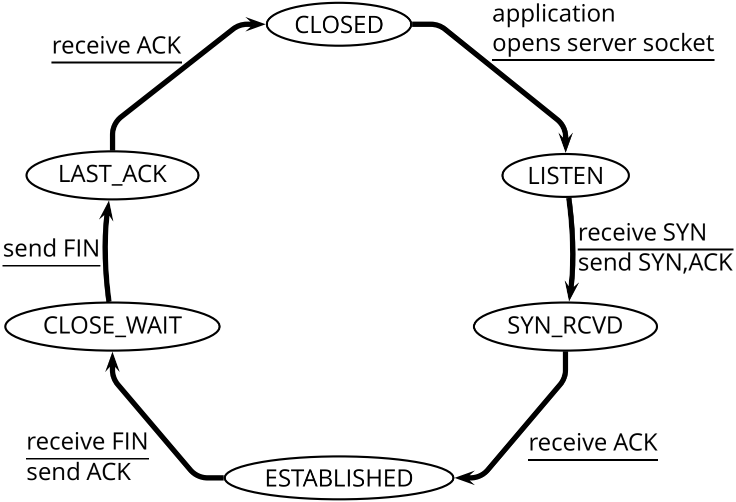
# The TCP State Machine (Server)



# The TCP State Machine (Server)



# The TCP State Machine (Server)



CLOSED

application
opens server socket

LISTEN

receive SYN
send SYN,ACK

SYN_RCVD

receive ACK

ESTABLISHED

receive FIN
send ACK

CLOSE_WAIT

send FIN

LAST_ACK

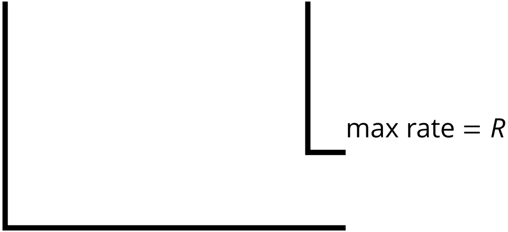
receive ACK

# Understanding Congestion

- A router behaves a lot like a kitchen sink

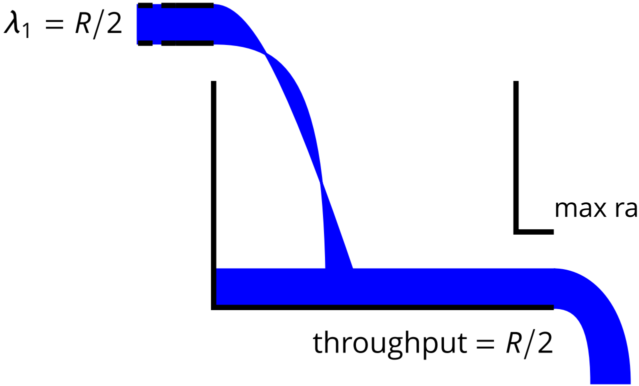
# Understanding Congestion

- A router behaves a lot like a kitchen sink



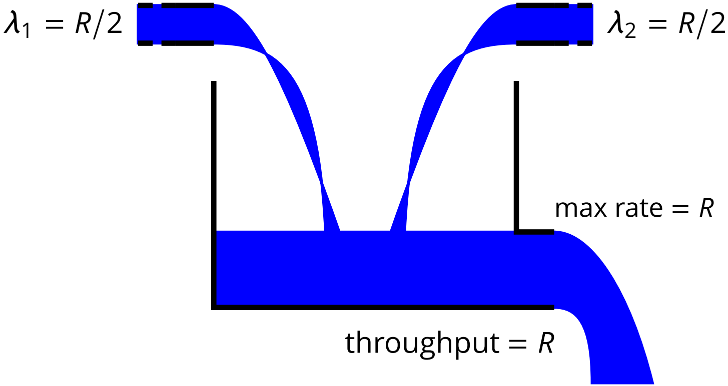
# Understanding Congestion

- A router behaves a lot like a kitchen sink



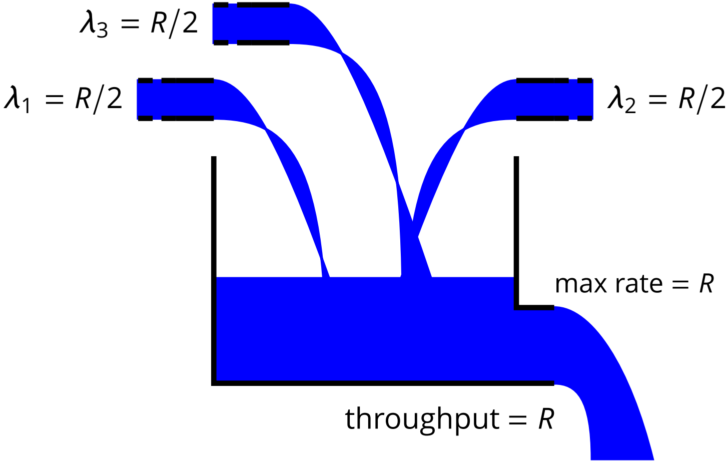
# Understanding Congestion

- A router behaves a lot like a kitchen sink



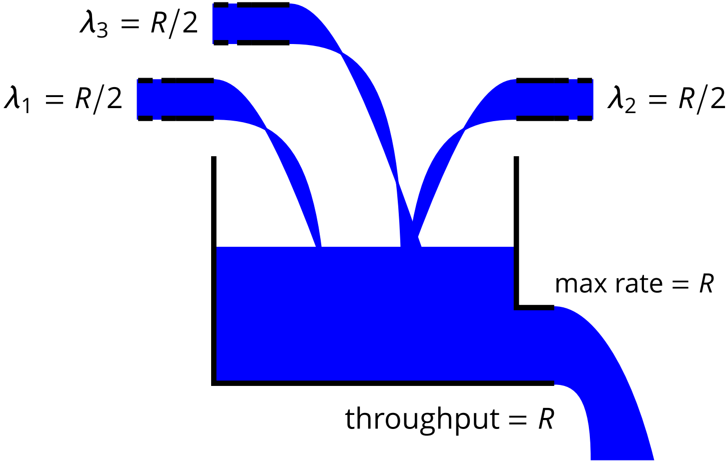
# Understanding Congestion

- A router behaves a lot like a kitchen sink



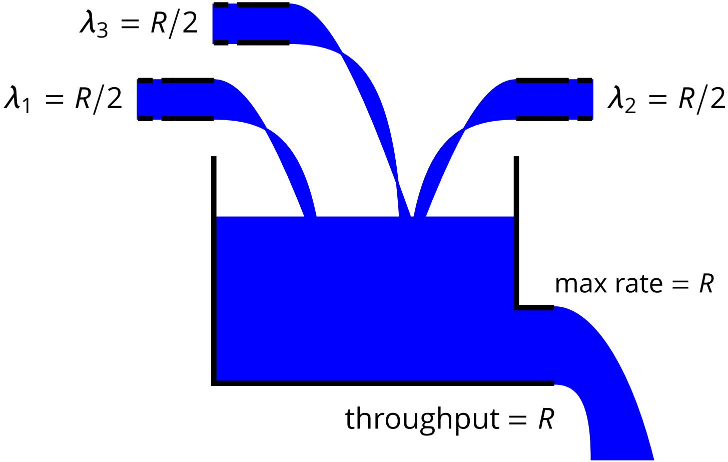
# Understanding Congestion

- A router behaves a lot like a kitchen sink



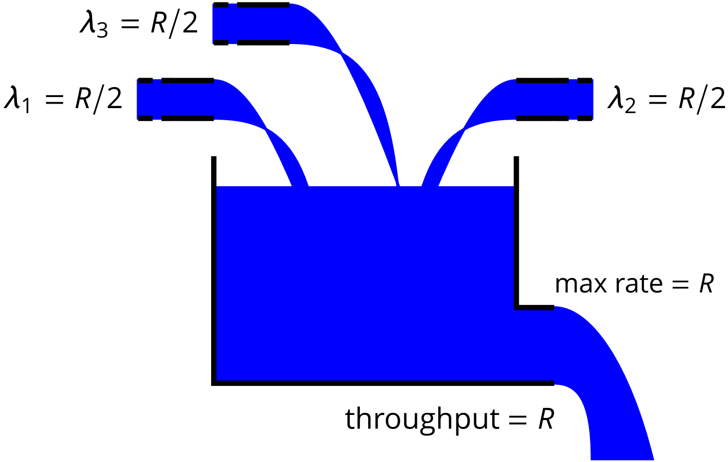
# Understanding Congestion

- A router behaves a lot like a kitchen sink



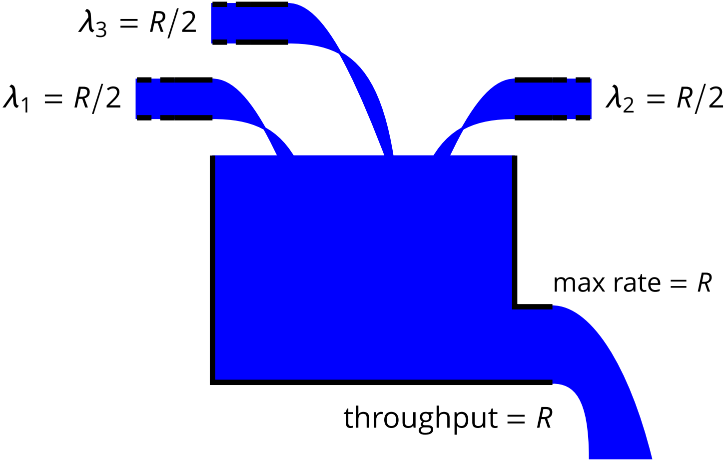
# Understanding Congestion

- A router behaves a lot like a kitchen sink



# Understanding Congestion

- A router behaves a lot like a kitchen sink



# Queuing Delay

# Queuing Delay

- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

$$L = d_{TX} + d_{CPU} + d_q \quad \text{where } d_q = |q|/R$$

# Queuing Delay

- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

$$L = d_{TX} + d_{CPU} + d_q \quad \text{where } d_q = |q|/R$$

- ***Ideal case:*** constant input data rate

$$\lambda_{in} < R$$

In this case the $d_q = 0$, because $|q| = 0$ (ideal input flow)

# Queuing Delay

- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

$$L = d_{TX} + d_{CPU} + d_q \quad \text{where } d_q = |q|/R$$

- ***Ideal case:*** constant input data rate

$$\lambda_{in} < R$$

In this case the $d_q = 0$, because $|q| = 0$ (ideal input flow)

- ***Extreme case:*** constant input data rate

$$\lambda_{in} > R$$

In this case $|q| = (\lambda_{in} - R)t$ and therefore

$$d_q = \frac{\lambda_{in} - R}{R} t$$

# Queuing Delay

# Queuing Delay

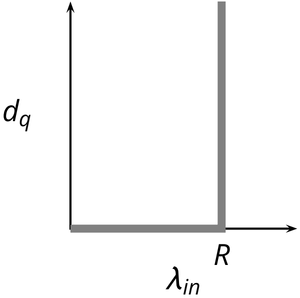
- Steady-state queuing delay

$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in} - R}{R} t & \lambda_{in} > R \end{cases}$$

# Queuing Delay

- Steady-state queuing delay

$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in} - R}{R} t & \lambda_{in} > R \end{cases}$$

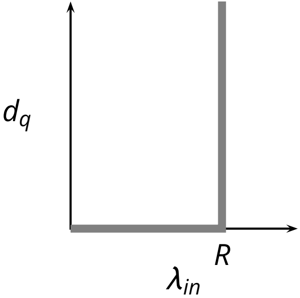


ideal input flow
$\lambda_{in}$ constant

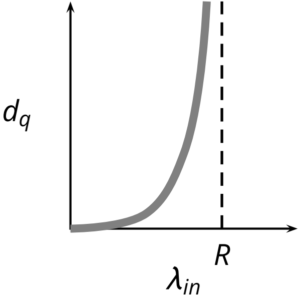
# Queuing Delay

- Steady-state queuing delay

$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in} - R}{R} t & \lambda_{in} > R \end{cases}$$



ideal input flow
$\lambda_{in}$ constant



realistic input flow
$\lambda_{in}$ variable

# Queuing Delay

# Queuing Delay

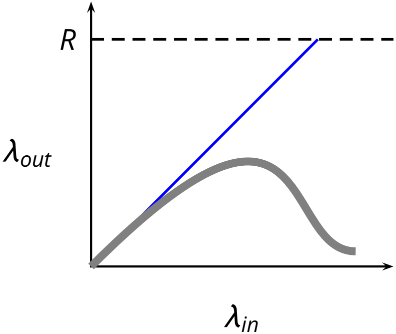
- **Conclusion:** as the input rate $\lambda_{in}$ approaches the maximum throughput $R$, packets will experience very long delays

# Queuing Delay

- **Conclusion:** as the input rate $\lambda_{in}$ approaches the maximum throughput $R$, packets will experience very long delays
- More realistic assumptions and models
  - finite queue length (buffers) in routers
  - effects of retransmission overhead
  - full queues along multi-hops paths

# Queuing Delay

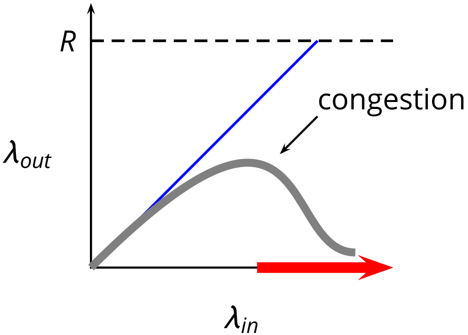
- **Conclusion:** as the input rate $\lambda_{in}$ approaches the maximum throughput $R$, packets will experience very long delays
- More realistic assumptions and models
  - finite queue length (buffers) in routers
  - effects of retransmission overhead
  - full queues along multi-hops paths



# Queuing Delay

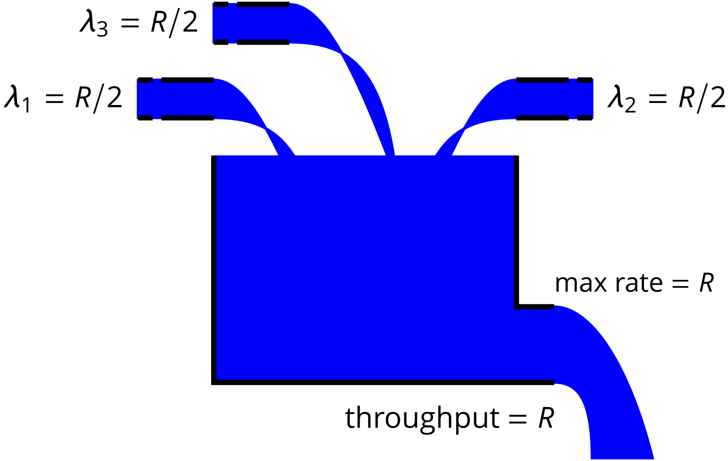
- **Conclusion:** as the input rate $\lambda_{in}$ approaches the maximum throughput $R$, packets will experience very long delays

- More realistic assumptions and models
  - finite queue length (buffers) in routers
  - effects of retransmission overhead
  - full queues along multi-hops paths



# What to Do?

- What to do when the network is congested?



$\lambda_3 = R/2$

$\lambda_1 = R/2$

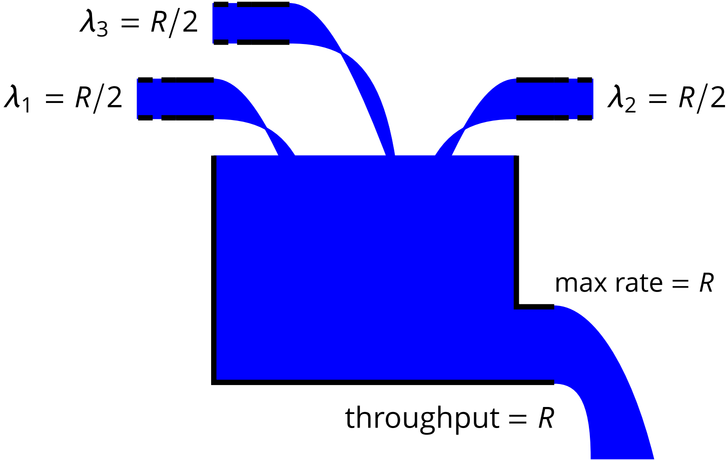
$\lambda_2 = R/2$

max rate $= R$

throughput $= R$

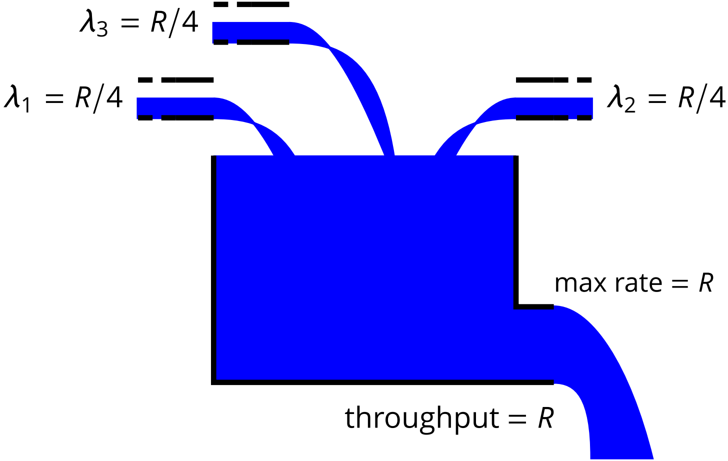
# What to Do?

- What to do when the network is congested? ***BACK OFF!***



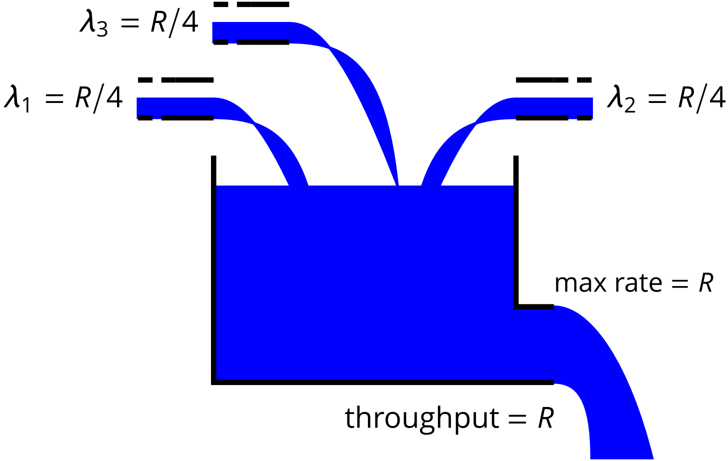
# What to Do?

- What to do when the network is congested? ***BACK OFF!***



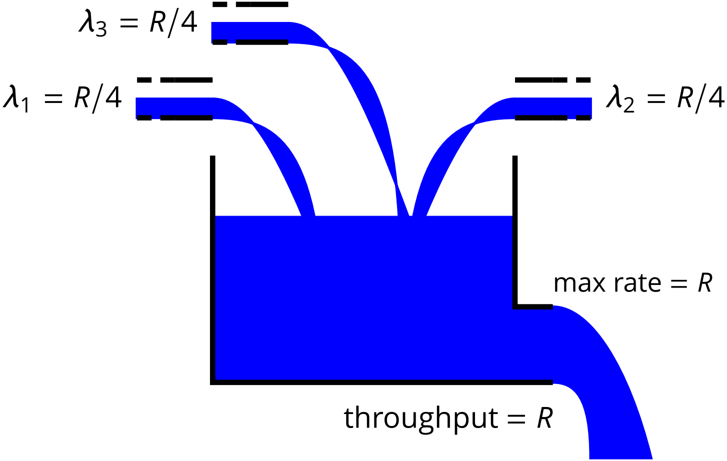
# What to Do?

- What to do when the network is congested? ***BACK OFF!***



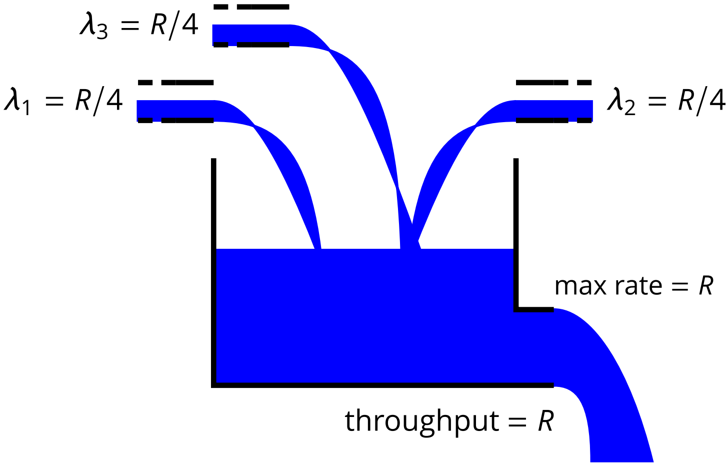
# What to Do?

- What to do when the network is congested? ***BACK OFF!***



# What to Do?

- What to do when the network is congested? ***BACK OFF!***



# Congestion Control (in TCP)

**Approach:**

- ***The sender limits its output rate according to the state of the network***
  - The sender output rate becomes (part of) the input rate for the network ($\lambda_{in}$)

# Congestion Control (in TCP)

**Approach:**

- ***The sender limits its output rate according to the state of the network***
  - The sender output rate becomes (part of) the input rate for the network ($\lambda_{in}$)

**Ingredients:**

1. How does the sender ***measure the state of the network***?
   - we need ***eyes*** to see the traffic ahead

# Congestion Control (in TCP)

**Approach:**

- ***The sender limits its output rate according to the state of the network***
  - The sender output rate becomes (part of) the input rate for the network ($\lambda_{in}$)

**Ingredients:**

1. How does the sender ***measure the state of the network***?
   - we need ***eyes*** to see the traffic ahead

2. how does the sender ***set its output rate***?
   - we need ***accelerator*** and ***brakes*** to speed up or slow down

# Congestion Control (in TCP)

**Approach:**

- ***The sender limits its output rate according to the state of the network***
  - The sender output rate becomes (part of) the input rate for the network ($\lambda_{in}$)

**Ingredients:**

1. How does the sender ***measure the state of the network***?
   - we need ***eyes*** to see the traffic ahead

2. how does the sender ***set its output rate***?
   - we need ***accelerator*** and ***brakes*** to speed up or slow down

3. how should the sender ***control its output rate***?
   - we need a ***brain*** and we need to know ***how to drive!***

# Detecting Congestion (Eyes)

# Detecting Congestion (Eyes)

- If all traffic is correctly acknowledged, with fresh acknowledgments, then the sender assumes (quite correctly) that there is no congestion

# Detecting Congestion (Eyes)

- If all traffic is correctly acknowledged, with fresh acknowledgments, then the sender assumes (quite correctly) that there is no congestion

- Congestion means that some queues overflow in one or more routers between the sender and the receiver
  - the visible effect is that some segments are dropped

# Detecting Congestion (Eyes)

- If all traffic is correctly acknowledged, with fresh acknowledgments, then the sender assumes (quite correctly) that there is no congestion

- Congestion means that some queues overflow in one or more routers between the sender and the receiver
  - the visible effect is that some segments are dropped

- Therefore the sender assumes that the network is congested when it (the sender) detects a segment loss
  - duplicate acknowledgements (i.e., NACK)
  - time out (i.e., no ACKs at all)

# Congestion Window (Accellerator/Brakes)

- The sender maintains a ***congestion window*** *W*

# Congestion Window (Accellerator/Brakes)

- The sender maintains a ***congestion window*** *W*
- The congestion window limits the amount of bytes that the sender pushes into the network before blocking waiting for acknowledgments

# Congestion Window (Accellerator/Brakes)

- The sender maintains a ***congestion window*** $W$

- The congestion window limits the amount of bytes that the sender pushes into the network before blocking waiting for acknowledgments

$$LastByteSent - LastByteAcked \leq W$$

where

$$W = \min\left(CongestionWindow, ReceiverWindow\right)$$

# Congestion Window (Accellerator/Brakes)

- The sender maintains a ***congestion window*** $W$

- The congestion window limits the amount of bytes that the sender pushes into the network before blocking waiting for acknowledgments

$$LastByteSent - LastByteAcked \leq W$$

where

$$W = \min\left(CongestionWindow, ReceiverWindow\right)$$

- The resulting maximum output rate is roughly

$$\lambda = \frac{W}{2L}$$

# Congestion Control (Brain, Algorithm)

# Congestion Control (Brain, Algorithm)

- *Additive-increase and multiplicative-decrease*

# Congestion Control (Brain, Algorithm)

- *Additive-increase and multiplicative-decrease*
- *Slow start*

# Congestion Control (Brain, Algorithm)

- *Additive-increase and multiplicative-decrease*
- *Slow start*
- *Reaction to timeout events*

# Additive-Increase/Multiplicative-Decrease

# Additive-Increase/Multiplicative-Decrease

- ***How** W **is reduced:*** at every *loss* event, TCP halves the congestion window

# Additive-Increase/Multiplicative-Decrease

- ***How** W **is reduced:*** at every *loss* event, TCP halves the congestion window
  - e.g., suppose the window size *W* is currently 20Kb, and a loss is detected
  - TCP reduces *W* to 10Kb

# Additive-Increase/Multiplicative-Decrease

- ***How $W$ is reduced:*** at every *loss* event, TCP halves the congestion window
  - e.g., suppose the window size $W$ is currently 20Kb, and a loss is detected
  - TCP reduces $W$ to 10Kb

- ***How $W$ is increased:*** at every (good) acknowledgment, TCP increments $W$ by $1MSS/W$, so as to increase $W$ by $MSS$ every round-trip time $2L$. This process is called ***congestion avoidance***

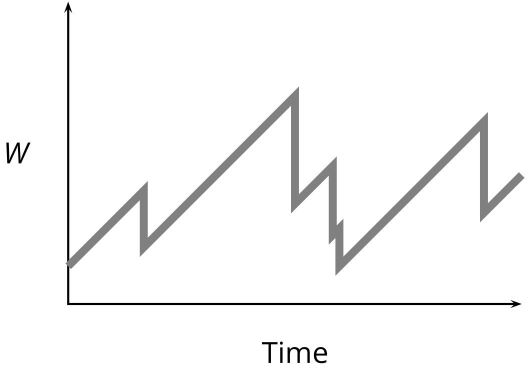
# Additive-Increase/Multiplicative-Decrease

- ***How** $W$ **is reduced:*** at every *loss* event, TCP halves the congestion window
  - e.g., suppose the window size $W$ is currently 20Kb, and a loss is detected
  - TCP reduces $W$ to 10Kb

- ***How** $W$ **is increased:*** at every (good) acknowledgment, TCP increments $W$ by $1MSS/W$, so as to increase $W$ by $MSS$ every round-trip time $2L$. This process is called ***congestion avoidance***
  - e.g., suppose $W = 14600$ and $MSS = 1460$, then the sender increases $W$ to 16060 after 10 acknowledgments acknowledgments

# Additive-Increase/Multiplicative-Decrease

- Window size *W* over time



# Slow Start

- What is the initial value of $W$?

# Slow Start

- What is the initial value of *W*?

- The initial value of *W* is *MSS*, that is ***1 segment***, which is quite low for modern networks

# Slow Start

- What is the initial value of *W*?
- The initial value of *W* is *MSS*, that is ***1 segment***, which is quite low for modern networks
- To get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase, up to a ***slow-start threshold*** (***ssthresh***)

# Slow Start

- What is the initial value of *W*?

- The initial value of *W* is *MSS*, that is ***1 segment***, which is quite low for modern networks

- To get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase, up to a ***slow-start threshold*** (***ssthresh***)

- After the threshold, TCP proceeds with its linear push

# Slow Start

- What is the initial value of *W*?
- The initial value of *W* is *MSS*, that is ***1 segment***, which is quite low for modern networks
- To get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase, up to a ***slow-start threshold*** (***ssthresh***)
- After the threshold, TCP proceeds with its linear push
- This process is called “slow start” because of the small initial value of *W*

# Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK

# Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network

# Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK

- Both timeouts and NACKs signal a loss, but they say different things about the status of the network

- A ***timeout indicates congestion***

# Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network
- A ***timeout indicates congestion***
- Three (duplicate) ACKs suggest that the network is still able to deliver segments along that path

# Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network
- A ***timeout indicates congestion***
- Three (duplicate) ACKs suggest that the network is still able to deliver segments along that path
- So, TCP reacts differently to a timeout and to a triple duplicate ACKs

# Timeouts vs. NACKs

Assuming the current window size is $W = \overline{W}$

# Timeouts vs. NACKs

Assuming the current window size is $W = \overline{W}$

- *Timeout*
  - go back to $W = MSS$
  - set $ssthresh = \overline{W}/2$
  - run *slow start* up to $W = ssthresh$
  - then proceed with *congestion avoidance*

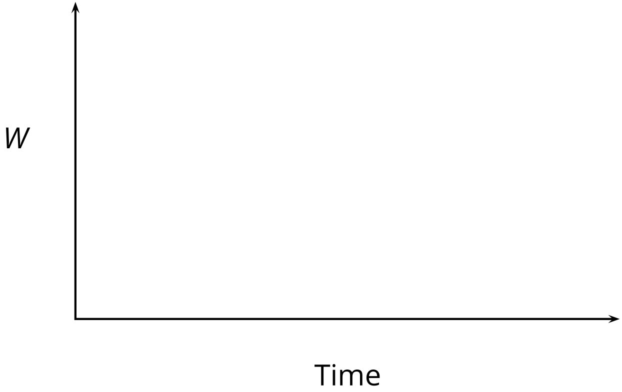
# Timeouts vs. NACKs

Assuming the current window size is $W = \overline{W}$

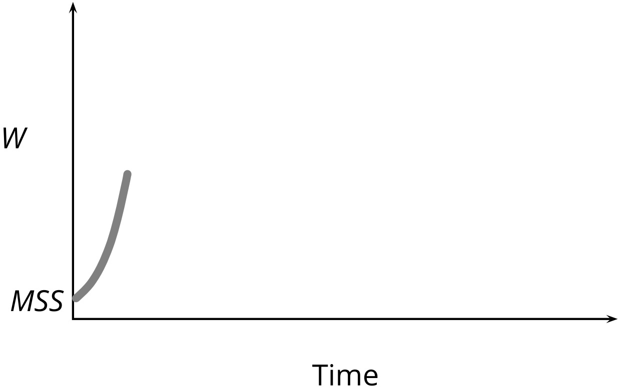
- *Timeout*
  - go back to $W = MSS$
  - set $ssthresh = \overline{W}/2$
  - run *slow start* up to $W = ssthresh$
  - then proceed with *congestion avoidance*

- *NACK* (i.e., triple duplicate-ack)
  - set $ssthresh = \overline{W}/2$
  - cut $W$ in half: $W = \overline{W}/2$
  - run *congestion avoidance*, ramping up $W$ linearly
  - This is called ***fast recovery***

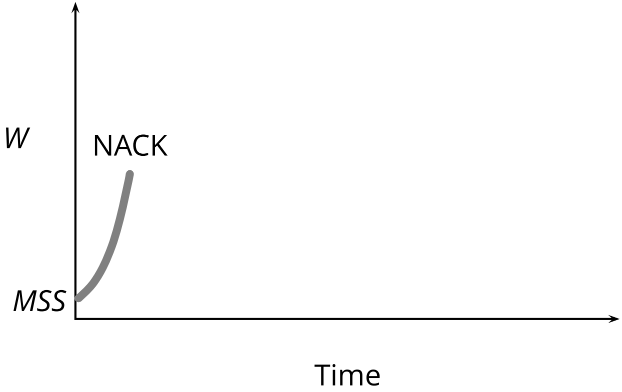
# Sender Behavior



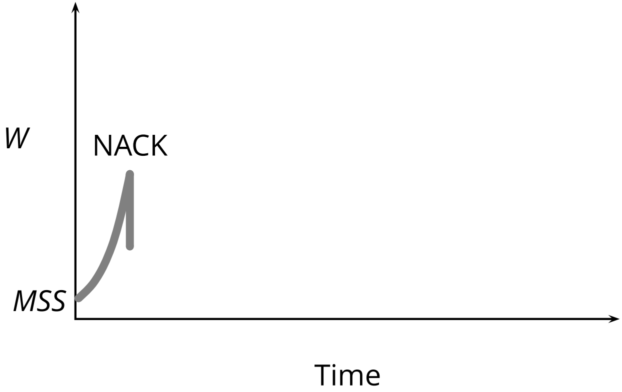
# Sender Behavior



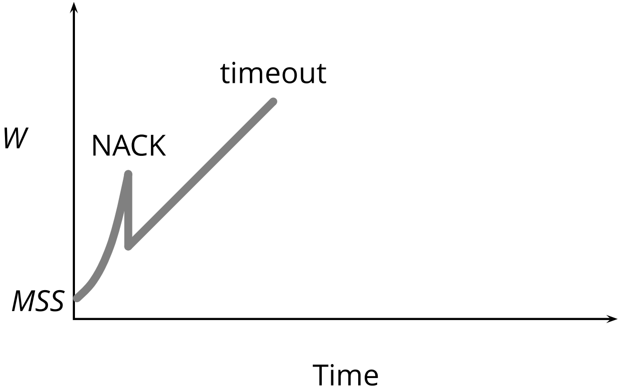
# Sender Behavior



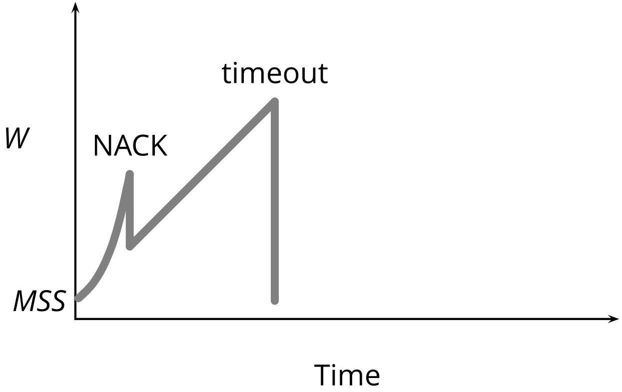
# Sender Behavior



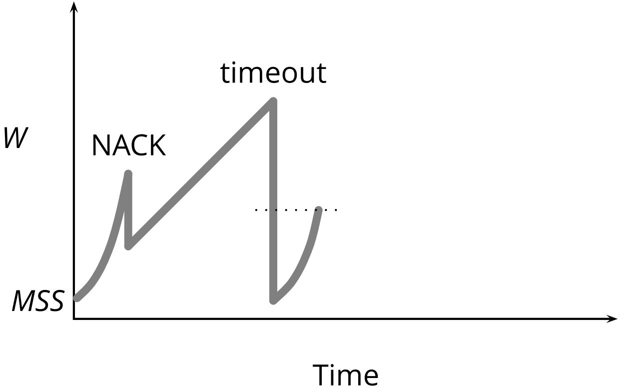
# Sender Behavior



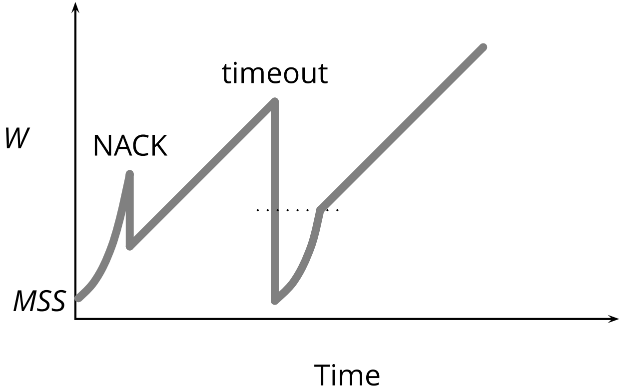
# Sender Behavior



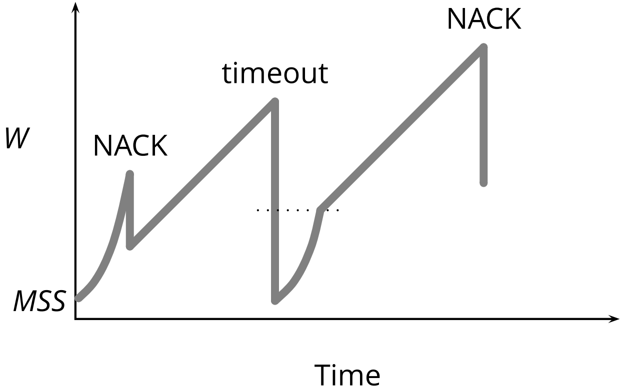
# Sender Behavior



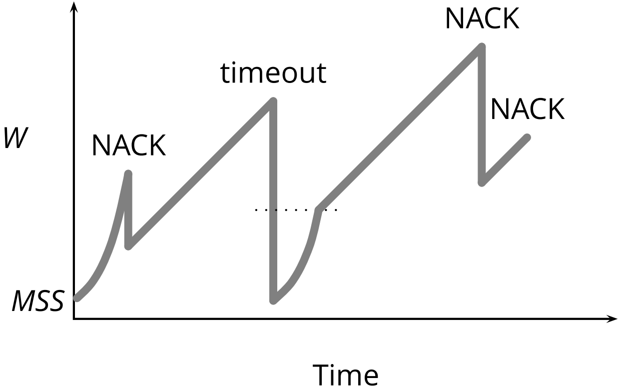
# Sender Behavior



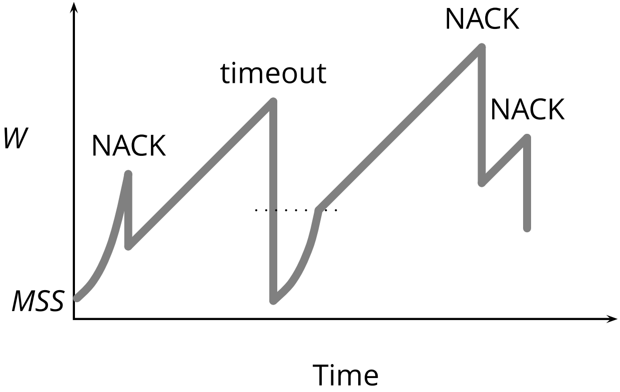
# Sender Behavior



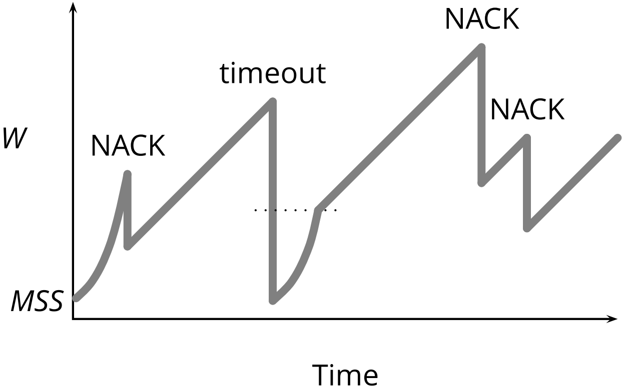
# Sender Behavior



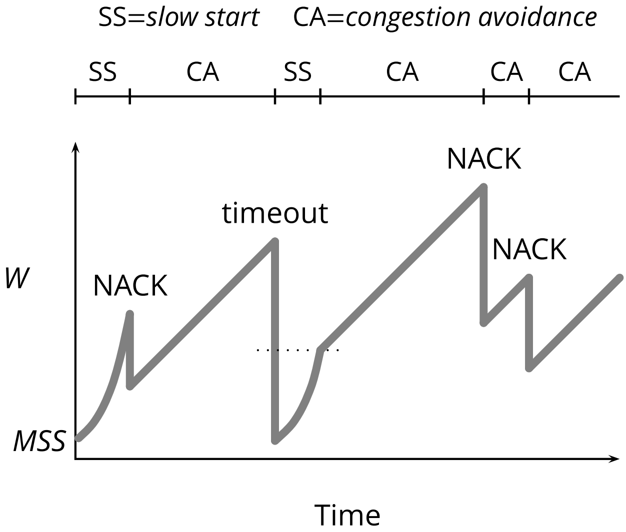
# Sender Behavior



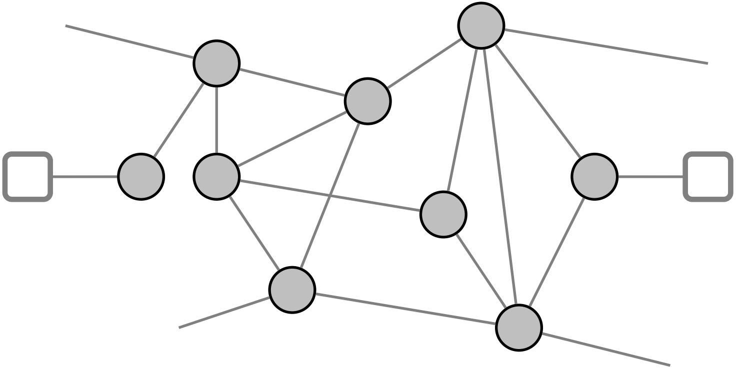
# Sender Behavior



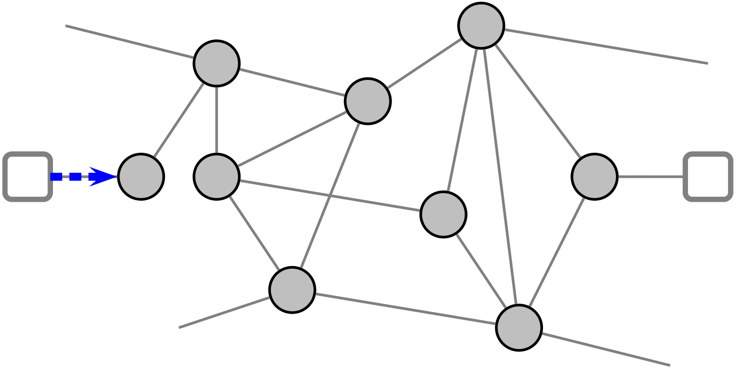
# Sender Behavior



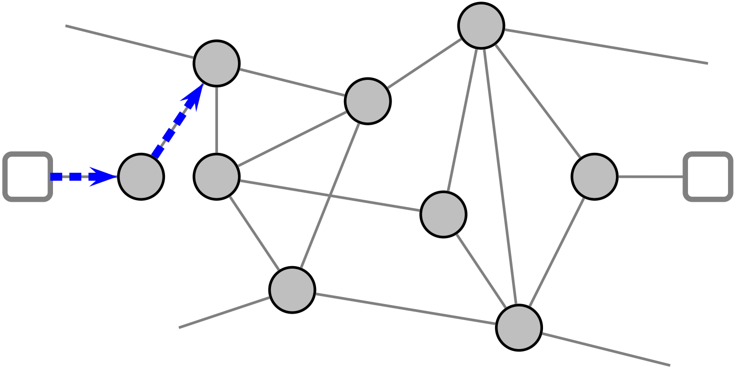
# Datagram Network



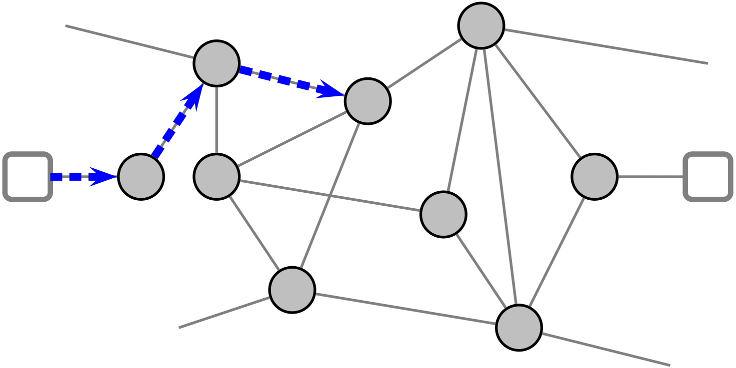
# Datagram Network



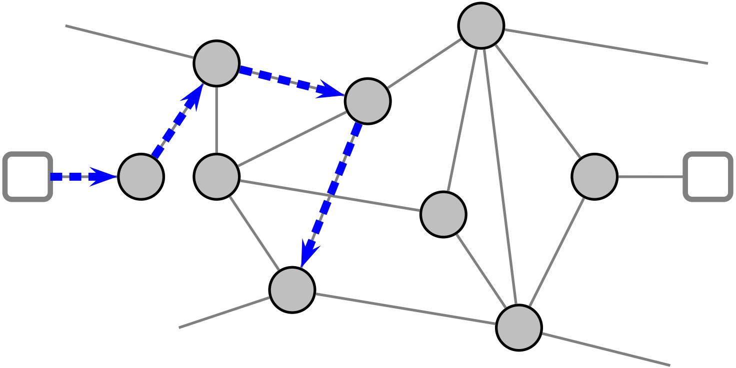
## Datagram Network



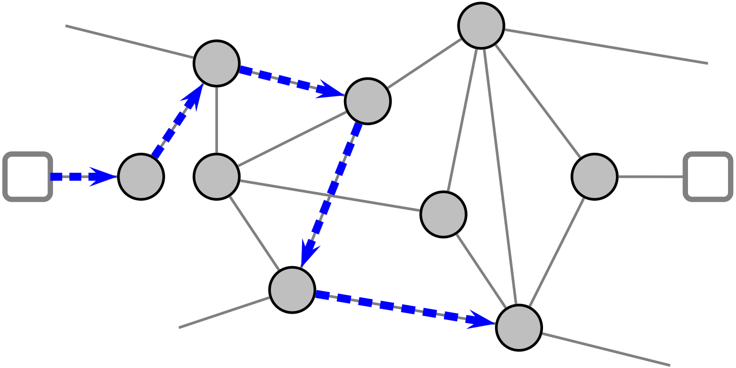
# Datagram Network



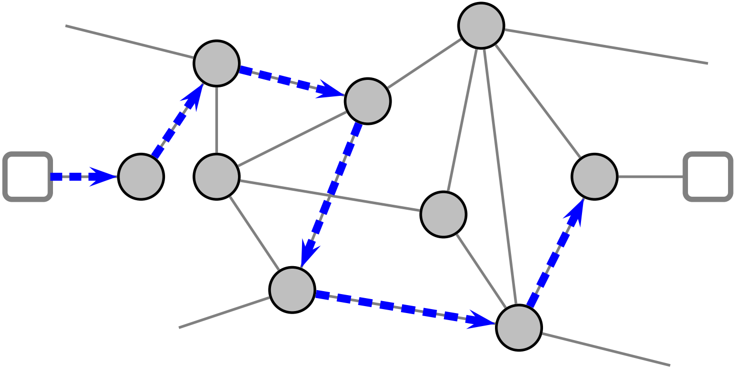
# Datagram Network



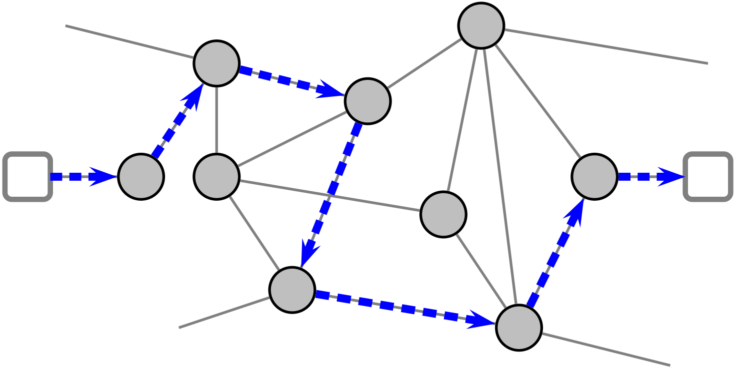
# Datagram Network



# Datagram Network

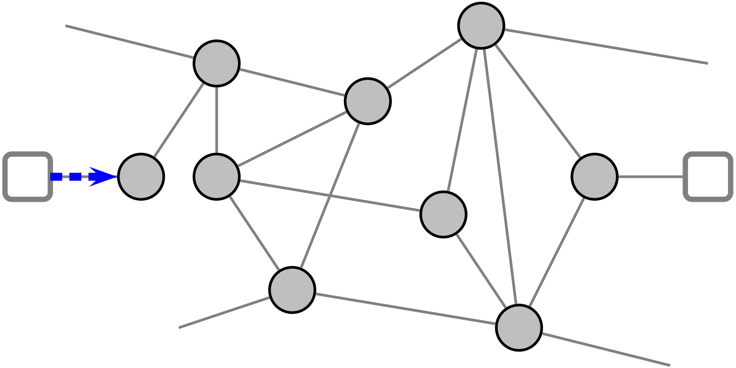


# Datagram Network



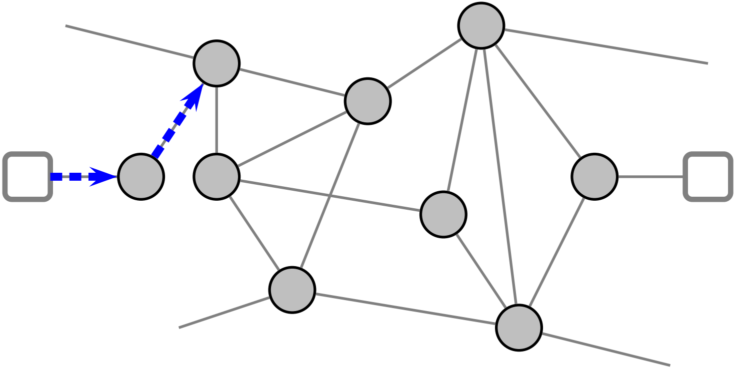
- Potentially *multiple paths* for the same source/destination

# Datagram Network



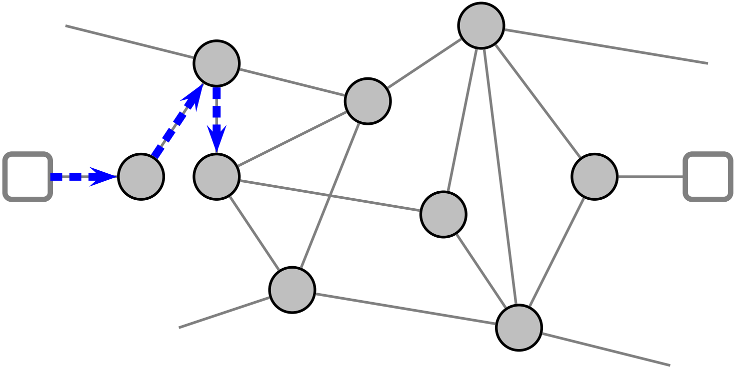
- Potentially *multiple paths* for the same source/destination

# Datagram Network



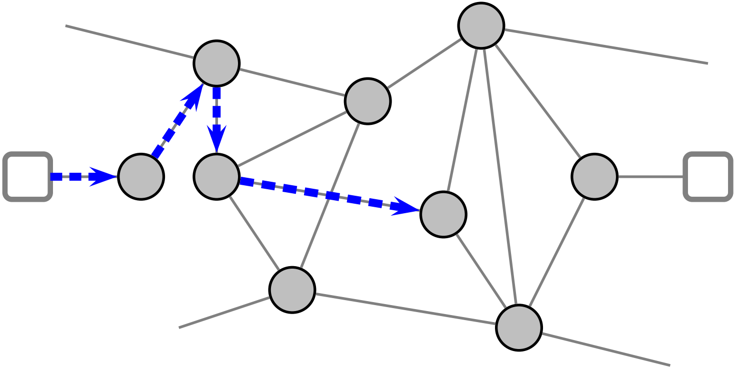
- Potentially *multiple paths* for the same source/destination

# Datagram Network



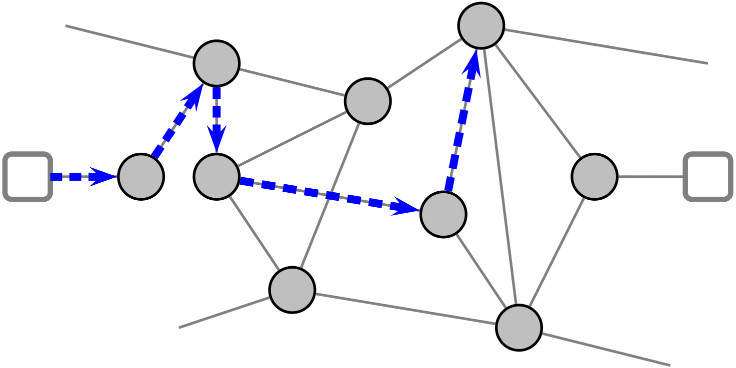
- Potentially *multiple paths* for the same source/destination

## Datagram Network



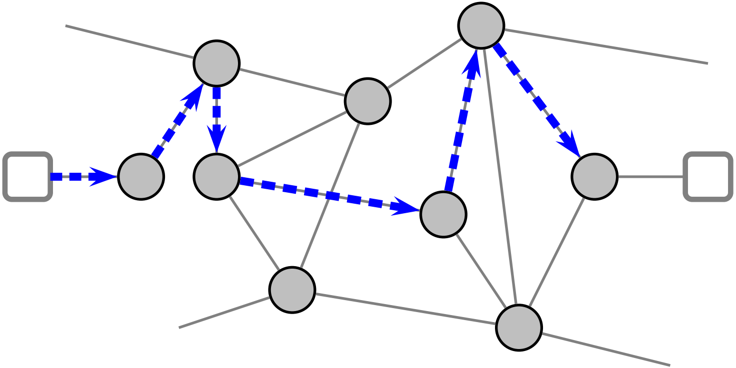
- Potentially *multiple paths* for the same source/destination

# Datagram Network



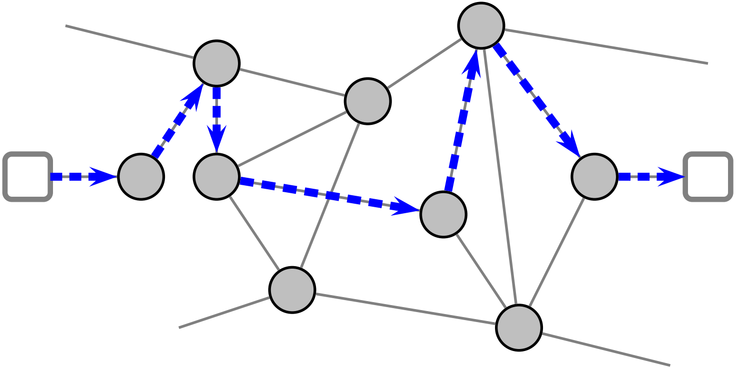
- Potentially *multiple paths* for the same source/destination

# Datagram Network



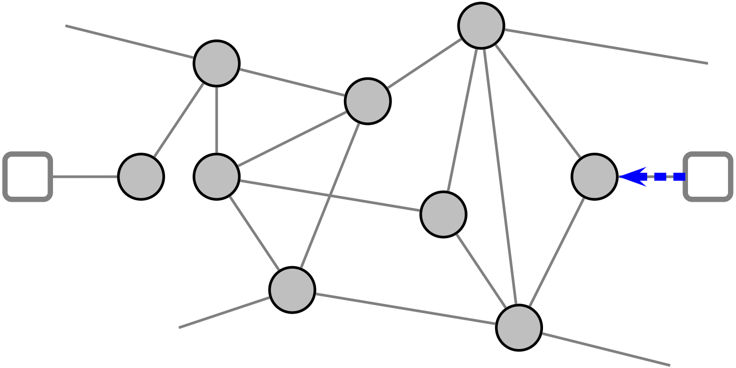
- Potentially *multiple paths* for the same source/destination

# Datagram Network



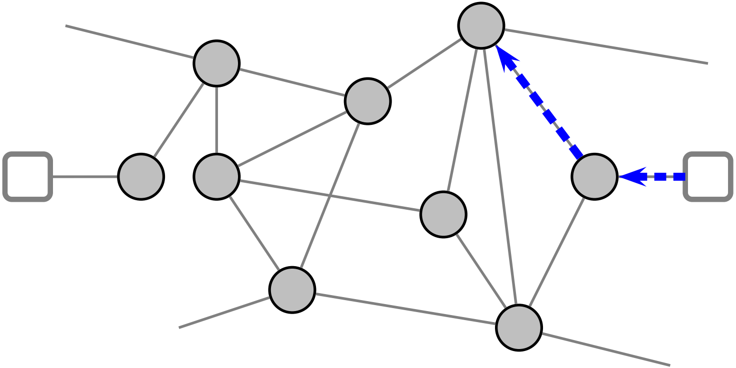
- Potentially *multiple paths* for the same source/destination

# Datagram Network



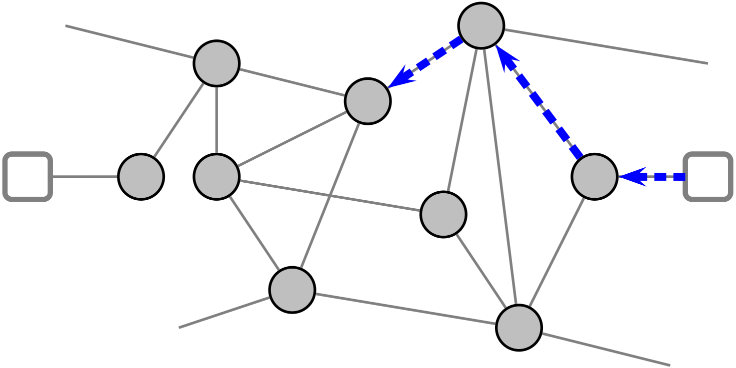
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

# Datagram Network



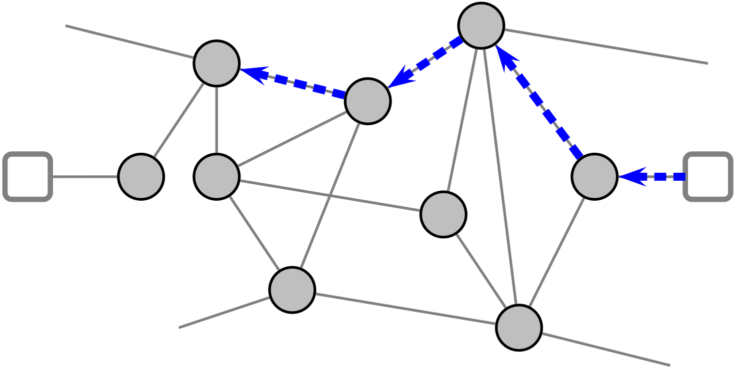
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

# Datagram Network



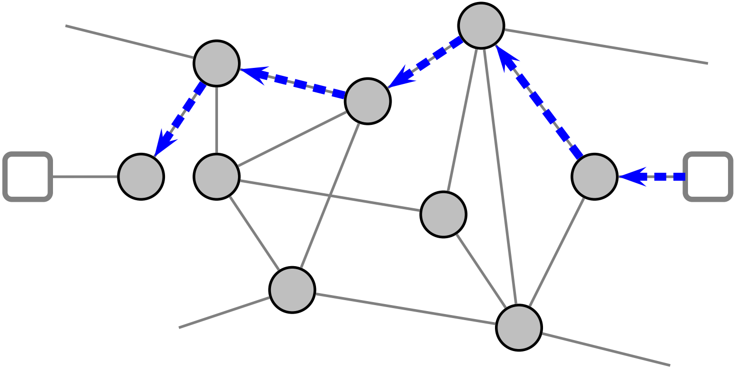
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

# Datagram Network



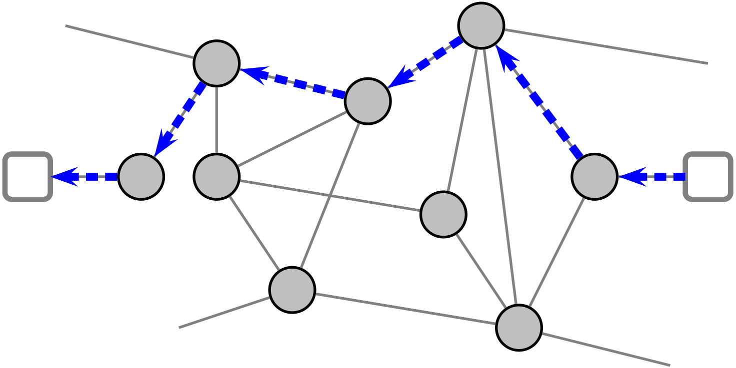
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

# Datagram Network



- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

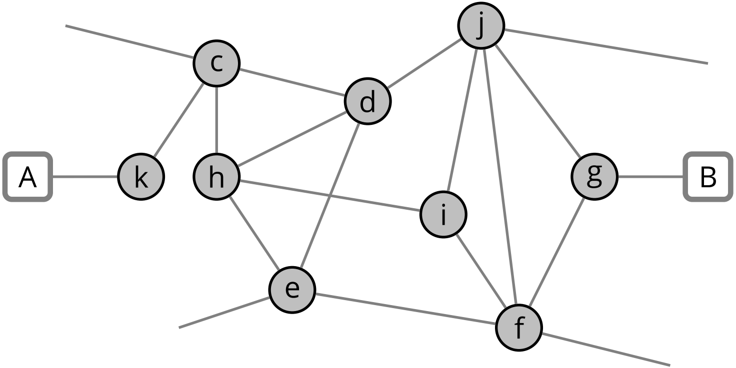
# Datagram Network



- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

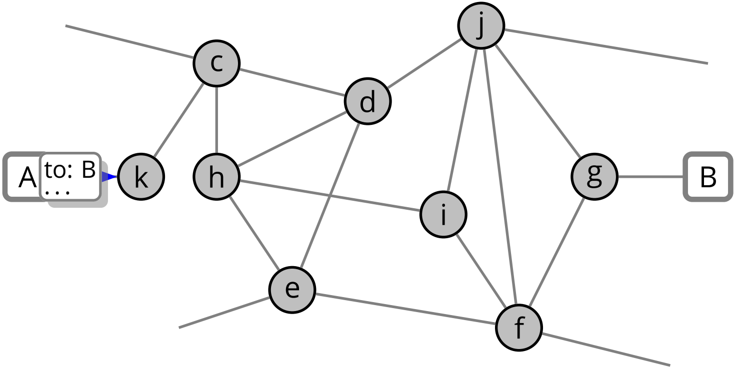
# Forwarding

# Forwarding



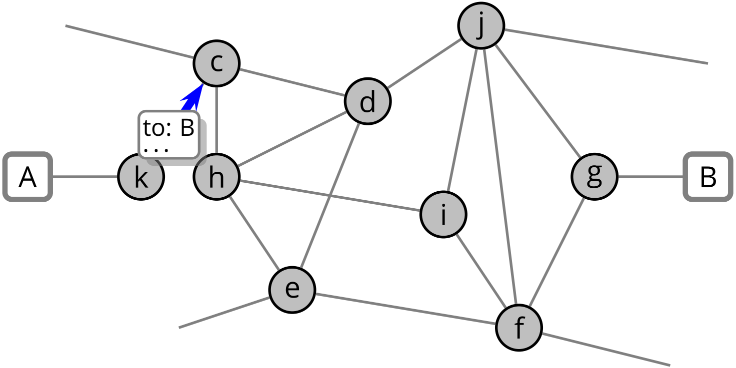
- *A* sends a datagram to *B*

# Forwarding



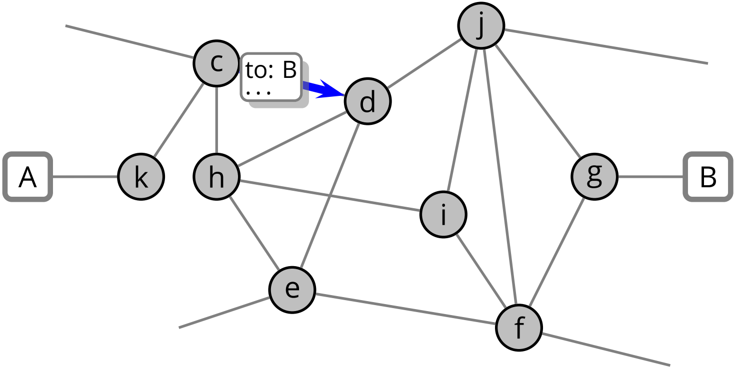
- *A* sends a datagram to *B*
- The datagram is ***forwarded*** towards *B*

# Forwarding

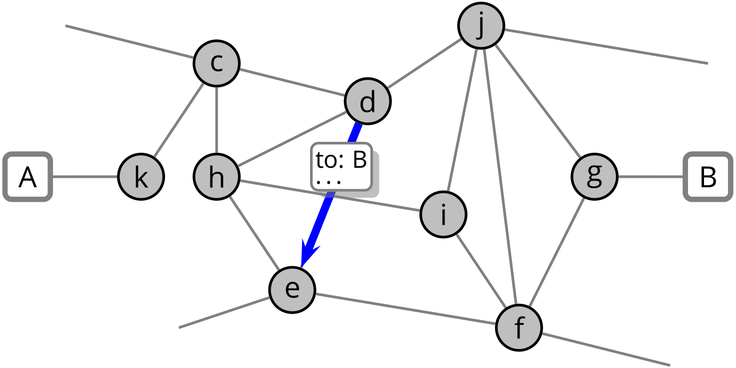


- *A* sends a datagram to *B*
- The datagram is ***forwarded*** towards *B*

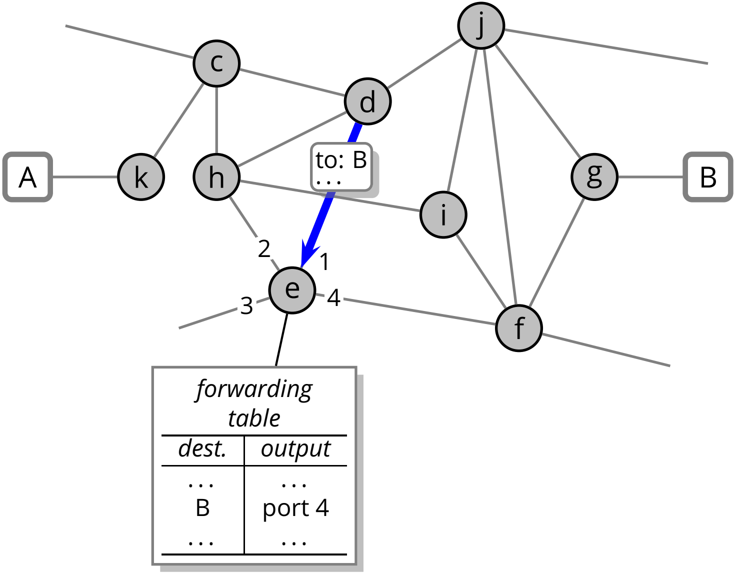
# Forwarding



# Forwarding



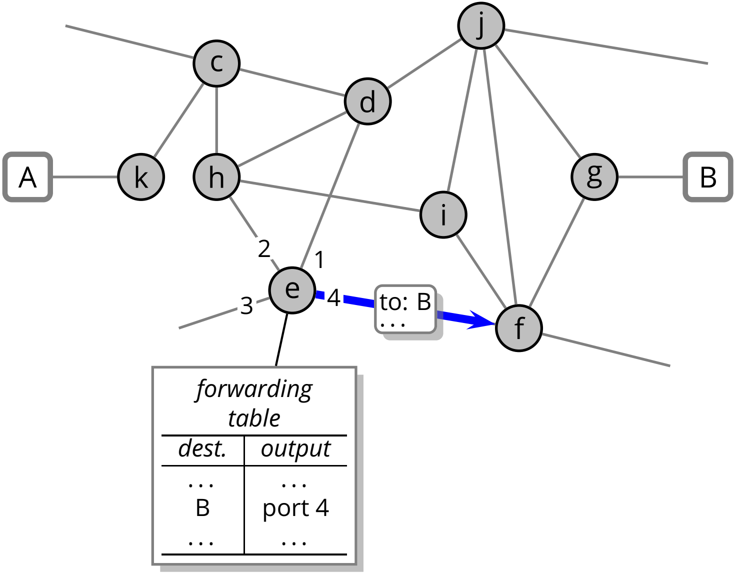
# Forwarding



*forwarding table*

| *dest.* | *output* |
|---|---|
| . . . | . . . |
| B | port 4 |
| . . . | . . . |

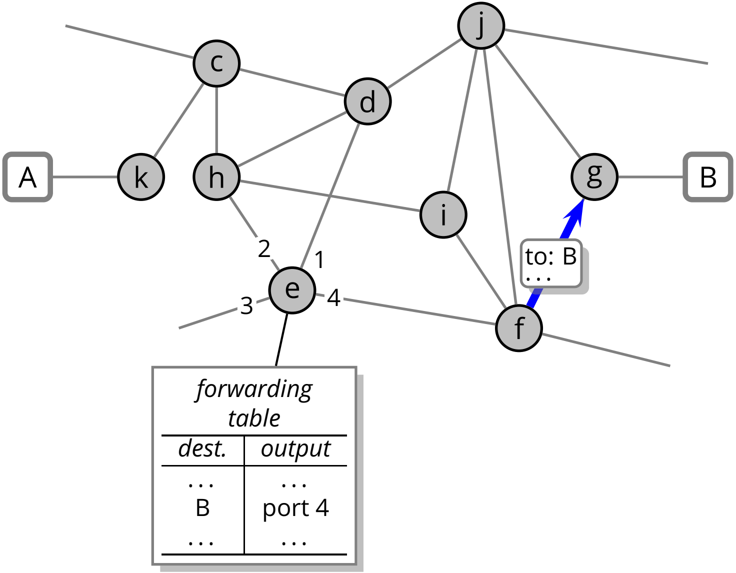
# Forwarding



*forwarding table*

| *dest.* | *output* |
|---|---|
| . . . | . . . |
| B | port 4 |
| . . . | . . . |

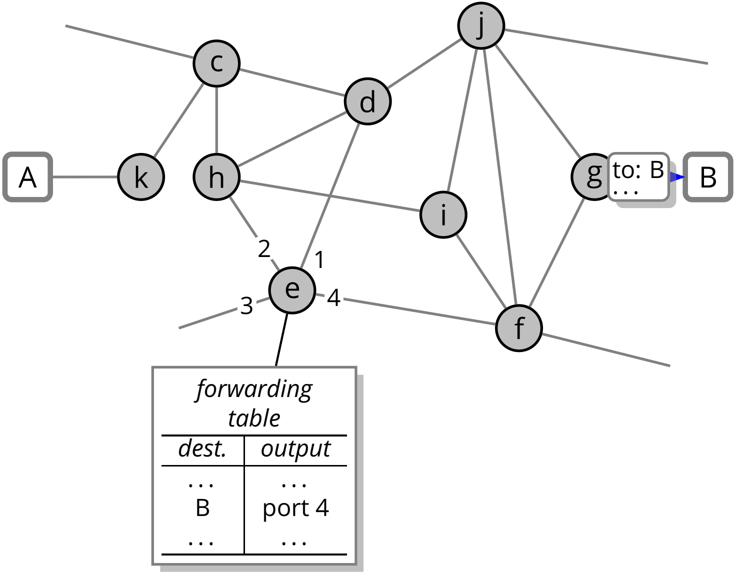
# Forwarding



*forwarding table*

| *dest.* | *output* |
|---|---|
| . . . | . . . |
| B | port 4 |
| . . . | . . . |

# Forwarding



*forwarding table*

| *dest.* | *output* |
|---|---|
| . . . | . . . |
| B | port 4 |
| . . . | . . . |

# Forwarding

- *Input:* datagram destination

# Forwarding

- *Input:* datagram destination
- *Output:* output port

# Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”

# Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues

# Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues
    - how big is the forwarding table?

# Forwarding

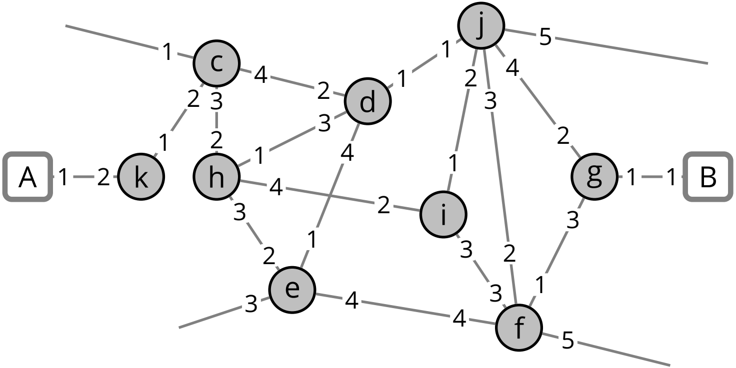
- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues
  - how big is the forwarding table?
  - how fast does the router have to forward datagrams?

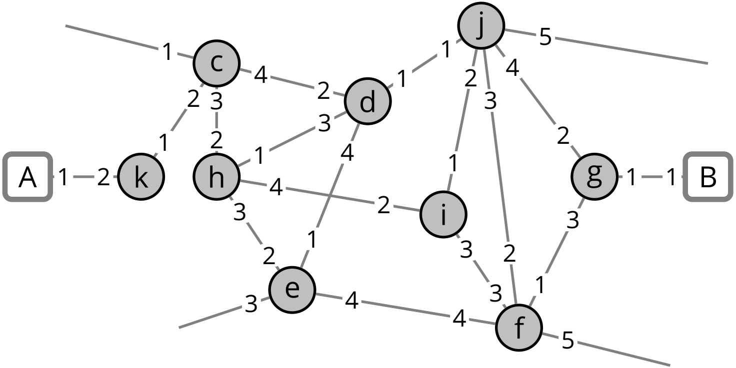
# Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues
    - how big is the forwarding table?
    - how fast does the router have to forward datagrams?
    - how does the router build and maintain the forwarding table?

# Routing

# Routing



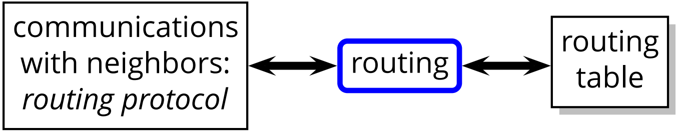


| *router k* | |
|---|---|
| A | 2 |
| B | 1 |

. . .

# Router Functions

# Router Functions

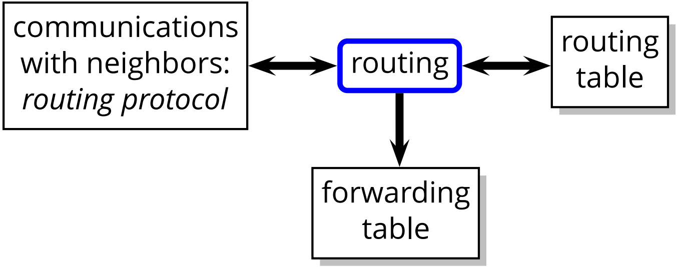


communications with neighbors: *routing protocol*

routing

routing table

# Router Functions



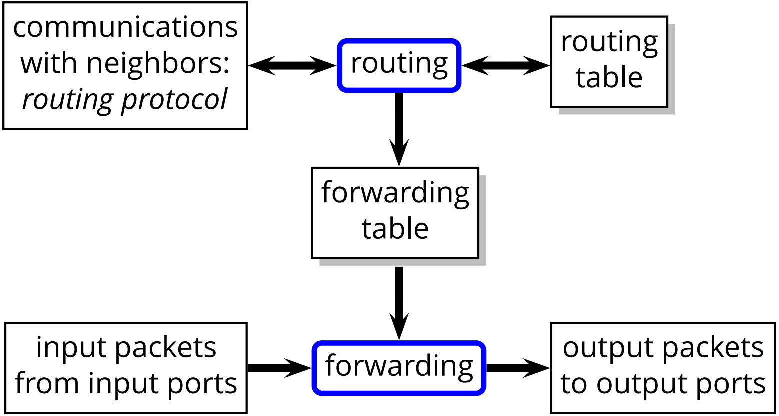
communications with neighbors: *routing protocol*

routing

routing table

forwarding table

# Router Functions



communications with neighbors: *routing protocol*

routing

routing table

forwarding table

input packets from input ports

forwarding

output packets to output ports

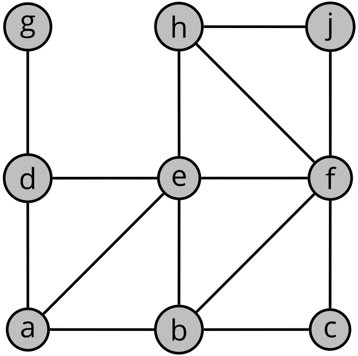
# Routing Problem

# Routing Problem

- Finding paths through a network

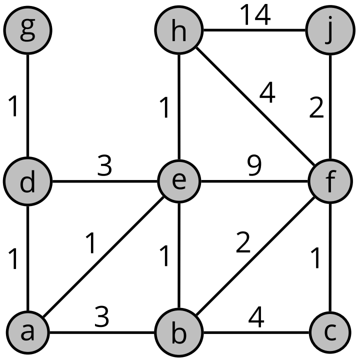
# Routing Problem

- Finding paths through a network



# Routing Problem

- Finding paths through a network



- Example: $a \longrightarrow j$?

# Graph Model

- The network is modeled as a graph

$$G = (V, E)$$

# Graph Model

- The network is modeled as a graph

$$G = (V, E)$$

  - $V$ is a set of ***vertices*** representing the routers

# Graph Model

- The network is modeled as a graph

$$G = (V, E)$$

  - $V$ is a set of ***vertices*** representing the routers
  - $E \subseteq V \times V$ is a set of ***edges*** representing communication links
    - e.g., $(u, v) \in E$ iff router $u$ is on the same subnet as $v$

# Graph Model

- The network is modeled as a graph

$$G = (V, E)$$

  - $V$ is a set of ***vertices*** representing the routers
  - $E \subseteq V \times V$ is a set of ***edges*** representing communication links
    - e.g., $(u, v) \in E$ iff router $u$ is on the same subnet as $v$
  - $G$ is assumed to be an ***undirected graph***, meaning that ***links are bidirectional***
    - i.e., $(u, v) \in E \Leftrightarrow (v, u) \in E$ for all $u, v \in N$

# Graph Model

- The network is modeled as a graph

$$G = (V, E)$$

  - $V$ is a set of ***vertices*** representing the routers
  - $E \subseteq V \times V$ is a set of ***edges*** representing communication links
    - e.g., $(u, v) \in E$ iff router $u$ is on the same subnet as $v$
  - $G$ is assumed to be an ***undirected graph***, meaning that ***links are bidirectional***
    - i.e., $(u, v) \in E \Leftrightarrow (v, u) \in E$ for all $u, v \in N$
  - A ***cost*** function $c : E \longrightarrow \mathbb{R}$
    - costs are always positive: $c(e) > 0$ for all $e \in E$
    - links are symmetric: $c(u, v) = c(v, u)$ for all $u, v \in N$

# Routing in the Graph Model

- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$ such that

# Routing in the Graph Model

- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \rightarrow v} = u, x_1, x_2, \ldots, x_n, v$ such that
  - $P_{u \rightarrow v}$ is completely contained in the network graph $G$. I.e., $(u, x_1) \in V, (x_1, x_2) \in V, \ldots, (x_n, v) \in V$

# Routing in the Graph Model

- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$ such that
  - $P_{u \to v}$ is completely contained in the network graph $G$. I.e., $(u, x_1) \in V, (x_1, x_2) \in V, \ldots, (x_n, v) \in V$
  - $P_{u \to v}$ is a *least-cost path*, where the cost of the path is $c(P_{u \to v}) = c(u, x_1) + c(x_1, x_2) + \ldots + c(x_n, v)$

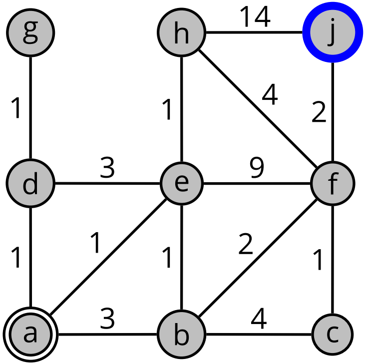
# Routing in the Graph Model

- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$ such that
  - $P_{u \to v}$ is completely contained in the network graph $G$. I.e., $(u, x_1) \in V, (x_1, x_2) \in V, \ldots, (x_n, v) \in V$
  - $P_{u \to v}$ is a *least-cost path*, where the cost of the path is $c(P_{u \to v}) = c(u, x_1) + c(x_1, x_2) + \ldots + c(x_n, v)$
- Compile *u*'s forwarding table by adding the following entry:

$$A(v) \longrightarrow I_u(x_1)$$

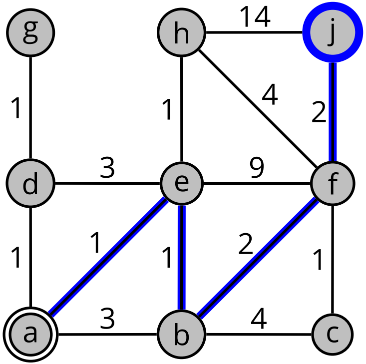
  - $A(v)$ is the address (or set of addresses) of router $v$
  - $I_u(x_1)$ is the interface that connects $u$ to the first next-hop router $x_1$ in $P_{u \to v} = u, x_1, x_2, \ldots, x_n, v$

# Back To The Example



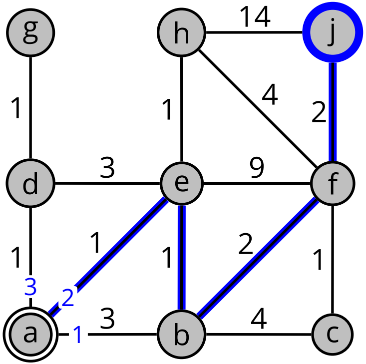
- Example: $a \longrightarrow j$

# Back To The Example



- Example: $a \longrightarrow j$
  - least-cost path is $P_{a \longrightarrow j} = a, e, b, f, j$

# Back To The Example



- Example: $a \longrightarrow j$
  - least-cost path is $P_{a \longrightarrow j} = a, e, b, f, j$
  - $a$'s forwarding table will contain an entry $\boxed{j \longrightarrow 2}$ since $l_a(e) = 2$

# Two General Strategies

- There are two main strategies to implement a routing algorithm

# Two General Strategies

- There are two main strategies to implement a routing algorithm
- ***Link-state routing***

# Two General Strategies

- There are two main strategies to implement a routing algorithm

- ***Link-state routing***
    - global view of the network
    - local computation of least-cost paths

# Two General Strategies

- There are two main strategies to implement a routing algorithm

- ***Link-state routing***
    - global view of the network
    - local computation of least-cost paths

- ***Distance-vector routing***

# Two General Strategies

- There are two main strategies to implement a routing algorithm

- ***Link-state routing***
  - global view of the network
  - local computation of least-cost paths

- ***Distance-vector routing***
  - local view of the network
  - global computation of least-cost paths

# Link-State Routing

- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)

# Link-State Routing

- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
  - every router *v* advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
  - ***link-state advertisements (LSAs)*** are broadcast through the entire network

# Link-State Routing

- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
    - every router *v* advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
    - ***link-state advertisements (LSAs)*** are broadcast through the entire network
    - routers collect link-state advertisements from other routers, and they use them to compile and maintain a complete view of *G*

# Link-State Routing

- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
  - every router *v* advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
  - ***link-state advertisements (LSAs)*** are broadcast through the entire network
  - routers collect link-state advertisements from other routers, and they use them to compile and maintain a complete view of *G*

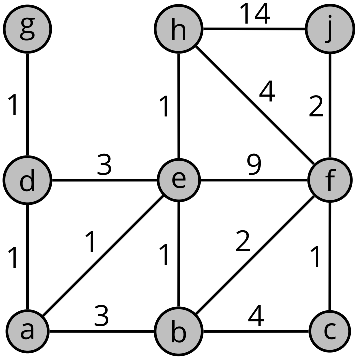
- Using its local representation of *G*, router *u* computes the least-cost paths from *u* to every other router in the network

# Link-State Routing

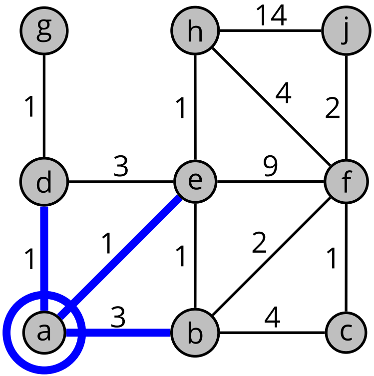
- Router *u* maintains a complete view of the network graph *G* (including all links and their costs)
    - every router *v* advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
    - ***link-state advertisements (LSAs)*** are broadcast through the entire network
    - routers collect link-state advertisements from other routers, and they use them to compile and maintain a complete view of *G*

- Using its local representation of *G*, router *u* computes the least-cost paths from *u* to every other router in the network
    - the computation is local

# Link-State Advertisements

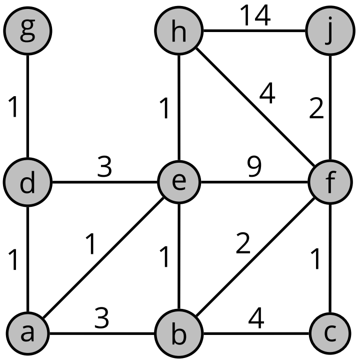


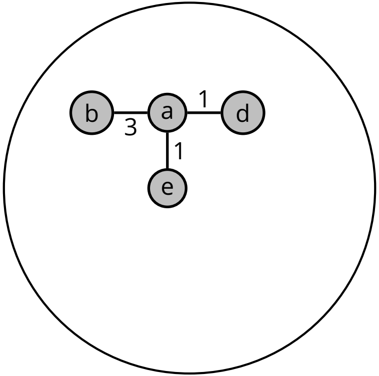
# Link-State Advertisements



$$LSA_a = \{(a,b,3),(a,e,1),(a,d,1)\}$$

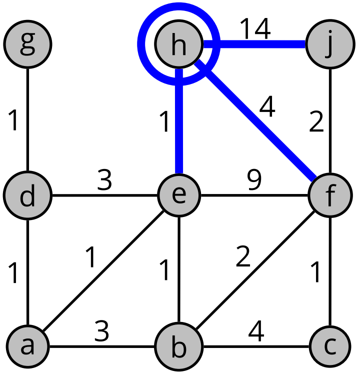
# Link-State Advertisements

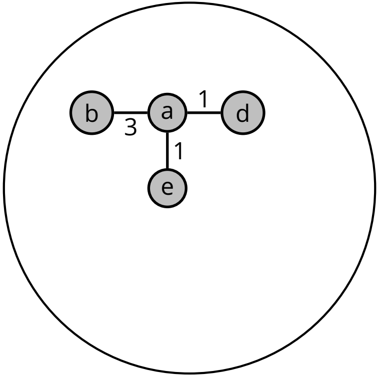




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$

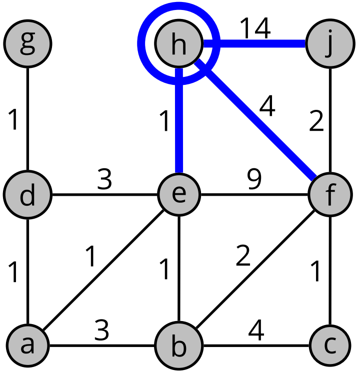
# Link-State Advertisements

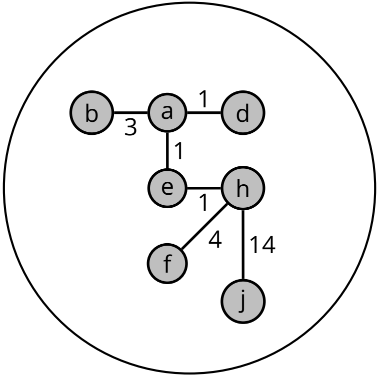




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

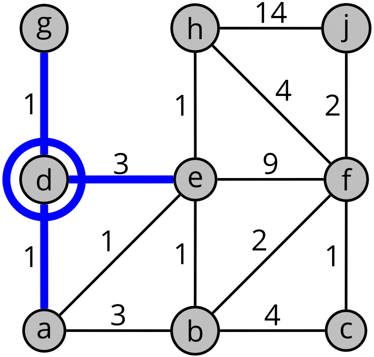
# Link-State Advertisements

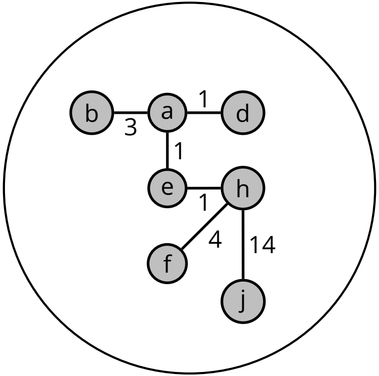




$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

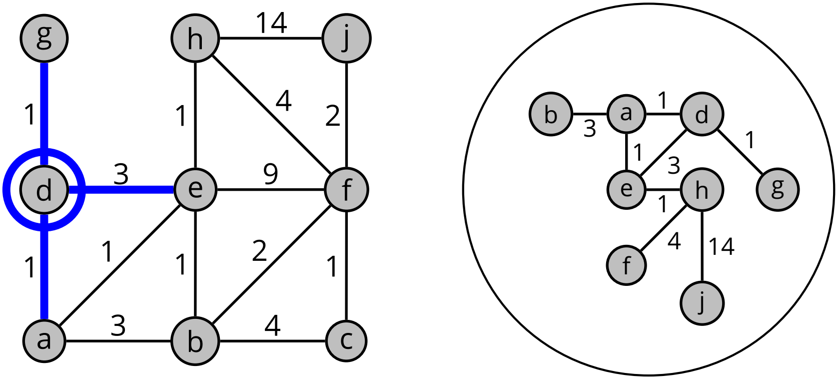
# Link-State Advertisements





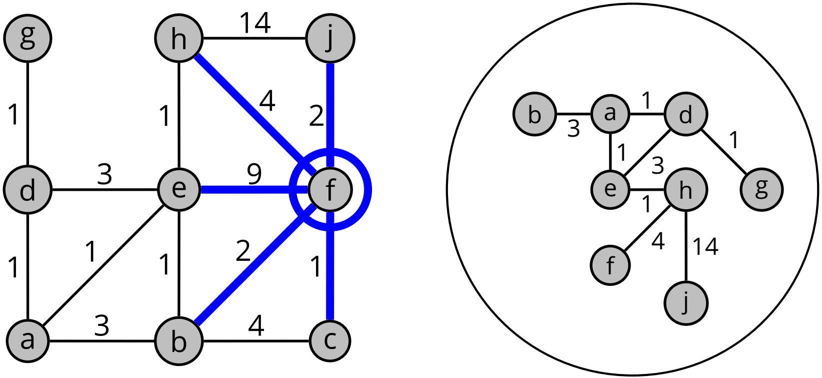
$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

# Link-State Advertisements



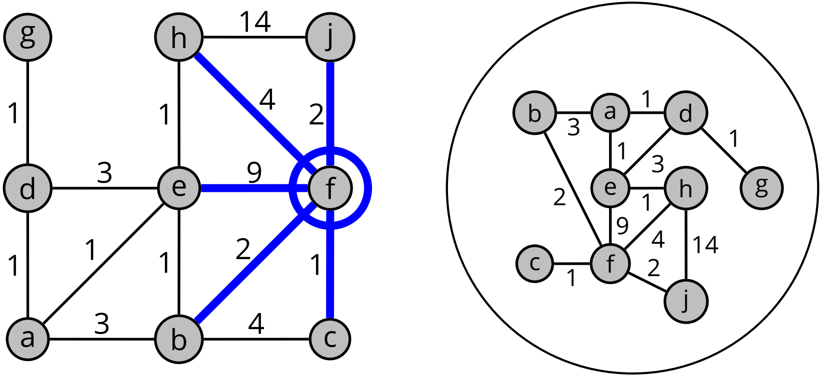
$$LSA_a = \{(a,b,3),(a,e,1),(a,d,1)\}$$
$$LSA_h = \{(h,e,1),(h,f,4),(h,j,14)\}$$
$$LSA_d = \{(d,a,1),(d,g,1),(d,e,3)\}$$

# Link-State Advertisements



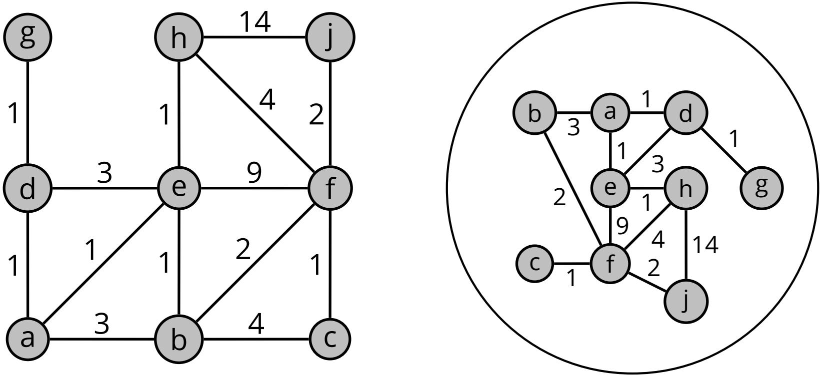
$$LSA_a = \{(a,b,3),(a,e,1),(a,d,1)\}$$
$$LSA_h = \{(h,e,1),(h,f,4),(h,j,14)\}$$
$$LSA_d = \{(d,a,1),(d,g,1),(d,e,3)\}$$
$$LSA_f = \{(f,c,1),(f,b,1),(f,e,3),(f,h,4),(f,j,2)\}$$

# Link-State Advertisements



$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$
$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$
$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$
$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

# Link-State Advertisements



$$
\begin{aligned}
LSA_a &= \{(a,b,3),(a,e,1),(a,d,1)\}\\
LSA_h &= \{(h,e,1),(h,f,4),(h,j,14)\}\\
LSA_d &= \{(d,a,1),(d,g,1),(d,e,3)\}\\
LSA_f &= \{(f,c,1),(f,b,1),(f,e,3),(f,h,4),(f,j,2)\}\\
&\dots
\end{aligned}
$$

# Link-State Routing Ingredients

What do we need to implement link-state routing?

# Link-State Routing Ingredients

What do we need to implement link-state routing?

- Every router sends its LSA to every other router in the network, so we need a ***broadcast routing scheme***

# Link-State Routing Ingredients

What do we need to implement link-state routing?

- Every router sends its LSA to every other router in the network, so we need a ***broadcast routing scheme***

- Once we have all the LSAs from every router, and therefore we complete knowledge of *G*, we need an ***algorithm to compute least-cost paths in a graph***

# Broadcast Routing

# Broadcast Routing

- ***Flooding***
  - every router forwards a broadcast packet to every adjacent router, except the one that sent the packet

# Broadcast Routing

- ***Flooding***
  - every router forwards a broadcast packet to every adjacent router, except the one that sent the packet

- Simple and elegant

# Broadcast Routing

- ***Flooding***
  - every router forwards a broadcast packet to every adjacent router, except the one that sent the packet

- Simple and elegant

- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

# Broadcast Routing

- ***Flooding***
  - every router forwards a broadcast packet to every adjacent router, except the one that sent the packet

- Simple and elegant

- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

- Any problem with this solution?

# Broadcast Routing

- ***Flooding***
  - every router forwards a broadcast packet to every adjacent router, except the one that sent the packet

- Simple and elegant

- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

- Any problem with this solution?
  - cycles in the network create *packet storms*

# Broadcast Routing (2)

# Broadcast Routing (2)

- ***Reverse-path broadcast***
  - every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
  - a router *u* accepts a broadcast packet *p* originating at router *s* only if *p* arrives on the link that is on the direct (unicast) path from *u* to *s*

# Broadcast Routing (2)

- ***Reverse-path broadcast***
  - every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
  - a router *u* accepts a broadcast packet *p* originating at router *s* only if *p* arrives on the link that is on the direct (unicast) path from *u* to *s*

- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

# Broadcast Routing (2)

- ***Reverse-path broadcast***
    - every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
    - a router *u* accepts a broadcast packet *p* originating at router *s* only if *p* arrives on the link that is on the direct (unicast) path from *u* to *s*

- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

- No packet storms even in the presence of cycles in *G*

# Broadcast Routing (2)

- ***Reverse-path broadcast***
  - every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
  - a router *u* accepts a broadcast packet *p* originating at router *s* only if *p* arrives on the link that is on the direct (unicast) path from *u* to *s*
- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router
- No packet storms even in the presence of cycles in *G*
- Any problem with this solution?

# Broadcast Routing (2)

- ***Reverse-path broadcast***
  - every router forwards a broadcast packet to every adjacent router, except the one where it received the packet router
  - a router *u* accepts a broadcast packet *p* originating at router *s* only if *p* arrives on the link that is on the direct (unicast) path from *u* to *s*

- Correct w.r.t. the broadcast requirement: a broadcast packet will eventually reach every router

- No packet storms even in the presence of cycles in *G*

- Any problem with this solution?
  - it requires (unicast) routing information
  - so it is obviously useless to implement a routing algorithm

# Broadcast Routing (3)

- ***Sequence-number controlled flooding***

# Broadcast Routing (3)

- ***Sequence-number controlled flooding***
  - the originator $s$ of a broadcast packet marks the packet with a sequence number $n_s$

# Broadcast Routing (3)

- ***Sequence-number controlled flooding***
  - the originator $s$ of a broadcast packet marks the packet with a sequence number $n_s$
  - every router $u$ stores the most recent sequence number seen from each source router. Let's assume that $u$ has seen sequence numbers from $s$ up to $n_s$

# Broadcast Routing (3)

- ***Sequence-number controlled flooding***
  - the originator $s$ of a broadcast packet marks the packet with a sequence number $n_s$
  - every router $u$ stores the most recent sequence number seen from each source router. Let's assume that $u$ has seen sequence numbers from $s$ up to $n_s$
  - a router accepts a broadcast packet $p$ originating at $s$ only if $p$ carries a sequence number $seq(p)$ that is higher than the most recent one seen from $s$: $seq(p) > n_s$

# Broadcast Routing (3)

- ***Sequence-number controlled flooding***
  - the originator $s$ of a broadcast packet marks the packet with a sequence number $n_s$
  - every router $u$ stores the most recent sequence number seen from each source router. Let's assume that $u$ has seen sequence numbers from $s$ up to $n_s$
  - a router accepts a broadcast packet $p$ originating at $s$ only if $p$ carries a sequence number $seq(p)$ that is higher than the most recent one seen from $s$: $seq(p) > n_s$
  - accepted packets are forwarded to every adjacent router, except the previous-hop router

# Broadcast Routing (3)

- ***Sequence-number controlled flooding***
  - the originator $s$ of a broadcast packet marks the packet with a sequence number $n_s$
  - every router $u$ stores the most recent sequence number seen from each source router. Let's assume that $u$ has seen sequence numbers from $s$ up to $n_s$
  - a router accepts a broadcast packet $p$ originating at $s$ only if $p$ carries a sequence number $seq(p)$ that is higher than the most recent one seen from $s$: $seq(p) > n_s$
  - accepted packets are forwarded to every adjacent router, except the previous-hop router
  - $u$ updates its table of sequence numbers $n_s \leftarrow seq(p)$

# Internet-Level Routing

# Internet-Level Routing

- *Scalability*
  - hundreds of millions of hosts in today’s Internet

# Internet-Level Routing

- *Scalability*
  - hundreds of millions of hosts in today’s Internet
  - transmitting routing information (e.g., LSAs) would be too expensive

# Internet-Level Routing

- *Scalability*
  - hundreds of millions of hosts in today’s Internet
  - transmitting routing information (e.g., LSAs) would be too expensive
  - forwarding would also be too expensive

# Internet-Level Routing

- *Scalability*
  - hundreds of millions of hosts in today's Internet
  - transmitting routing information (e.g., LSAs) would be too expensive
  - forwarding would also be too expensive
- *Administrative autonomy*

# Internet-Level Routing

- *Scalability*
  - hundreds of millions of hosts in today's Internet
  - transmitting routing information (e.g., LSAs) would be too expensive
  - forwarding would also be too expensive
- *Administrative autonomy*
  - one organization might want to run a distance-vector routing protocol, while another might want to run a link-state protocol

# Internet-Level Routing

- *Scalability*
  - hundreds of millions of hosts in today's Internet
  - transmitting routing information (e.g., LSAs) would be too expensive
  - forwarding would also be too expensive

- *Administrative autonomy*
  - one organization might want to run a distance-vector routing protocol, while another might want to run a link-state protocol
  - an organization might not want to expose its internal network structure

# Hierarchical Structure

- Today's Internet is organized in ***autonomous systems (ASs)***
  - independent administrative domains

# Hierarchical Structure

- Today's Internet is organized in ***autonomous systems (ASs)***
  - independent administrative domains

- ***Gateway routers*** connect an autonomous system with other autonomous systems

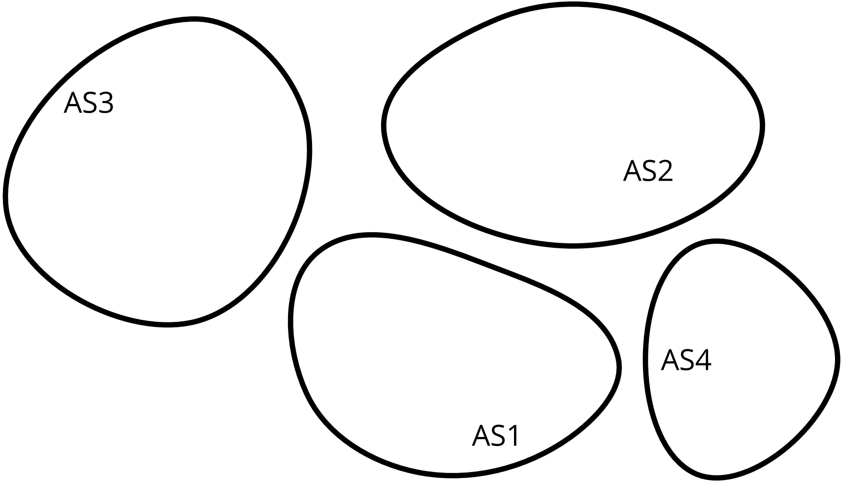
# Hierarchical Structure

- Today's Internet is organized in ***autonomous systems (ASs)***
  - independent administrative domains

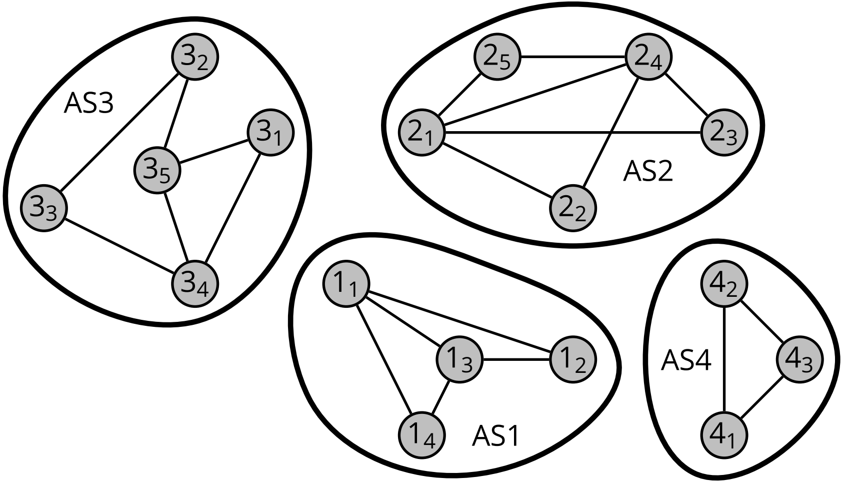
- ***Gateway routers*** connect an autonomous system with other autonomous systems

- An *intra-autonomous system routing protocol* runs within an autonomous system (e.g., OSPF)
  - this protocol determines internal routes
    - internal router ↔ internal router
    - internal router ↔ gateway router
    - gateway router ↔ gateway router

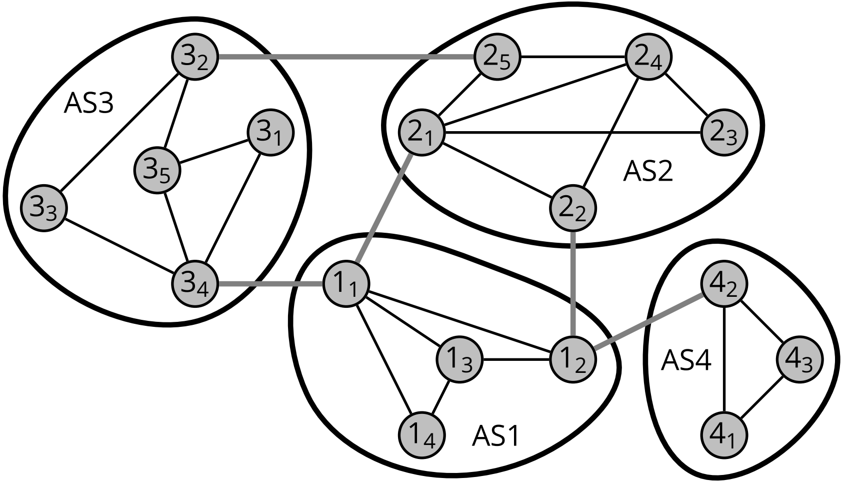
# Hierarchical Structure



# Hierarchical Structure



# Hierarchical Structure

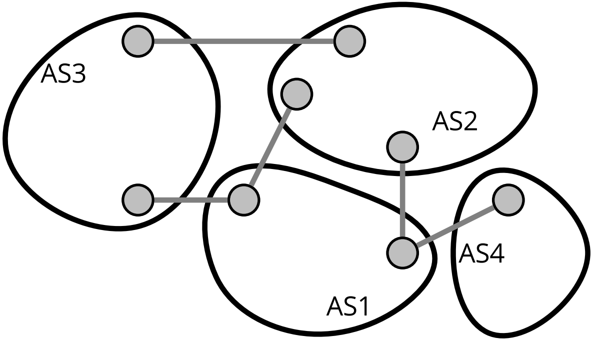


# Inter-AS Routing

- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level

# Inter-AS Routing

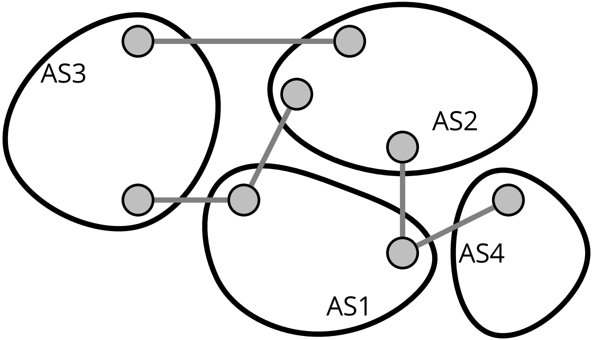
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:
AS1 →

# Inter-AS Routing

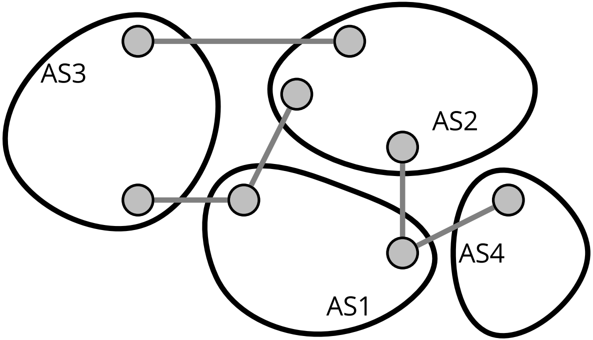
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:
AS1 ⟶ AS1;

# Inter-AS Routing

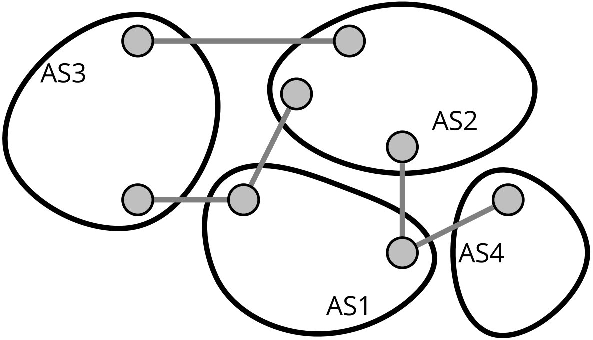
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:
AS1 ⟶ AS1; AS2 ⟶

# Inter-AS Routing

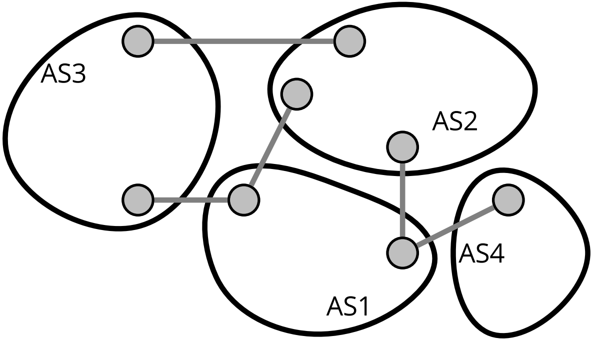
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:
AS1 ⟶ AS1; AS2 ⟶ AS2;

# Inter-AS Routing

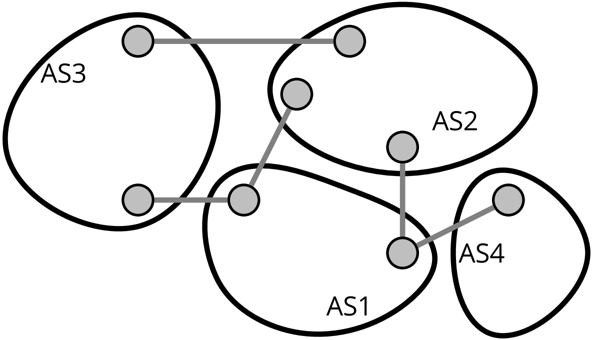
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:
AS1 ⟶ AS1; AS2 ⟶ AS2; AS4 ⟶

# Inter-AS Routing

- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:
AS1 ⟶ AS1; AS2 ⟶ AS2; AS4 ⟶ AS1.

# Hierarchical Routing

- All routers within an AS compute their *intra-AS* routing information
  - using an *intra-doman* routing protocol

# Hierarchical Routing

- All routers within an AS compute their *intra-AS* routing information
  - using an *intra-doman* routing protocol

- Gateway routers figure out *inter-AS* routing information
  - using an *inter-domain* routing protocol

# Hierarchical Routing

- All routers within an AS compute their *intra-AS* routing information
  - using an *intra-doman* routing protocol
- Gateway routers figure out *inter-AS* routing information
  - using an *inter-domain* routing protocol
- *inter-AS* routing information is propagated within an AS
  - using an appropriate protocol

# Hierarchical Routing

- All routers within an AS compute their *intra-AS* routing information
    - using an *intra-doman* routing protocol

- Gateway routers figure out *inter-AS* routing information
    - using an *inter-domain* routing protocol

- *inter-AS* routing information is propagated within an AS
    - using an appropriate protocol

- Both *inter-AS* and *intra-AS* routing information is used to compile the forwarding tables

# Hierarchical Routing (2)

- Destinations within the same autonomous system are reached as usual

# Hierarchical Routing (2)

- Destinations within the same autonomous system are reached as usual
- What about a destination *x* outside the autonomous system?

# Hierarchical Routing (2)

- Destinations within the same autonomous system are reached as usual

- What about a destination *x* outside the autonomous system?
  - *inter-AS* information is used to figure out that $x$ is reachable through gateway $G_x$

# Hierarchical Routing (2)

- Destinations within the same autonomous system are reached as usual
- What about a destination $x$ outside the autonomous system?
  - *inter-AS* information is used to figure out that $x$ is reachable through gateway $G_x$
  - *intra-AS* information is used to figure out how to reach $G_x$ within the AS

# Hierarchical Routing (2)

- Destinations within the same autonomous system are reached as usual
- What about a destination $x$ outside the autonomous system?
    - *inter-AS* information is used to figure out that $x$ is reachable through gateway $G_x$
    - *intra-AS* information is used to figure out how to reach $G_x$ within the AS
    - what if $x$ is reachable through multiple gateway routers $G_x, G'_x, \ldots$?

# Hierarchical Routing (2)

- Destinations within the same autonomous system are reached as usual

- What about a destination $x$ outside the autonomous system?
  - *inter-AS* information is used to figure out that $x$ is reachable through gateway $G_x$
  - *intra-AS* information is used to figure out how to reach $G_x$ within the AS
  - what if $x$ is reachable through multiple gateway routers $G_x, G'_x, \ldots$?
    - use *intra-AS* routing information to determine the costs of the (least-cost) paths to $G_x, G'_x, \ldots$
    - “hot-potato” routing: send it through the closest gateway

# Benefits of Hierarchical Routing

- *Administrative autonomy*

# Benefits of Hierarchical Routing

- *Administrative autonomy*
  - each autonomous system decides what intra-AS routing to use

# Benefits of Hierarchical Routing

- *Administrative autonomy*
  - each autonomous system decides what intra-AS routing to use
  - an autonomous system needs to expose only minimal information about the internal structure of its network
    - essentially only (sub)net addresses

# Benefits of Hierarchical Routing

- *Administrative autonomy*
  - each autonomous system decides what intra-AS routing to use
  - an autonomous system needs to expose only minimal information about the internal structure of its network
    - essentially only (sub)net addresses

- *Scalability*

# Benefits of Hierarchical Routing

- *Administrative autonomy*
  - each autonomous system decides what intra-AS routing to use
  - an autonomous system needs to expose only minimal information about the internal structure of its network
    - essentially only (sub)net addresses

- *Scalability*
  - routers within an autonomous system need to know very little about the internal structure of other autonomous systems

# Benefits of Hierarchical Routing

- *Administrative autonomy*
    - each autonomous system decides what intra-AS routing to use
    - an autonomous system needs to expose only minimal information about the internal structure of its network
        - essentially only (sub)net addresses

- *Scalability*
    - routers within an autonomous system need to know very little about the internal structure of other autonomous systems
        - essentially only (sub)net addresses

# Benefits of Hierarchical Routing

- *Administrative autonomy*
  - each autonomous system decides what intra-AS routing to use
  - an autonomous system needs to expose only minimal information about the internal structure of its network
    - essentially only (sub)net addresses
- *Scalability*
  - routers within an autonomous system need to know very little about the internal structure of other autonomous systems
    - essentially only (sub)net addresses
- External subnet addresses are likely to "aggregate" in groups that admit compact representations
  - this process is called *supernetting*

# Inter-AS Routing in the Internet

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today's Internet

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today's Internet
    - provides reachability information from neighbor ASs

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today’s Internet
  - provides reachability information from neighbor ASs
  - transmits reachability information to all internal routers within an AS

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today's Internet
    - provides reachability information from neighbor ASs
    - transmits reachability information to all internal routers within an AS
    - determines good routes to all outside subnets

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today's Internet
  - provides reachability information from neighbor ASs
  - transmits reachability information to all internal routers within an AS
  - determines good routes to all outside subnets
    - based on reachability information

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today’s Internet
    - provides reachability information from neighbor ASs
    - transmits reachability information to all internal routers within an AS
    - determines good routes to all outside subnets
        - based on reachability information
        - based on *policies*

# Inter-AS Routing in the Internet

- The ***Border Gateway Protocol (BGP)*** is the inter-AS routing protocol in today's Internet
  - provides reachability information from neighbor ASs
  - transmits reachability information to all internal routers within an AS
  - determines good routes to all outside subnets
    - based on reachability information
    - based on *policies*
  - BGP is a *path-vector* protocol

# BGP Architecture and Terminology

- ***BGP session:*** a semi-permanent connection between two routers

# BGP Architecture and Terminology

- ***BGP session:*** a semi-permanent connection between two routers
- ***BGP peers:*** two routers engaged in a BGP session
    - BGP sessions are established over TCP

# BGP Architecture and Terminology

- ***BGP session:*** a semi-permanent connection between two routers
- ***BGP peers:*** two routers engaged in a BGP session
  - BGP sessions are established over TCP
- ***BGP external session (eBGP):*** a session across two autonomous systems

# BGP Architecture and Terminology

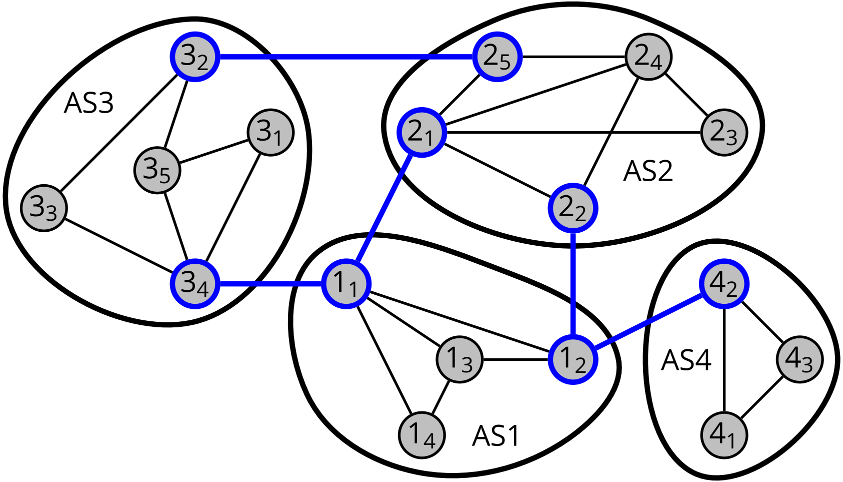
- ***BGP session:*** a semi-permanent connection between two routers

- ***BGP peers:*** two routers engaged in a BGP session
  - BGP sessions are established over TCP

- ***BGP external session (eBGP):*** a session across two autonomous systems

- ***BGP internal session (iBGP):*** a session within an autonomous system
  - note that internal sessions carry *inter-AS* information
  - *intra-AS* routing uses a separate protocol (e.g., OSPF)

# Gateway Routers and *eBGP*



# BGP Architecture and Terminology (2)

- ***BGP advertisement:*** a router advertises routes to networks, much like an entry in a distance-vector
    - destinations are denoted by address *prefixes*

# BGP Architecture and Terminology (2)

- ***BGP advertisement:*** a router advertises routes to networks, much like an entry in a distance-vector
    - destinations are denoted by address *prefixes*
    - an AS may or may not forward an advertisement for a foreign network; doing so means being willing to carry traffic for that network

# BGP Architecture and Terminology (2)

- ***BGP advertisement:*** a router advertises routes to networks, much like an entry in a distance-vector
  - destinations are denoted by address *prefixes*
  - an AS may or may not forward an advertisement for a foreign network; doing so means being willing to carry traffic for that network
  - this is where a router may aggregate prefixes (a.k.a., “supernetting”)
    E.g.,

$$\left.\begin{array}{l}\mathrm{128.138.242.0/24}\\ \mathrm{128.138.243.0/24}\end{array}\right\} \longrightarrow \mathrm{128.138.242.0/23}$$

# BGP Architecture and Terminology (2)

- ***BGP advertisement:*** a router advertises routes to networks, much like an entry in a distance-vector
  - destinations are denoted by address *prefixes*
  - an AS may or may not forward an advertisement for a foreign network; doing so means being willing to carry traffic for that network
  - this is where a router may aggregate prefixes (a.k.a., "supernetting")  
    E.g.,

$$\left.\begin{array}{l}128.138.242.0/24\\128.138.243.0/24\end{array}\right\}\longrightarrow 128.138.242.0/23$$

$$\left.\begin{array}{l}191.224.128.0/22\\191.224.136.0/21\\191.224.132.0/22\end{array}\right\}\longrightarrow$$

# BGP Architecture and Terminology (2)

- ***BGP advertisement:*** a router advertises routes to networks, much like an entry in a distance-vector
  - destinations are denoted by address *prefixes*
  - an AS may or may not forward an advertisement for a foreign network; doing so means being willing to carry traffic for that network
  - this is where a router may aggregate prefixes (a.k.a., “supernetting”)
    E.g.,

$$\left.\begin{array}{l} 128.138.242.0/24 \\ 128.138.243.0/24 \end{array}\right\} \longrightarrow 128.138.242.0/23$$

$$\left.\begin{array}{l} 191.224.128.0/22 \\ 191.224.136.0/21 \\ 191.224.132.0/22 \end{array}\right\} \longrightarrow 191.224.128.0/20$$

# BGP Architecture and Terminology (3)

- ***Autonomous system number (ASN):*** a unique identifier for each AS (with more than one gateway)

# BGP Architecture and Terminology (3)

- ***Autonomous system number (ASN):*** a unique identifier for each AS (with more than one gateway)
- ***BGP attributes:*** a route advertisement includes a number of attributes
  - *AS-PATH:* sequence of ASNs through which the advertisement has been sent

# BGP Architecture and Terminology (3)

- ***Autonomous system number (ASN):*** a unique identifier for each AS (with more than one gateway)

- ***BGP attributes:*** a route advertisement includes a number of attributes
  - *AS-PATH:* sequence of ASNs through which the advertisement has been sent
  - *NEXT-HOP:* specifies the interface (IP address) to use to forward packets towards the advertised destination
    - used to resolve ambiguous cases where an AS can be reached through multiple gateways (interfaces)

# BGP Architecture and Terminology (3)

- ***Autonomous system number (ASN):*** a unique identifier for each AS (with more than one gateway)
- ***BGP attributes:*** a route advertisement includes a number of attributes
  - *AS-PATH:* sequence of ASNs through which the advertisement has been sent
  - *NEXT-HOP:* specifies the interface (IP address) to use to forward packets towards the advertised destination
    - used to resolve ambiguous cases where an AS can be reached through multiple gateways (interfaces)
- ***BGP import policy:*** used to decide whether to accept or reject the route advertisement
  - e.g., a router may not want to send its traffic through one of the AS listed in *AS-PATH*

# Route Selection

1. Router preference: routes are ranked according to a *preference* value
    - configured at the router
    - or learned from another router within the same AS
    - essentially a configuration parameter for the AS

# Route Selection

1. Router preference: routes are ranked according to a *preference* value
    - configured at the router
    - or learned from another router within the same AS
    - essentially a configuration parameter for the AS

2. Shortest AS-PATH

# Route Selection

1. Router preference: routes are ranked according to a *preference* value
    - configured at the router
    - or learned from another router within the same AS
    - essentially a configuration parameter for the AS

2. Shortest AS-PATH

3. Closest NEXT-HOP router

# Route Selection

1. Router preference: routes are ranked according to a *preference* value
    - configured at the router
    - or learned from another router within the same AS
    - essentially a configuration parameter for the AS

2. Shortest AS-PATH

3. Closest NEXT-HOP router

4. . . .