

A Quick Review of Computer Networking

Architecture, Applications, Transport (TCP), Routing

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

February 17, 2025

What is the Internet?

What is the Internet?

Chapo Trap House (@CHAPOTRAPHOUSE)
3,810 Tweets

Chapo Trap House
@CHAPOTRAPHOUSE

Brooklyn, NY chapotra

305 Following 164.2K Followers

Don't miss what's happen
People on Twitter are the first to know

WIKIPEDIA
The Free Encyclopedia

Noam Chomsky

From Wikipedia, the free encyclopedia
(Redirected from Noam chomsky)

"Chomsky" redirects here. For other uses, see *Chomsky (disambiguation)*.

Avram Noam Chomsky^[a] (born December 7, 1928) is an American linguist, philosopher, cognitive scientist, historian,^[b]^[c] social critic, and political activist. Sometimes called "the father of modern linguistics",^[d] Chomsky is also a major figure in analytic philosophy and one of the founders of the field of cognitive science. He is Laureate Professor of Linguistics at the University of Arizona and Institute Professor Emeritus at the Massachusetts Institute of Technology (MIT), and is the author of more than 150 books on topics such as linguistics, war, politics, and mass media. Ideologically, he aligns with anarcho-syndicalism and libertarian socialism.

Born to Jewish immigrants in Philadelphia, Chomsky developed an early interest in anarchism from alternative bookstores in New York City. He studied at the University of Pennsylvania. During his postgraduate work in the Harvard

Numberphile - YouTube

Numberphile

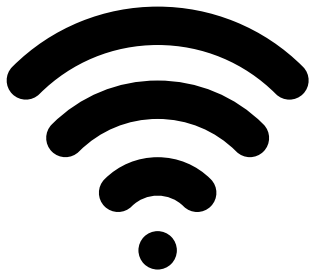
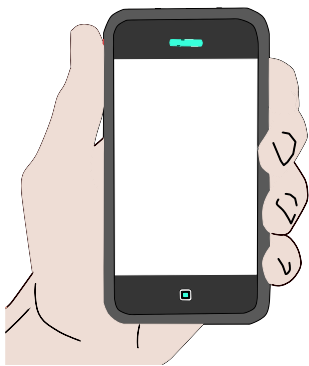
3.7M subscribers

Singing a Pig - Numberphile
146K views • 3 weeks ago

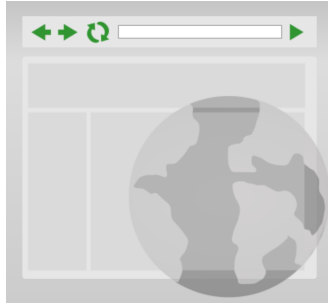
Eureka Sequences - Numberphile
155K views • 1 month ago

The Levine Sequence - Numberphile
189K views • 1 month ago

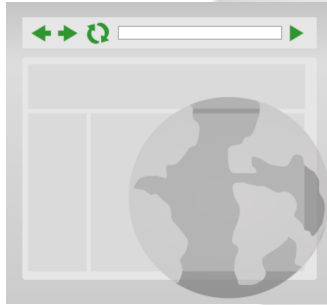
What is the Internet?



What is the Internet?

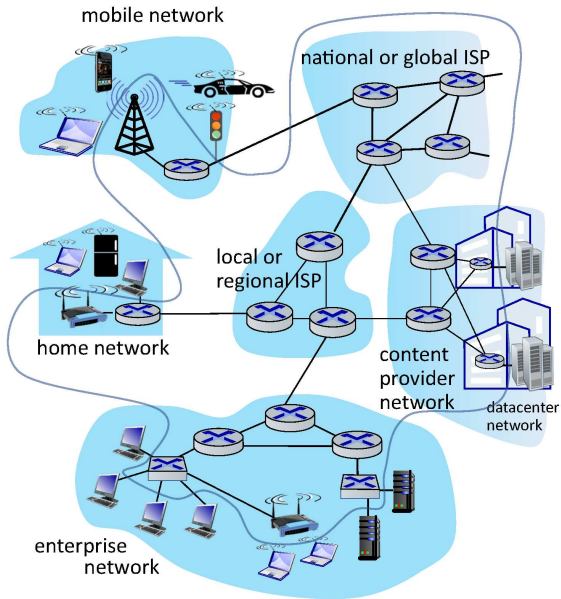


What is the Internet?

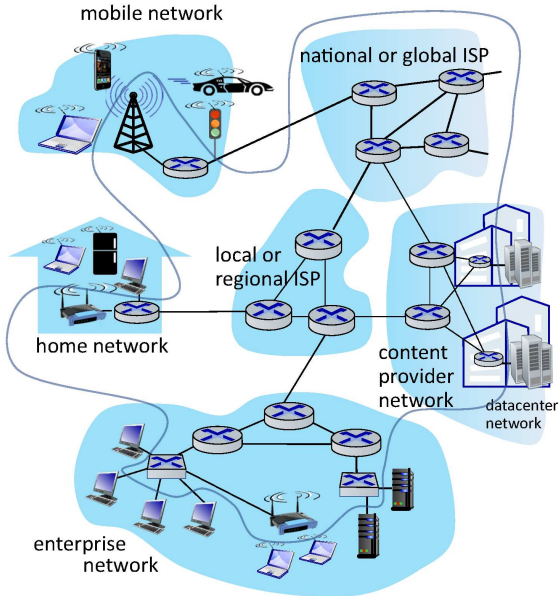


What's *Inside*?

What's Inside?

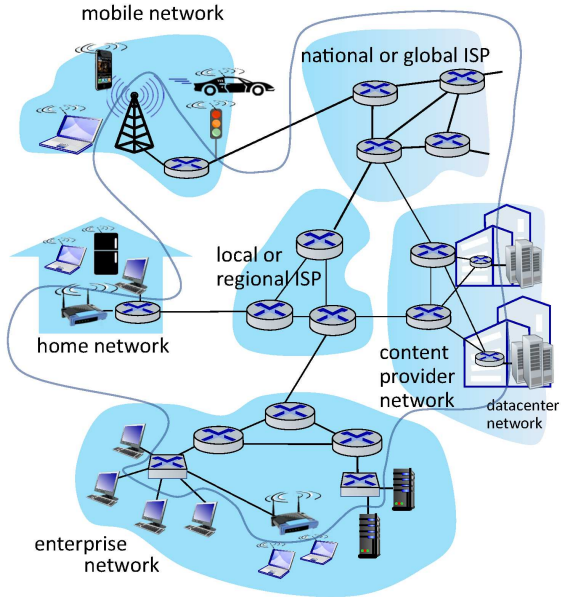


What's Inside?



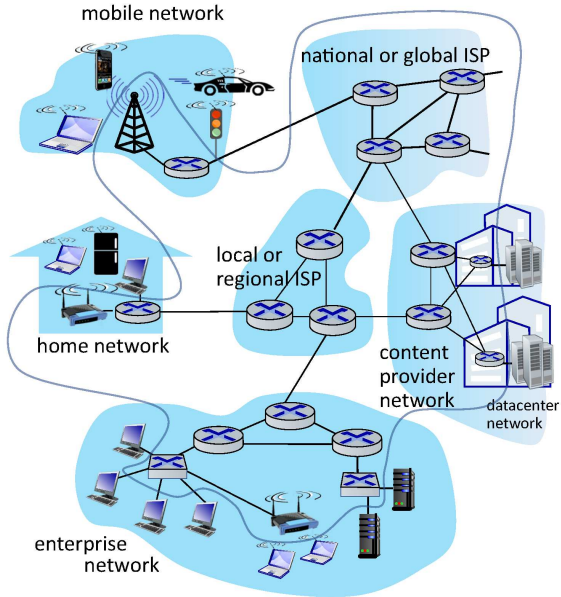
- Billions of connected devices
 - ▶ “host” or “end system”
 - ▶ run network applications at the “edge”

What's Inside?



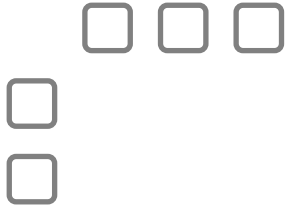
- Billions of connected devices
 - ▶ “host” or “end system”
 - ▶ run network applications at the “edge”
- Links
 - ▶ wireless: radio, satellite
 - ▶ wired: copper, optic fiber

What's Inside?

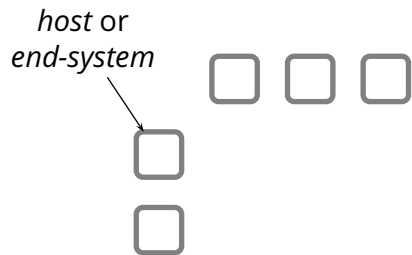


- Billions of connected devices
 - ▶ “host” or “end system”
 - ▶ run network applications at the “edge”
- Links
 - ▶ wireless: radio, satellite
 - ▶ wired: copper, optic fiber
- Router, switch
 - ▶ forward “packets”
 - ▶ routing

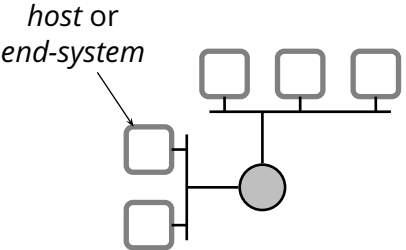
A Schematic View



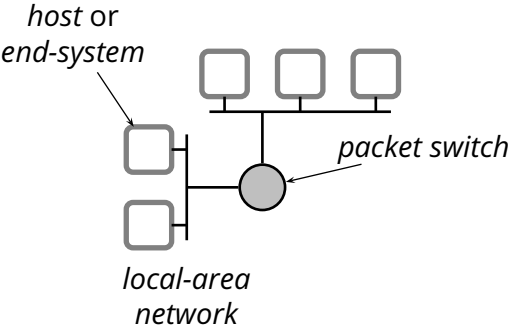
A Schematic View



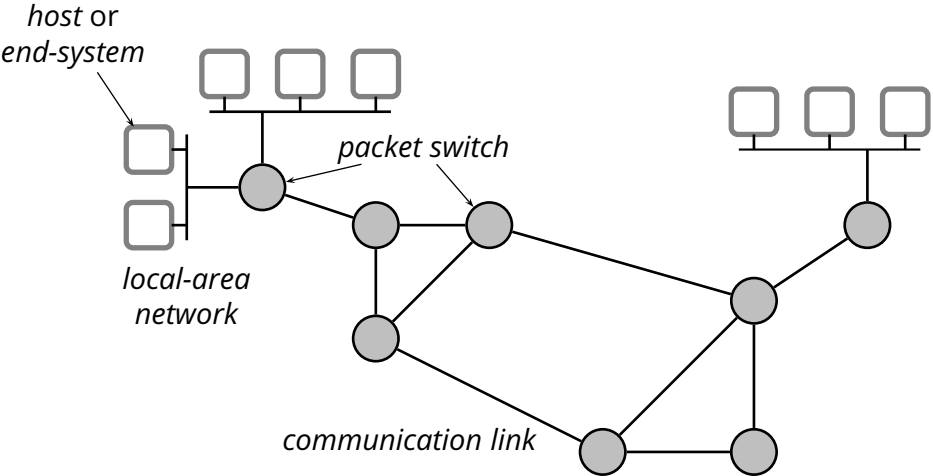
A Schematic View



A Schematic View



A Schematic View



- The Internet uses *packet switching*

- The Internet uses *packet switching*
- *Packet switch*: a *link-layer switch* or a *router*

- 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

- 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*)

- 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

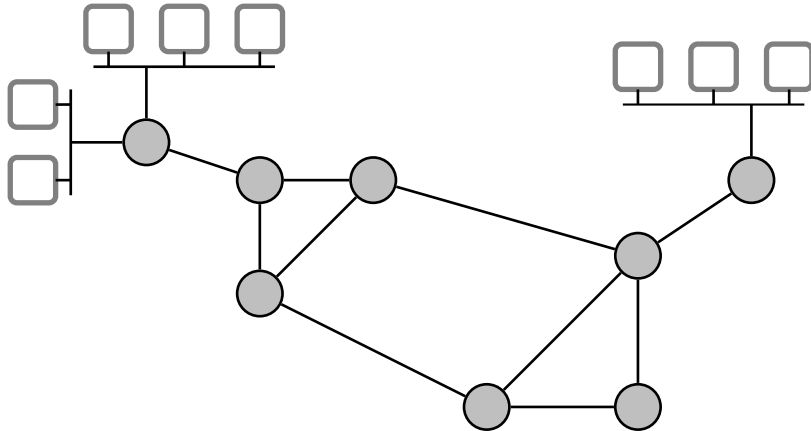
- Various types and forms of medium

- 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
 - ▶ ...

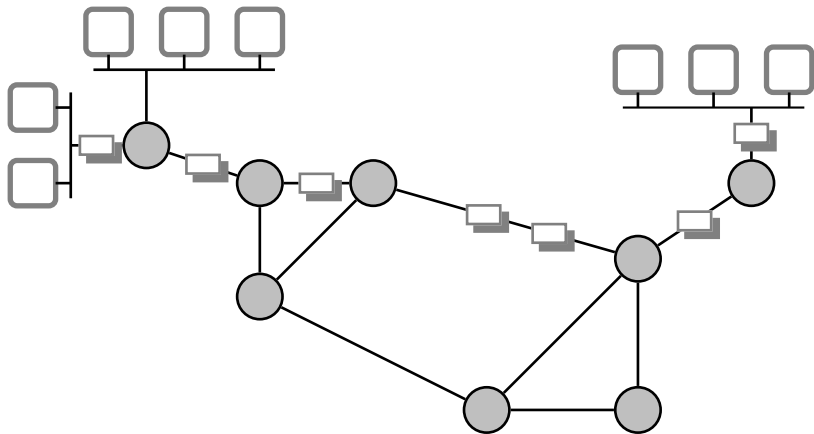
Part I

Network Architecture

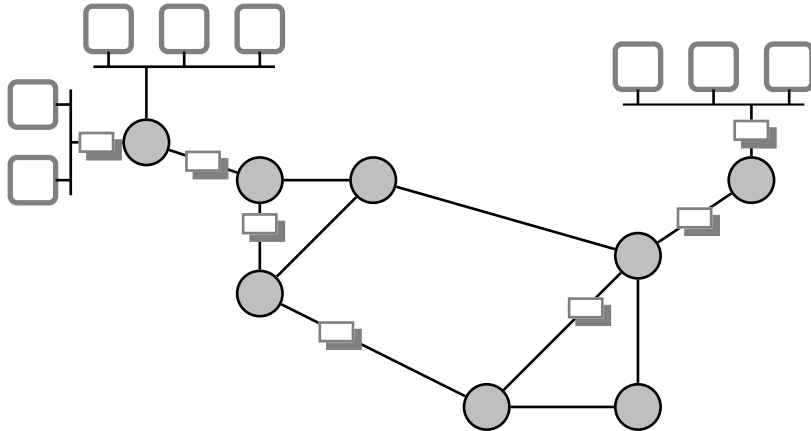
Packet Switching



Packet Switching



Packet Switching



- The Internet is a *packet-switched* network

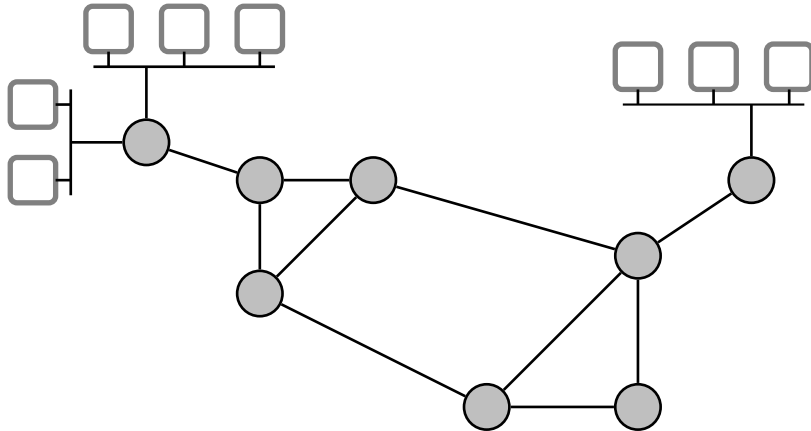
- The Internet is a *packet-switched* network
- Information is transmitted in *packets*

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

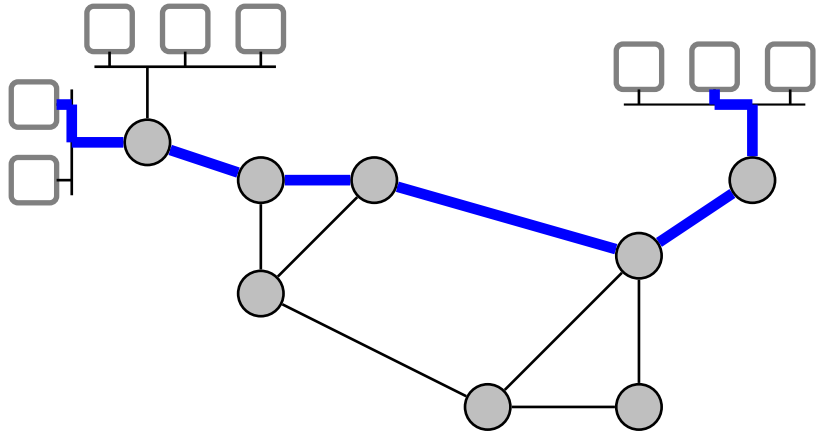
- The Internet is a *packet-switched* network
- Information is transmitted in *packets*
- Switches operate on individual packets
- A switch (router) receives packets and *forwards* them along to other switches or to end systems

- 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



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

- 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.)

- 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

- 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
 - ▶ the advantage is fundamental, as we will see in studying queuing theory

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

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

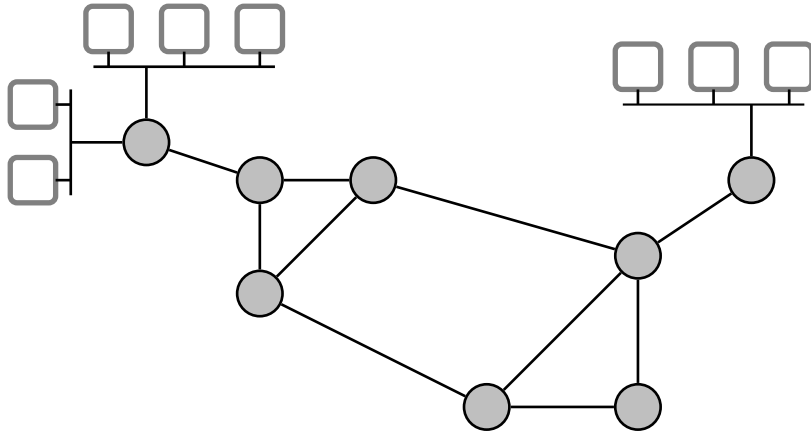
- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase
- The connection does not create a physical circuit, but rather a “virtual circuit”

- 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

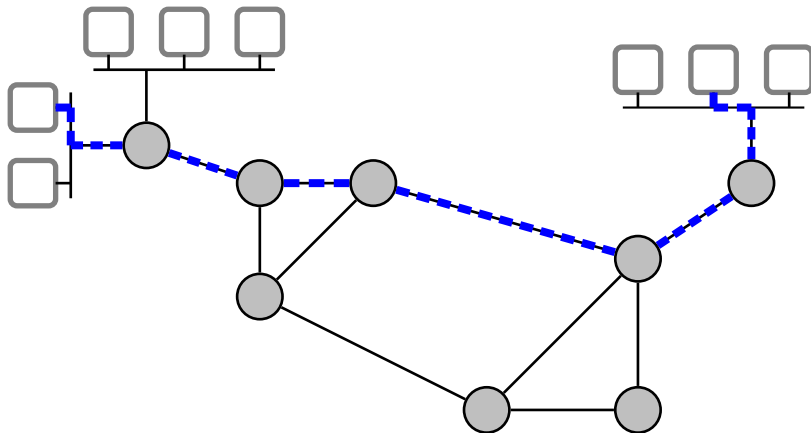
- Idea: combine the advantages of circuit switching and packet switching
- There is a connection setup phase
- The connection does not create a physical circuit, but rather a “virtual circuit”
- Information is sent in packets, so links can be shared more effectively
- Packets carry a *virtual circuit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

- 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

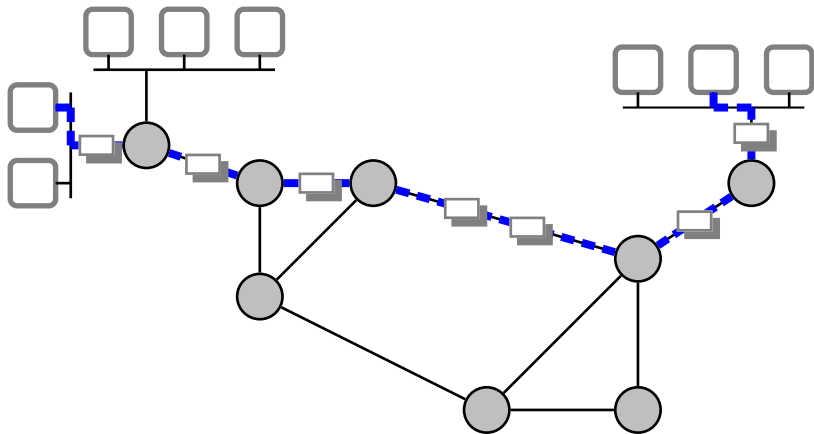
Virtual Circuit



Virtual Circuit



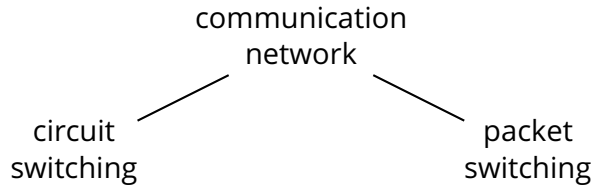
Virtual Circuit



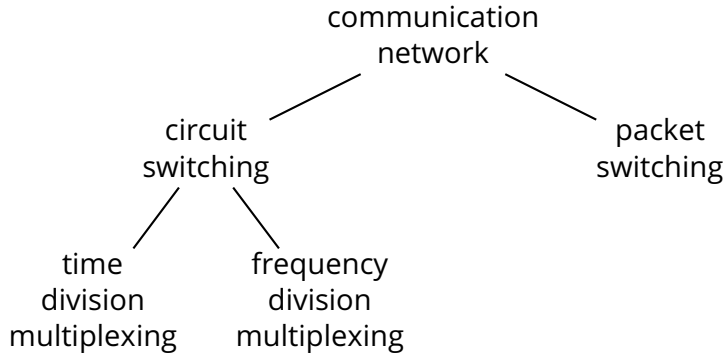
Taxonomy of Networks

communication
network

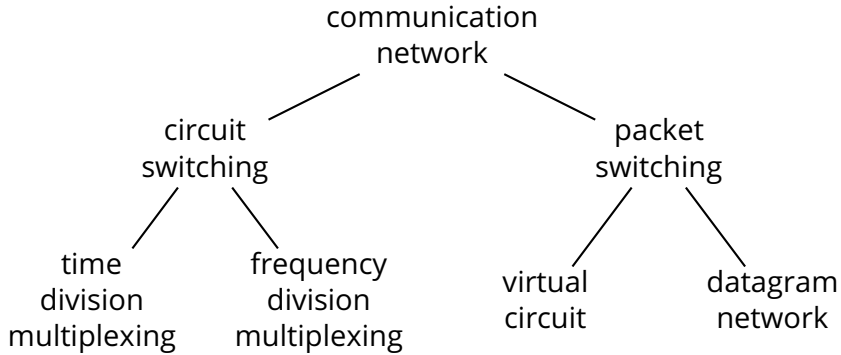
Taxonomy of Networks



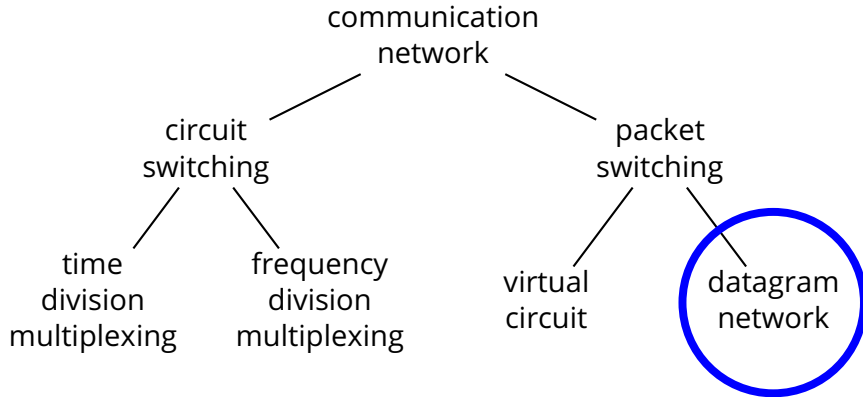
Taxonomy of Networks



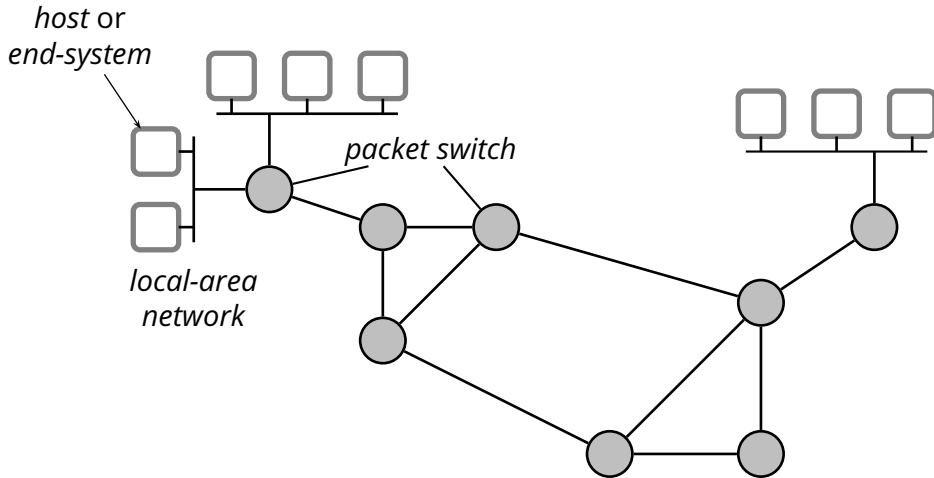
Taxonomy of Networks



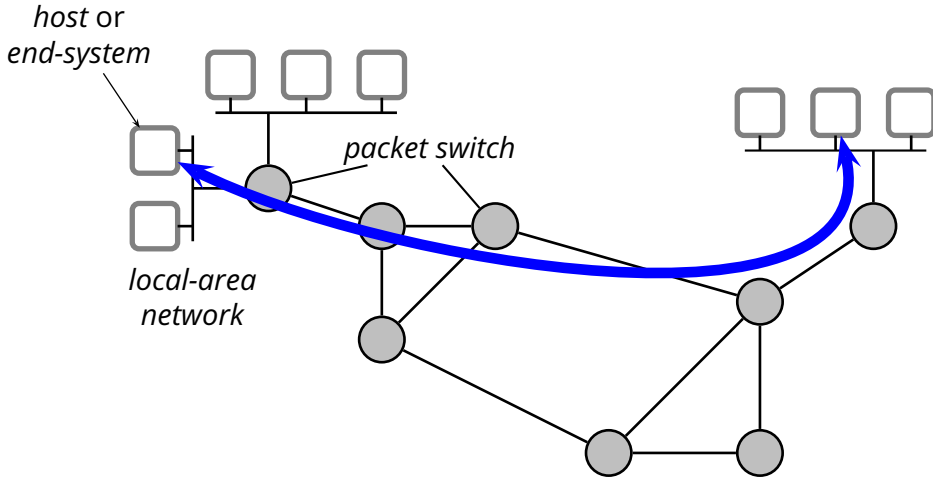
Taxonomy of Networks



Service Perspective



Service Perspective



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

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

- 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

- 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

- How reliable is a “reliable” service?

- 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

- 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)

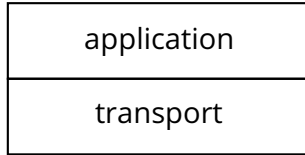
Internet Protocol Stack

Internet Protocol Stack

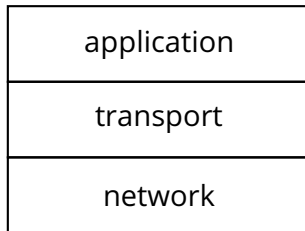


application

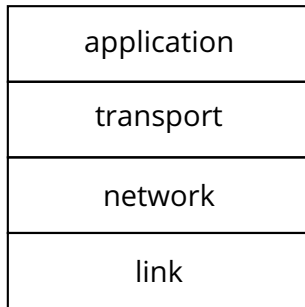
Internet Protocol Stack



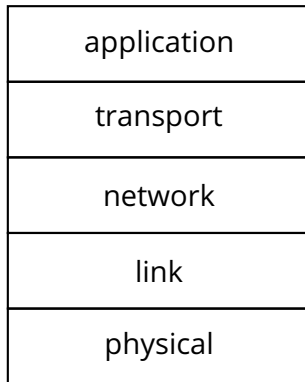
Internet Protocol Stack



Internet Protocol Stack



Internet Protocol Stack



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

- *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)

- *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)

- *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)

- *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*

Part II

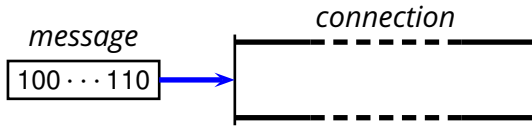
Delay and Throughput

Delay (Latency) and Rate (Throughput)

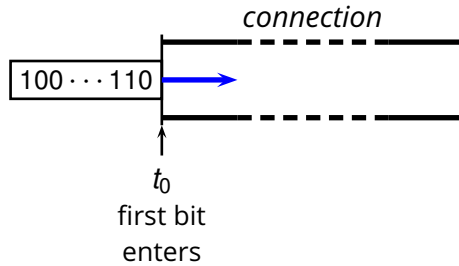
connection



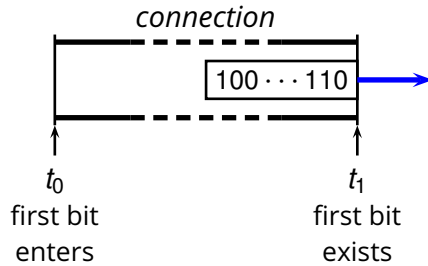
Delay (Latency) and Rate (Throughput)



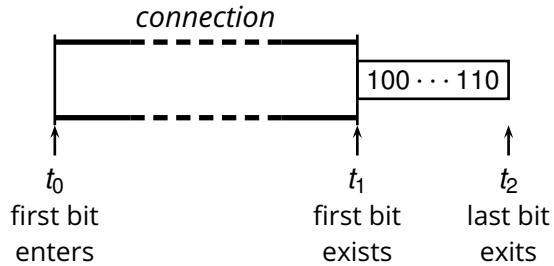
Delay (Latency) and Rate (Throughput)



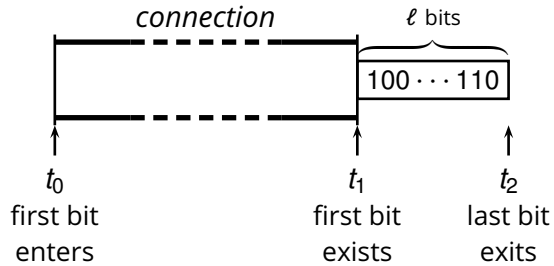
Delay (Latency) and Rate (Throughput)



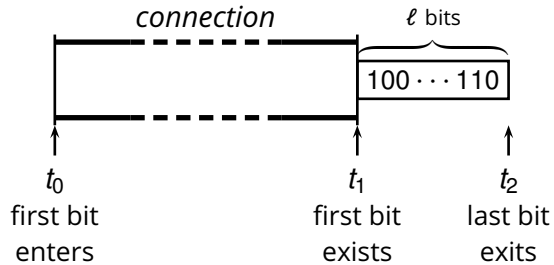
Delay (Latency) and Rate (Throughput)



Delay (Latency) and Rate (Throughput)



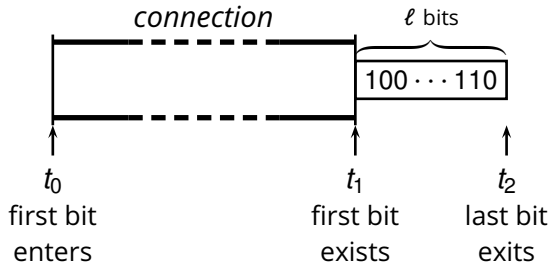
Delay (Latency) and Rate (Throughput)



Propagation **Delay**

$$d_{prop} = t_1 - t_0 \quad \text{sec}$$

Delay (Latency) and Rate (Throughput)



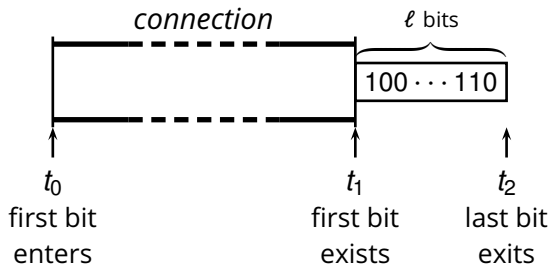
Propagation **Delay**

$$d_{prop} = t_1 - t_0 \quad \text{sec}$$

Transmission **Rate**

$$R = \frac{l}{t_2 - t_1} \quad \text{bits/sec}$$

Delay (Latency) and Rate (Throughput)



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

Transmission Rate $R = \frac{l}{t_2 - t_1}$ bits/sec

Total transfer time $d_{end-end} = d + \frac{l}{R}$ sec

Hosts A and B are connected through a link with propagation delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$.

Hosts A and B are connected through a link with propagation delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$.

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?

Hosts A and B are connected through a link with propagation delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$.

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?
- **Question 2:** How long does it take for host A to transmit $S = 1\text{MB}$ of data if the usable payload is $MSS = 1400\text{B}$?

Hosts A and B are connected through a link with propagation delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$.

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?
- **Question 2:** How long does it take for host A to transmit $S = 1\text{MB}$ of data if the usable payload is $MSS = 1400\text{B}$?
- **Question 3:** What is the total transfer time for host B to receive $S = 1\text{MB}$ of data if the usable payload is $MSS = 1400\text{B}$?

Hosts A and B are connected through a link with propagation delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$.

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?
- **Question 2:** How long does it take for host A to transmit $S = 1\text{MB}$ of data if the usable payload is $MSS = 1400\text{B}$?
- **Question 3:** What is the total transfer time for host B to receive $S = 1\text{MB}$ of data if the usable payload is $MSS = 1400\text{B}$?
- **Question 4:** What is the total transfer time for host B to receive $S = 2\text{KB}$ of data if the usable payload is $MSS = 1400\text{B}$?

Store-And-Forward Delay

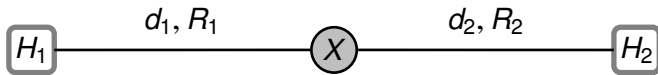
Store-And-Forward Delay

H_1

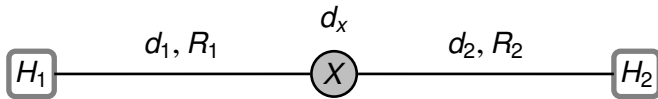
X

H_2

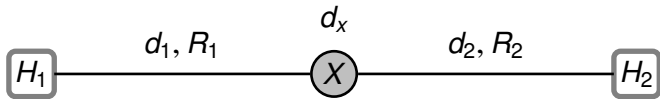
Store-And-Forward Delay



Store-And-Forward Delay

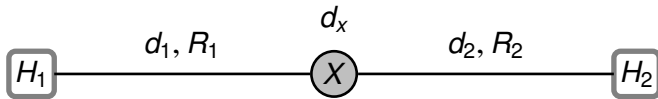


Store-And-Forward Delay



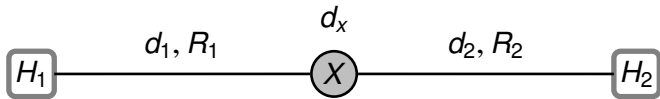
$$d_{end-end} = d_1 + \frac{\ell}{R_1}$$

Store-And-Forward Delay



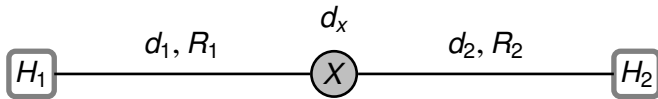
$$d_{end-end} = d_1 + \frac{\ell}{R_1} + d_x$$

Store-And-Forward Delay



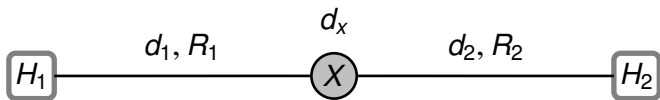
$$d_{end-end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2}$$

Store-And-Forward Delay

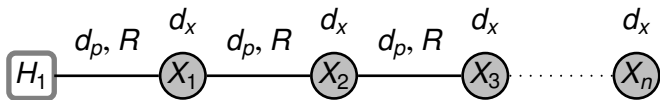


$$d_{end-end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2} + d_2$$

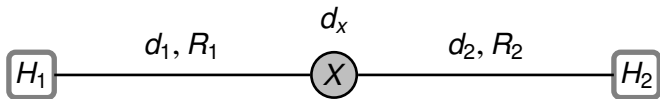
Store-And-Forward Delay



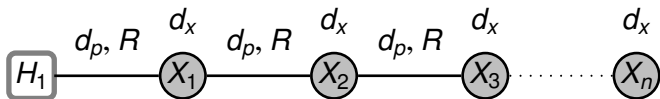
$$d_{end-end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2} + d_2$$



Store-And-Forward Delay



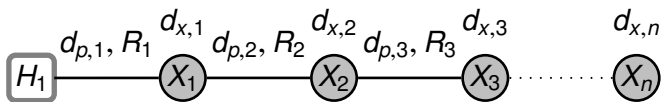
$$d_{end-end} = d_1 + \frac{\ell}{R_1} + d_x + \frac{\ell}{R_2} + d_2$$



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

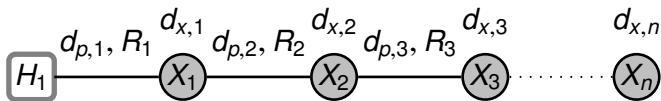
Store-And-Forward Delay

Store-And-Forward Delay



$$d_{end-end} = n \left(\overline{d_p} + \overline{\tau} \ell + \overline{d_x} \right)$$

Store-And-Forward Delay



$$d_{end-end} = n \left(\overline{d_p} + \overline{\tau} \ell + \overline{d_x} \right)$$

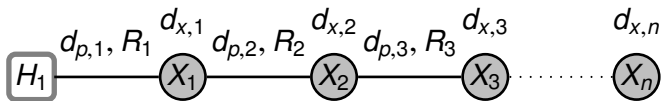
where

$$\overline{d_p} = \text{avg}\{d_{p,i}\}$$

$$\overline{\tau} = \text{avg} \left\{ \frac{1}{R_i} \right\}$$

$$\overline{d_x} = \text{avg}\{d_{x,i}\}$$

Store-And-Forward Delay



$$d_{end-end} = n \left(\overline{d_p} + \overline{\tau} \ell + \overline{d_x} \right)$$

where

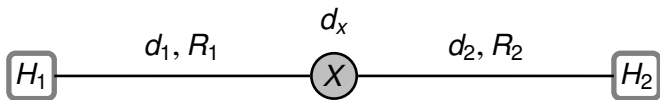
$$\overline{d_p} = \text{avg}\{d_{p,i}\}$$

$$\overline{\tau} = \text{avg} \left\{ \frac{1}{R_i} \right\}$$

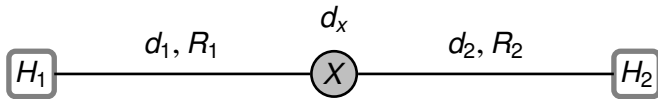
$$\overline{d_x} = \text{avg}\{d_{x,i}\}$$

End-to-End Throughput

End-to-End Throughput

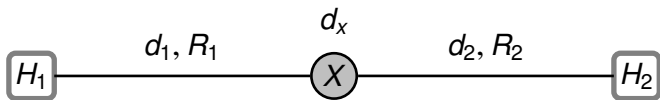


End-to-End Throughput



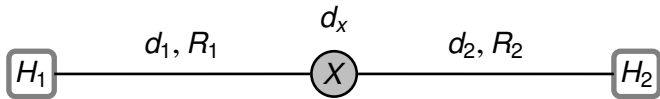
$$R_{end-end} =$$

End-to-End Throughput

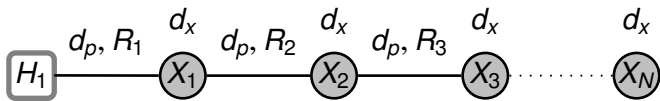


$$R_{end-end} = \min\{R_1, R_2\}$$

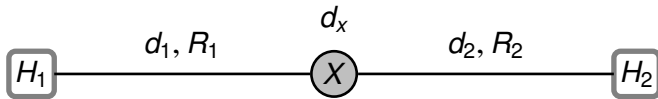
End-to-End Throughput



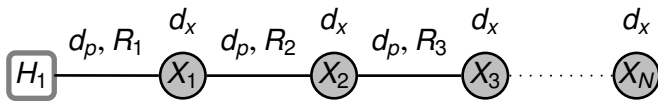
$$R_{end-end} = \min\{R_1, R_2\}$$



End-to-End Throughput



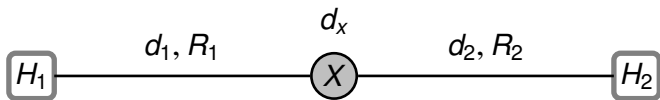
$$R_{end-end} = \min\{R_1, R_2\}$$



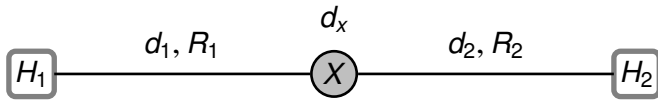
$$R_{end-end} = \min\{R_1, R_2, \dots, R_N\}$$

Processing and Queuing Delays

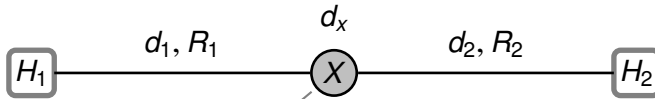
Processing and Queuing Delays



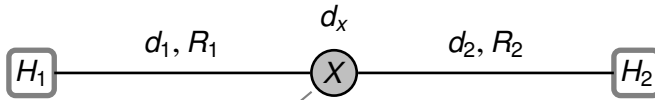
Processing and Queuing Delays



Processing and Queuing Delays



Processing and Queuing Delays



queue length

$$d_x = d_{cpu} + d_{queue}$$
$$d_{queue} = |q| / R_x$$

output rate



where


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

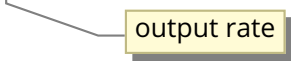
queue length

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

output rate

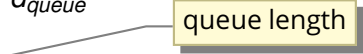
where

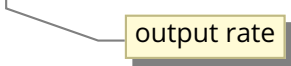
$$d_x = d_{cpu} + d_{queue}$$


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


R_x is the transmission rate, which means that it is also the the rate at which packets get out of the queue

where

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


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


R_x is the transmission rate, which means that it is also the rate at which packets get out of the queue

$$d_{end-end} = n \left(\overline{d_p} + \overline{\tau} \ell + \overline{d_{cpu}} + \text{avg} \left\{ \frac{|q_i|}{R_i} \right\} \right)$$

Hosts A and B are connected through a path of $n = 10$ hops, each with delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$. Each router along the path has $Q = 20\text{MB}$ buffers (queues).

Hosts A and B are connected through a path of $n = 10$ hops, each with delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$. Each router along the path has $Q = 20\text{MB}$ buffers (queues).

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?

Hosts A and B are connected through a path of $n = 10$ hops, each with delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$. Each router along the path has $Q = 20\text{MB}$ buffers (queues).

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?
- **Question 2:** What is the total transfer time for $S = 1\text{MB}$ of data from A to B if the usable payload is $MSS = 1400\text{B}$ in the best case?

Hosts A and B are connected through a path of $n = 10$ hops, each with delay $d_p = 10\text{ms}$ and transmission rate $R = 1\text{Gb/s}$. The maximum packet size is $MTU = 1500\text{B}$. Each router along the path has $Q = 20\text{MB}$ buffers (queues).

- **Question 1:** How long does it take for host A to transmit $S = 1\text{MB}$ of data?
- **Question 2:** What is the total transfer time for $S = 1\text{MB}$ of data from A to B if the usable payload is $MSS = 1400\text{B}$ in the best case?
- **Question 3:** What is the total transfer time for $S = 1\text{MB}$ of data from A to B if the usable payload is $MSS = 1400\text{B}$ in the worst case?

Part III

Application Protocols

HTTP

HTTP

SMTP

Application Protocols

HTTP

SMTP

DNS

Application Protocols

HTTP

SMTP

DNS

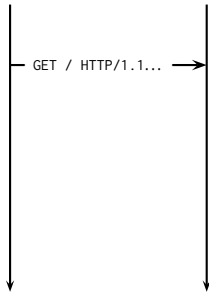


Application Protocols

HTTP

SMTP

DNS

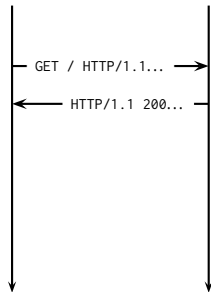


Application Protocols

HTTP

SMTP

DNS

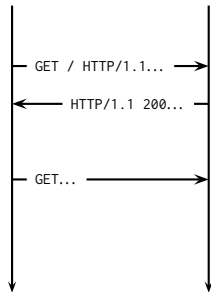


Application Protocols

HTTP

SMTP

DNS

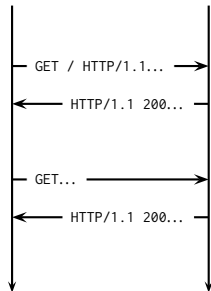


Application Protocols

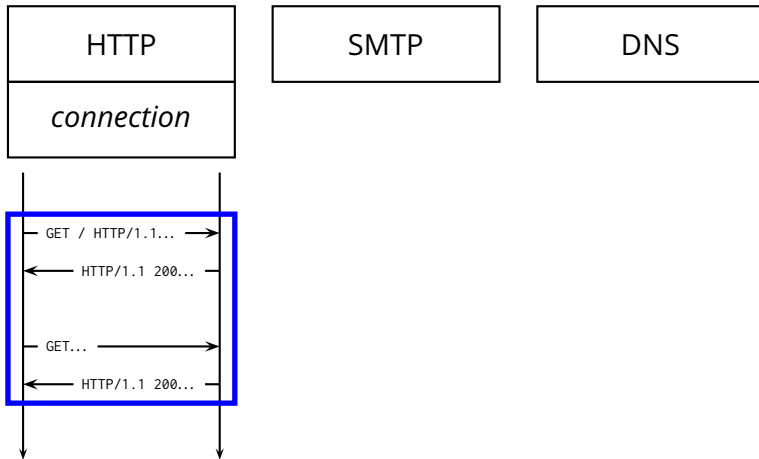
HTTP

SMTP

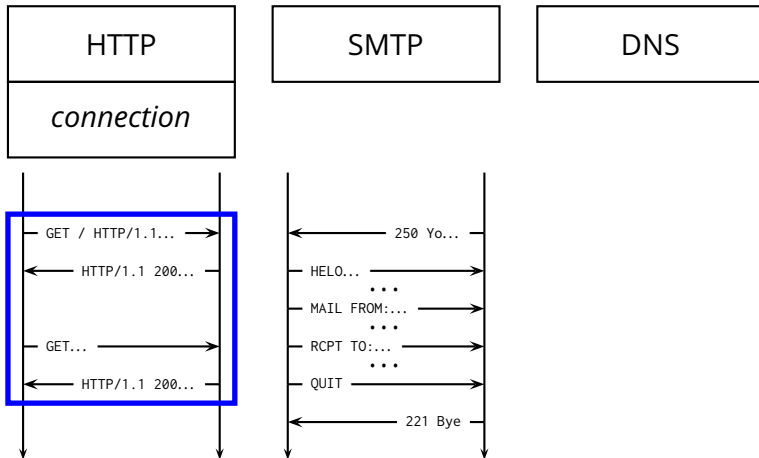
DNS



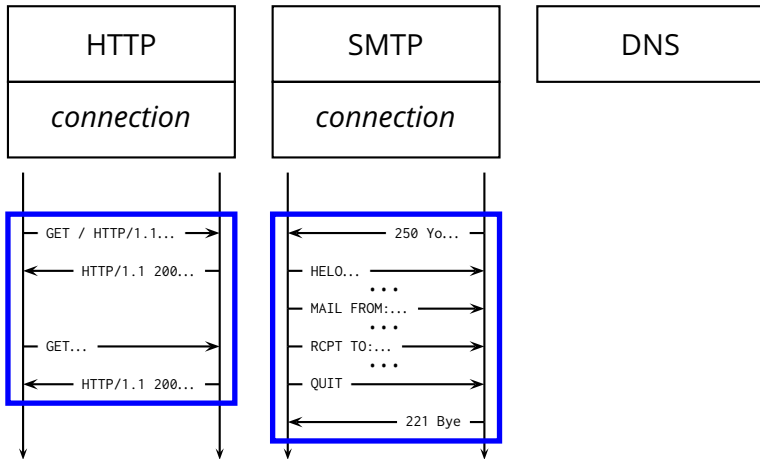
Application Protocols



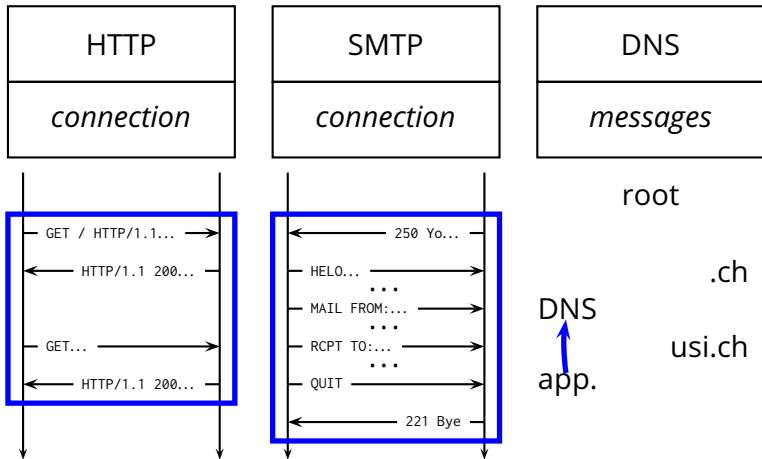
Application Protocols



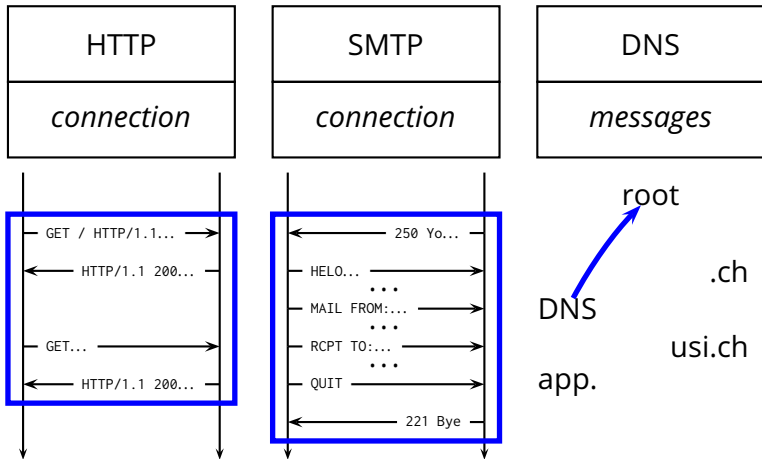
Application Protocols



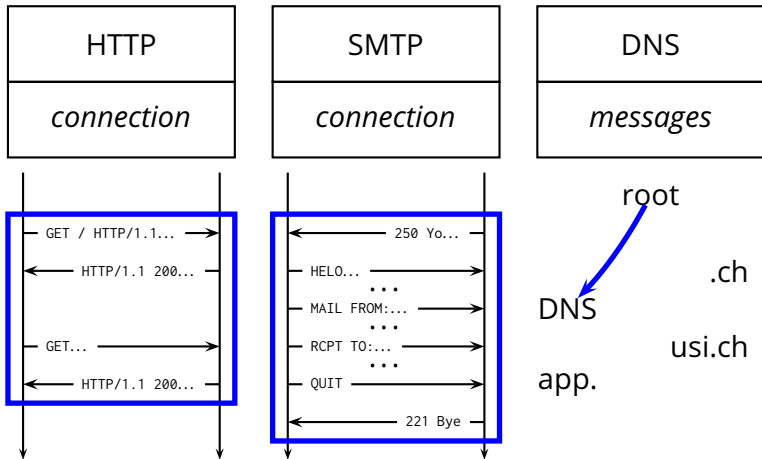
Application Protocols



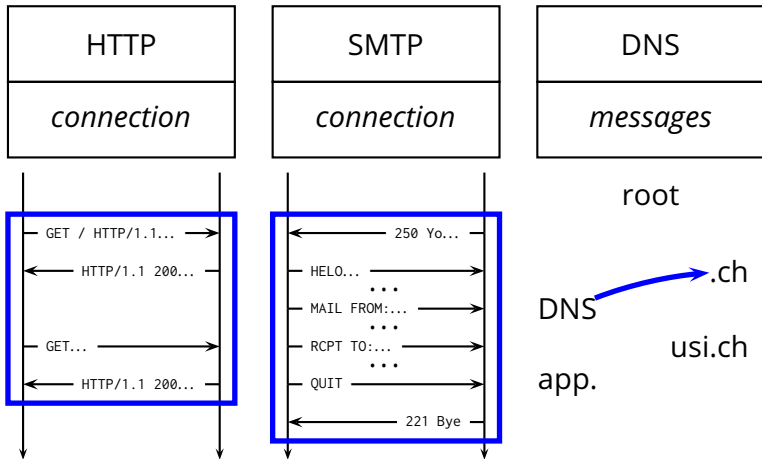
Application Protocols



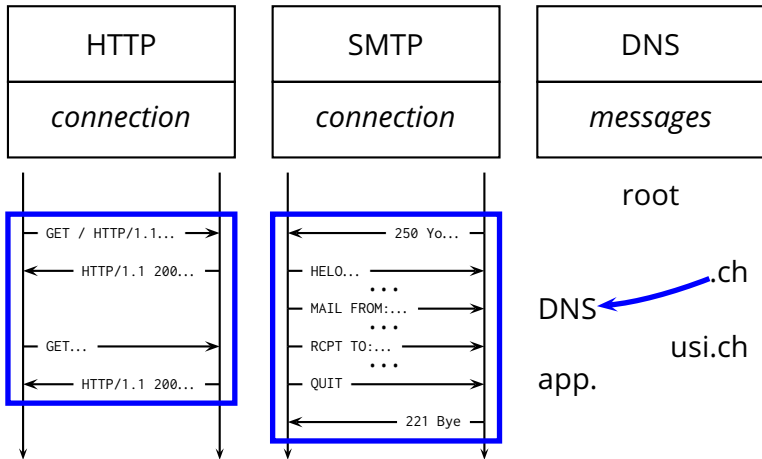
Application Protocols



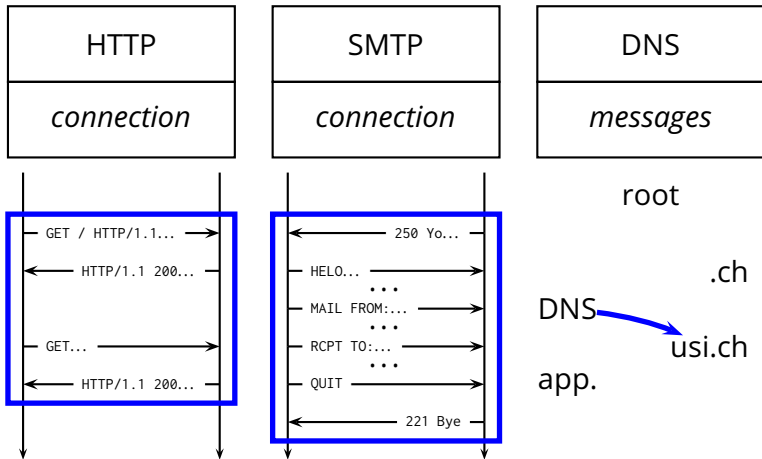
Application Protocols



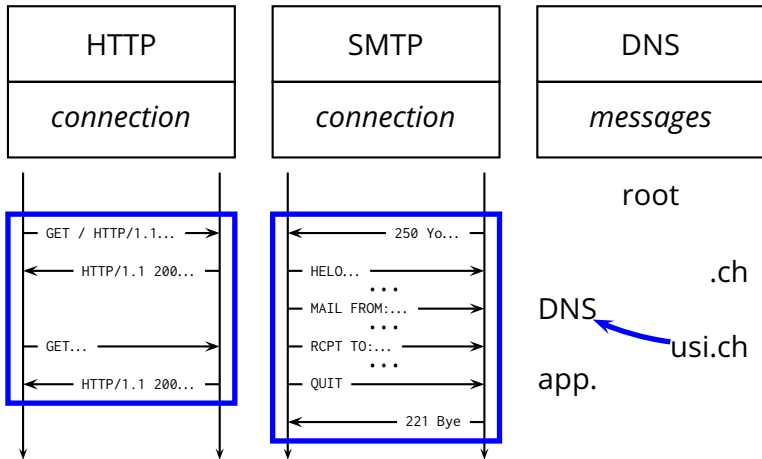
Application Protocols



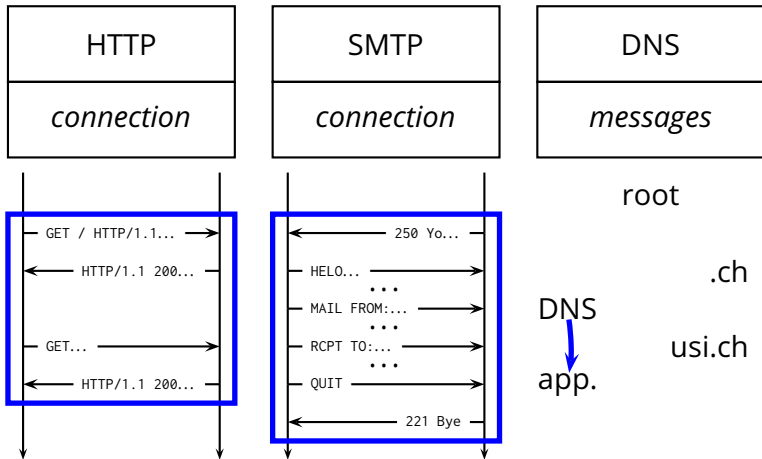
Application Protocols



Application Protocols



Application Protocols



Part IV

Application Multiplexing

Transport Layer in the Internet

- ***Transport Control Protocol (TCP)***

- ▶ connection-oriented (i.e., “connections”)

Transport Layer in the Internet

■ *Transport Control Protocol (TCP)*

- ▶ connection-oriented (i.e., “connections”)

■ *User Datagram Protocol (UDP)*

- ▶ connectionless (i.e., “messages”)

■ *Transport Control Protocol (TCP)*

- ▶ connection-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)***

- ▶ connection-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 Control Protocol (TCP)***

- ▶ connection-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 Control Protocol (TCP)***

- ▶ connection-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

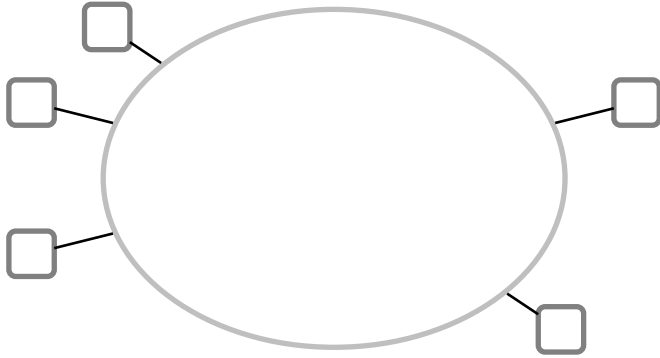
■ *Connections*

- ▶ i.e., streams
- ▶ can be seen as the same as ordered delivery

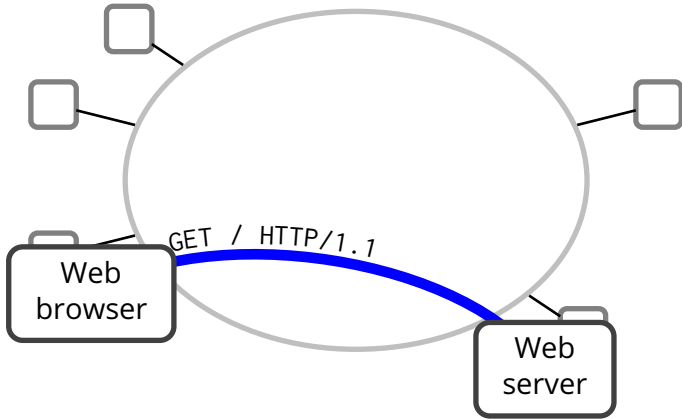
■ *Congestion control*

- ▶ i.e., end-to-end traffic (admission) control so as to avoid destructive congestions within the network

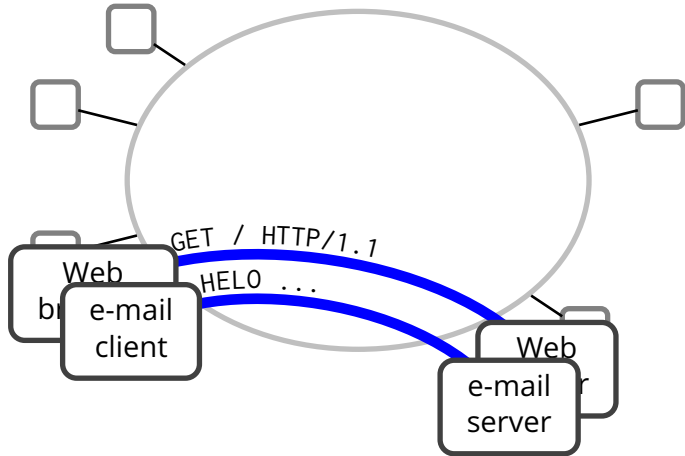
Multiplexing/Demultiplexing



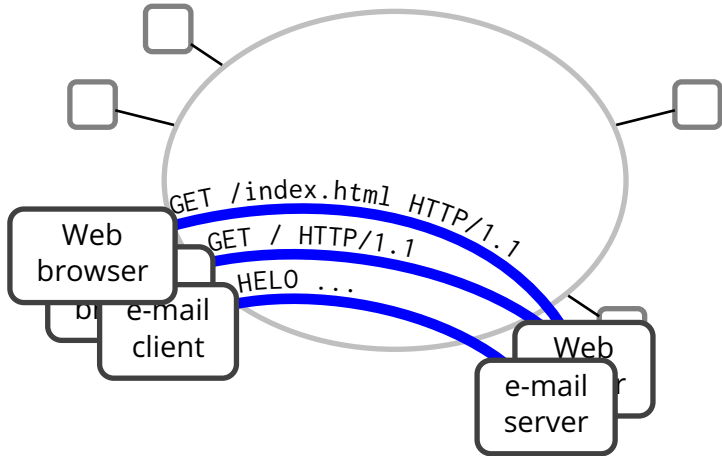
Multiplexing/Demultiplexing



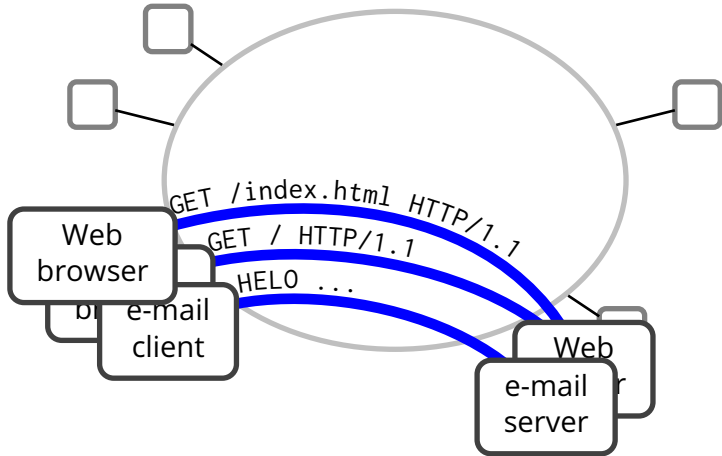
Multiplexing/Demultiplexing



Multiplexing/Demultiplexing

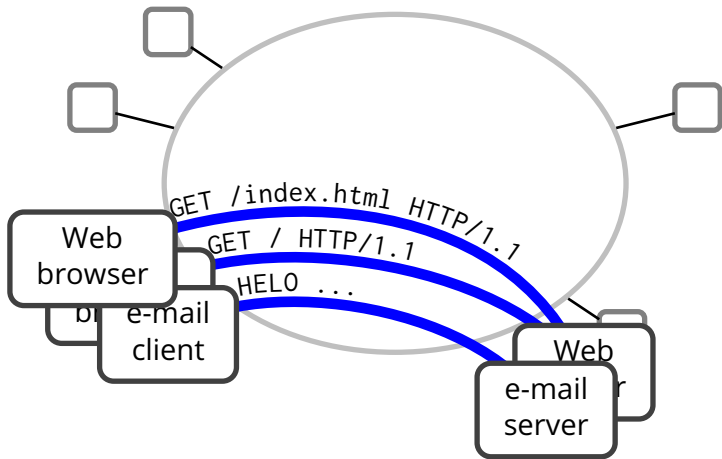


Multiplexing/Demultiplexing



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)

- Each connection from host A to host B is identified by two **port numbers** P_A and P_B

- Each connection from host A to host B is identified by two **port numbers** P_A and P_B
- Thus a “connection” is identified by two pairs of *host* and *port* identifiers

$$(IP\ address, port)_A \longleftrightarrow (IP\ address, port)_B$$

- Each connection from host A to host B is identified by two **port numbers** P_A and P_B
- Thus a “connection” is identified by two pairs of *host* and *port* identifiers

$$(IP\ address, port)_A \longleftrightarrow (IP\ address, port)_B$$

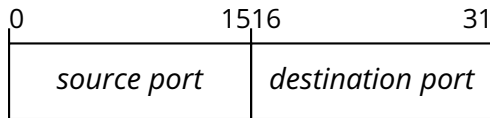
- How do we find out which application (host and port number) to connect to?

- Each connection from host A to host B is identified by two **port numbers** P_A and P_B
- Thus a “connection” is identified by two pairs of *host* and *port* identifiers

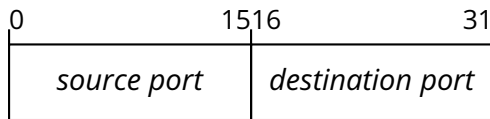
$$(IP\ address, port)_A \longleftrightarrow (IP\ address, port)_B$$

- 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

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

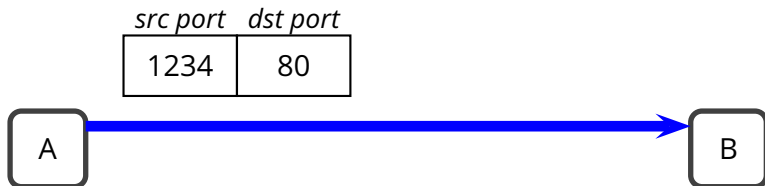


- The message format of both UDP and TCP starts with the source and destination port numbers

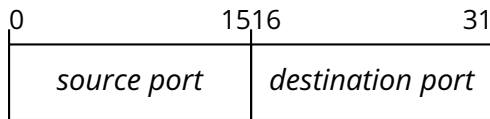


...

- E.g.,

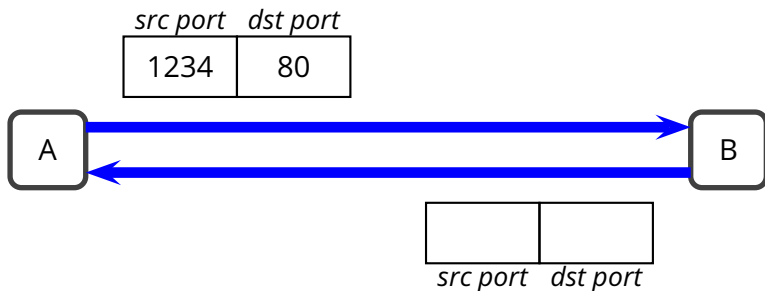


- The message format of both UDP and TCP starts with the source and destination port numbers

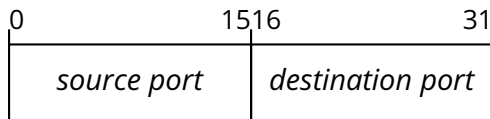


...

- E.g.,

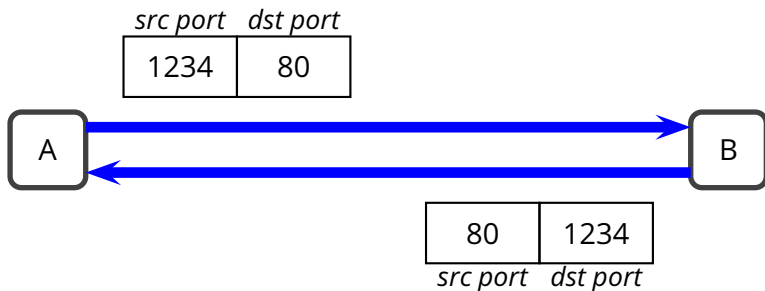


- The message format of both UDP and TCP starts with the source and destination port numbers



...

- E.g.,



Part V

Reliability

A Human Reliability Protocol

- Transmission coded for error detection
 - ▶ the sender sends information using a redundant “code”
 - ▶ the receiver can decode the correct data it receives, or it can notice a transmission error

A Human Reliability Protocol

- Transmission coded for error detection
 - ▶ the sender sends information using a redundant “code”
 - ▶ the receiver can decode the correct data it receives, or it can notice a transmission error
- Receiver feedback
 - ▶ the receiver sends positive or negative acknowledgments

A Human Reliability Protocol

- Transmission coded for error detection
 - ▶ the sender sends information using a redundant “code”
 - ▶ the receiver can decode the correct data it receives, or it can notice a transmission error
- Receiver feedback
 - ▶ the receiver sends positive or negative acknowledgments
- Retransmission
 - ▶ the sender retransmits upon a NACK or a timeout (silence)

A Human Reliability Protocol

- Transmission coded for error detection
 - ▶ the sender sends information using a redundant “code”
 - ▶ the receiver can decode the correct data it receives, or it can notice a transmission error
- Receiver feedback
 - ▶ the receiver sends positive or negative acknowledgments
- Retransmission
 - ▶ the sender retransmits upon a NACK or a timeout (silence)
- Sequence numbers
 - ▶ each transmission is stamped with a sequence number that the receiver can detect and discard data already delivered

Simple Reliability Protocol

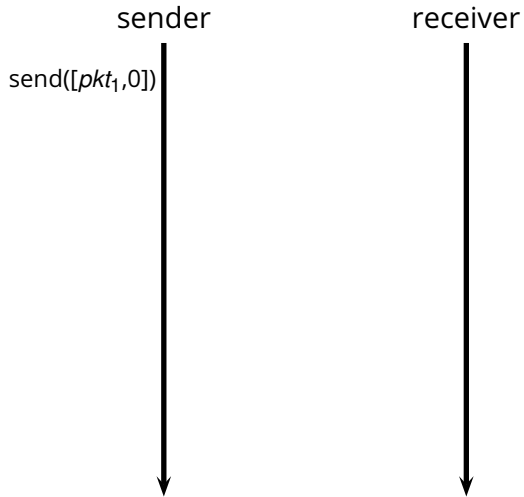
sender



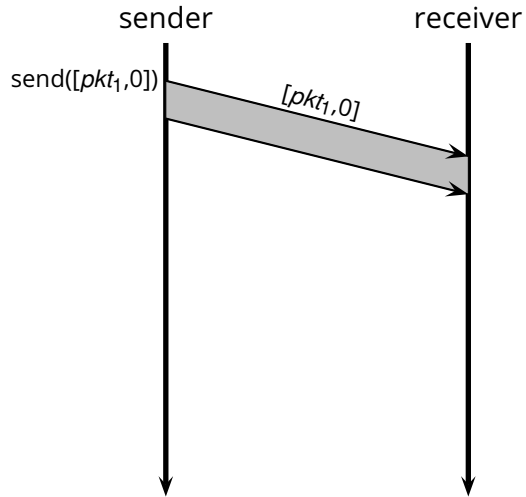
receiver



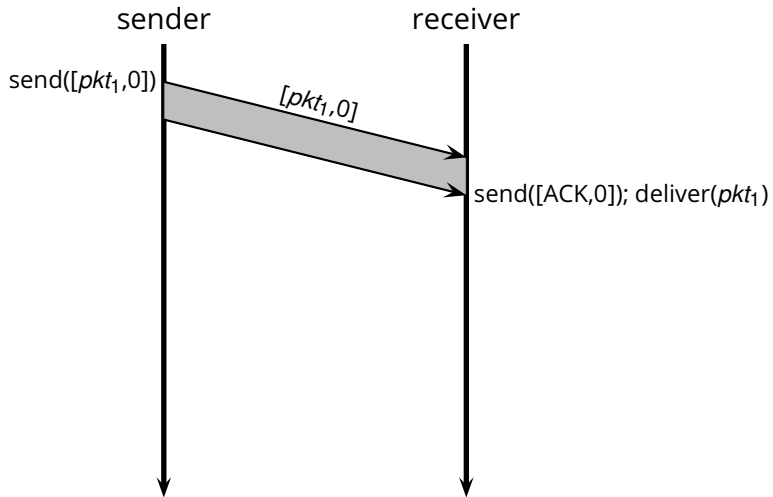
Simple Reliability Protocol



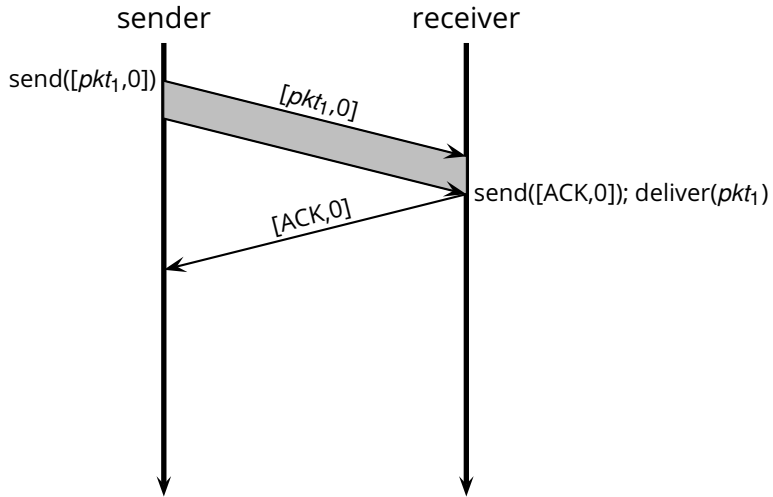
Simple Reliability Protocol



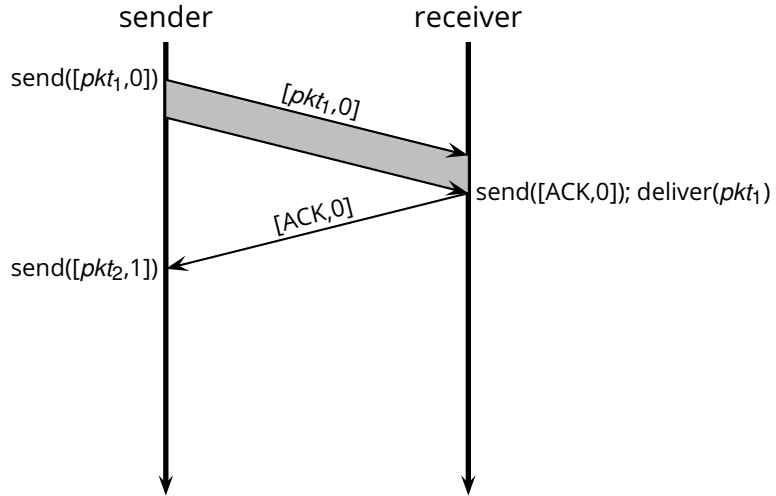
Simple Reliability Protocol



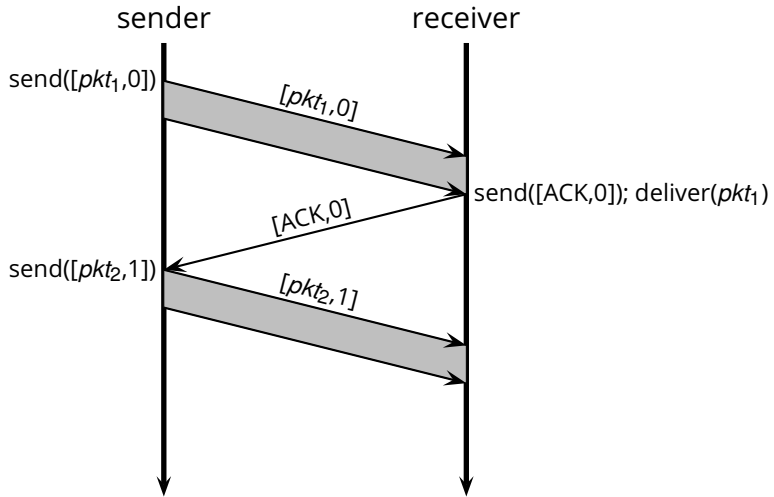
Simple Reliability Protocol



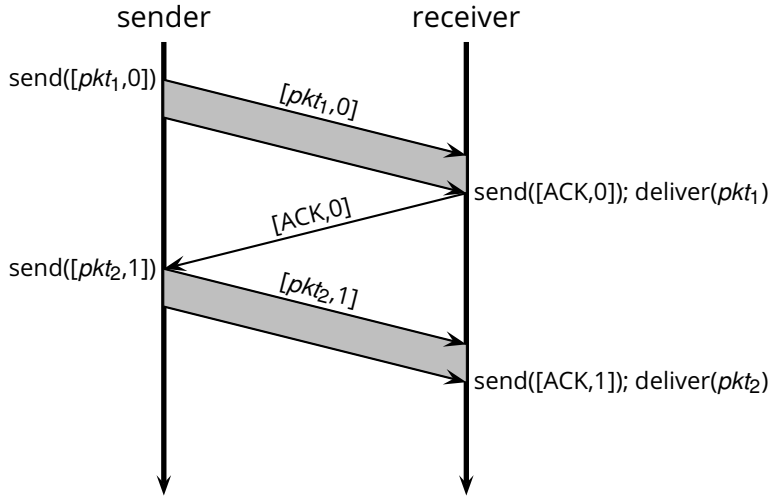
Simple Reliability Protocol



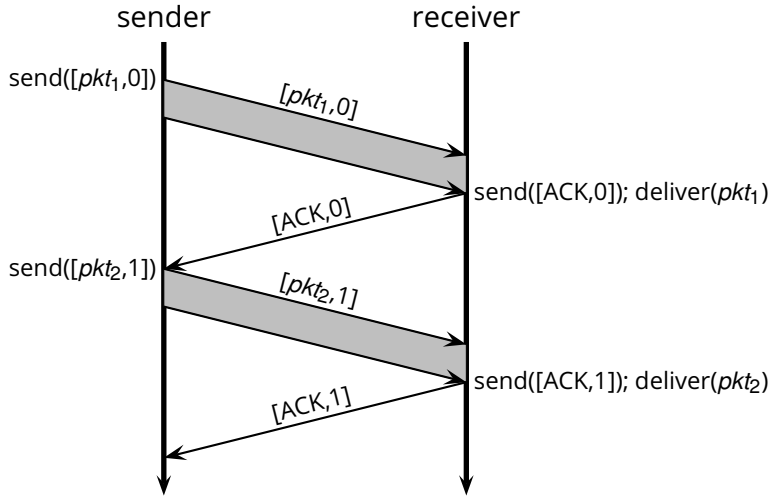
Simple Reliability Protocol

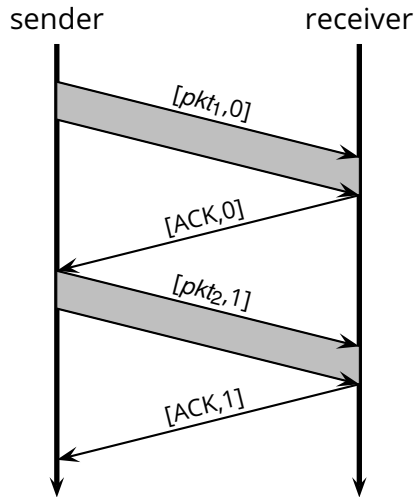


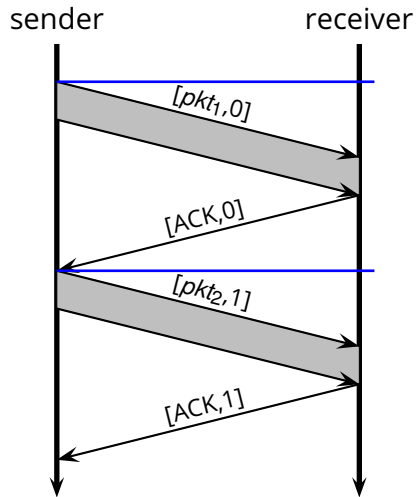
Simple Reliability Protocol

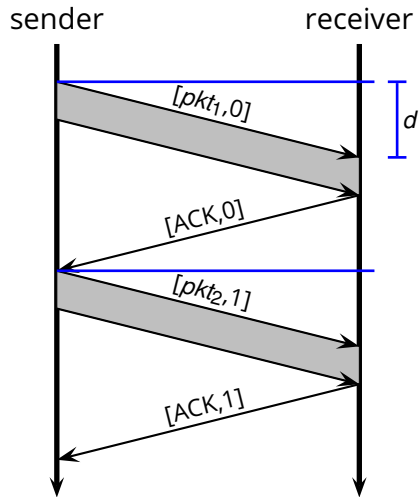


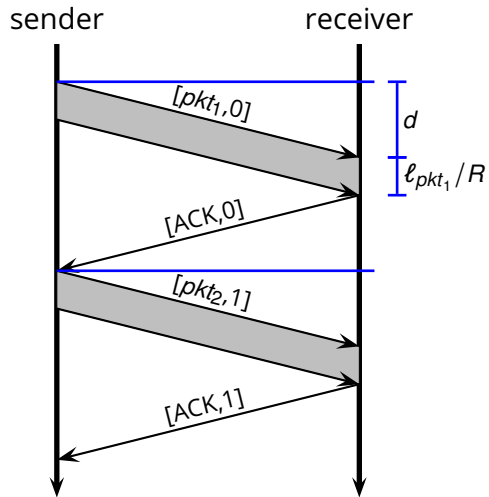
Simple Reliability Protocol

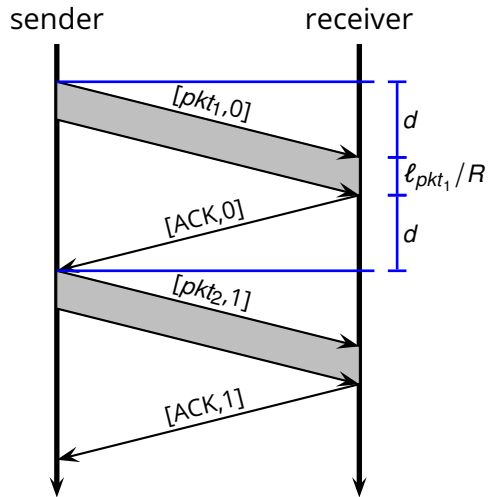


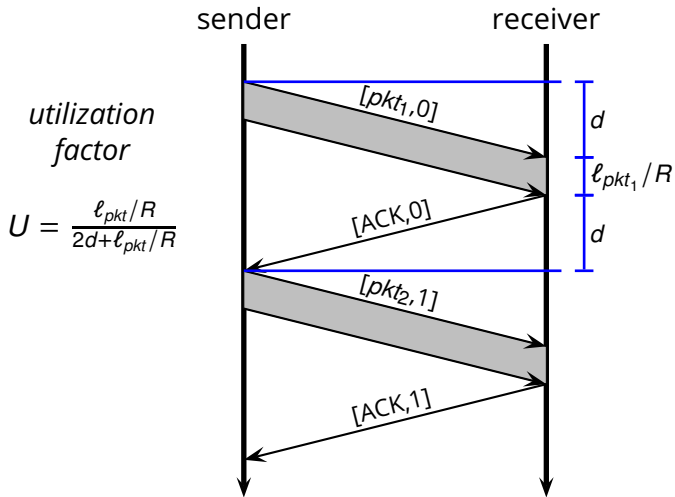










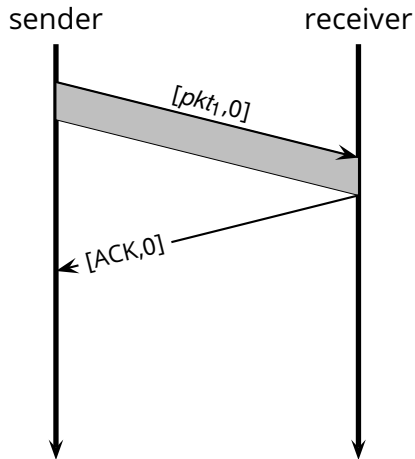


Example: How much of the network capacity is actually used over a link with with propagation delay $d_p = 50\text{ms}$, transmission rate $R = 1\text{MB/s}$, and maximum packet size $MTU = 1500\text{B}$?

- How do we achieve a better *utilization factor*?

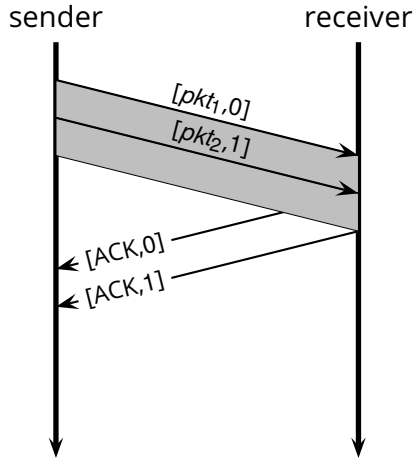
Improving Network Usage

- How do we achieve a better *utilization factor*?



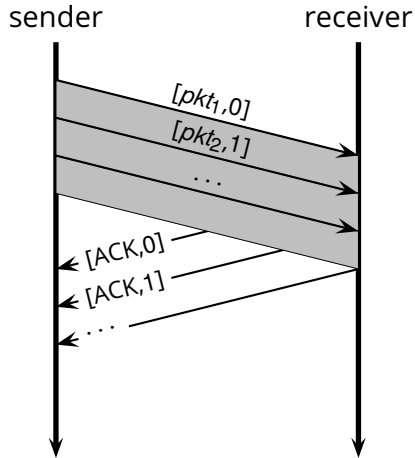
Improving Network Usage

- How do we achieve a better *utilization factor*?



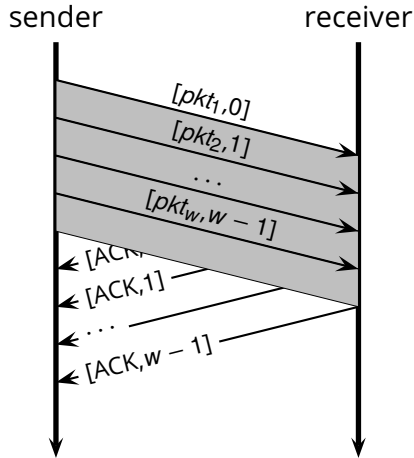
Improving Network Usage

- How do we achieve a better *utilization factor*?



Improving Network Usage

- How do we achieve a better *utilization factor*?



- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement

- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement
- Sender has up to W unacknowledged packets in the pipeline
 - ▶ the sender's state machine gets very complex
 - ▶ we represent the sender's state with its queue of acknowledgements

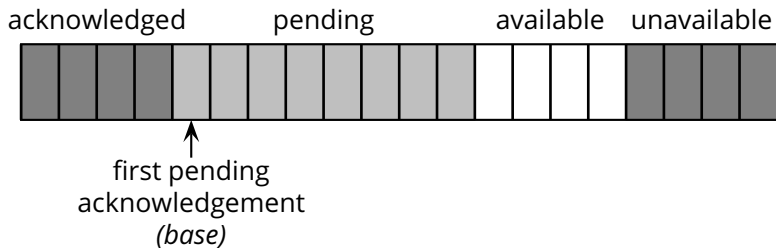
- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement
- Sender has up to W unacknowledged packets in the pipeline
 - ▶ the sender's state machine gets very complex
 - ▶ we represent the sender's state with its queue of acknowledgements



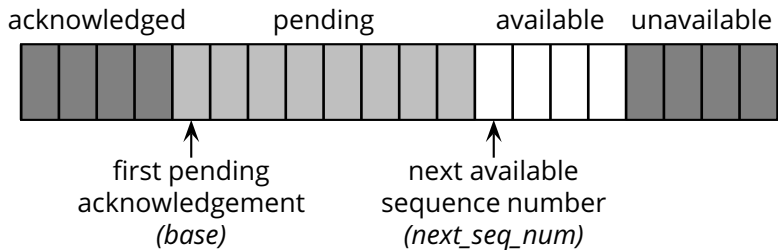
- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement
- Sender has up to W unacknowledged packets in the pipeline
 - ▶ the sender's state machine gets very complex
 - ▶ we represent the sender's state with its queue of acknowledgements



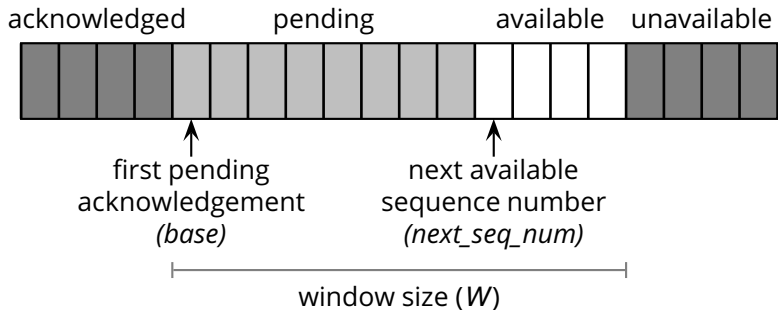
- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement
- Sender has up to W unacknowledged packets in the pipeline
 - ▶ the sender's state machine gets very complex
 - ▶ we represent the sender's state with its queue of acknowledgements



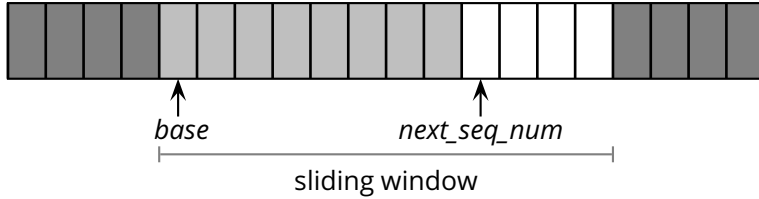
- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement
- Sender has up to W unacknowledged packets in the pipeline
 - ▶ the sender's state machine gets very complex
 - ▶ we represent the sender's state with its queue of acknowledgements



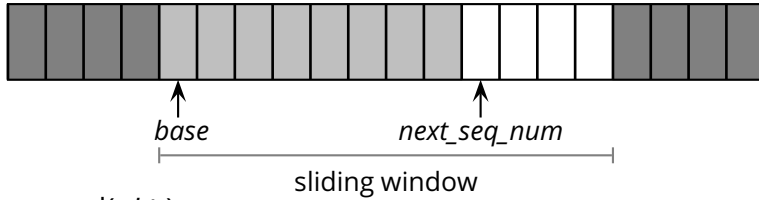
- **Idea:** the sender transmits multiple packets without waiting for an acknowledgement
- Sender has up to W unacknowledged packets in the pipeline
 - ▶ the sender's state machine gets very complex
 - ▶ we represent the sender's state with its queue of acknowledgements



Sliding Window Protocol: Sender

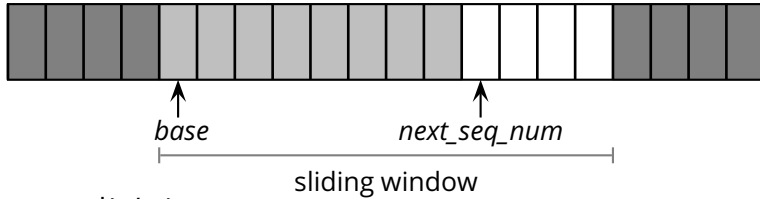


Sliding Window Protocol: Sender



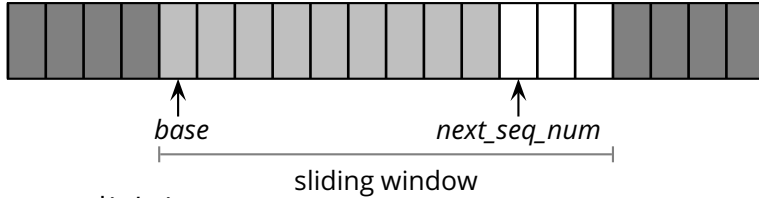
■ application_send(pkt_1)

Sliding Window Protocol: Sender



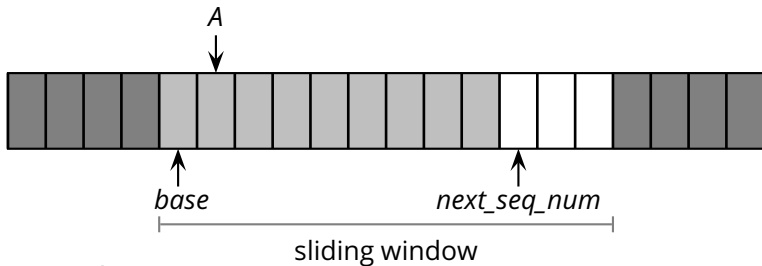
- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`

Sliding Window Protocol: Sender



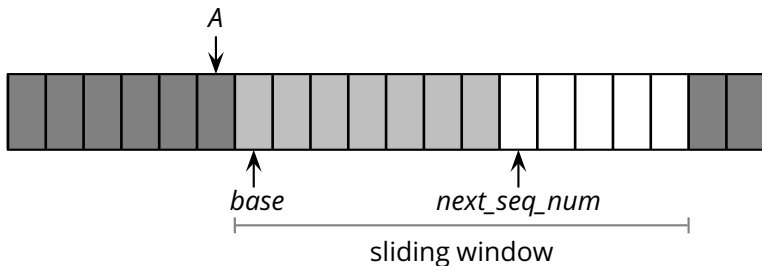
- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`
 - ▶ $next_seq_num \leftarrow next_seq_num + 1$

Sliding Window Protocol: Sender



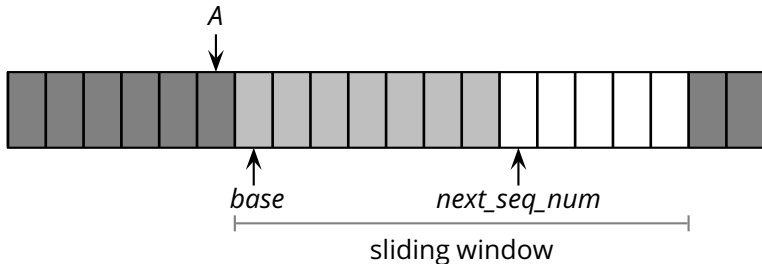
- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`
 - ▶ $next_seq_num \leftarrow next_seq_num + 1$
- `rcv([ACK, A])`

Sliding Window Protocol: Sender



- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`
 - ▶ $next_seq_num \leftarrow next_seq_num + 1$
- `recv([ACK, A])`
 - ▶ $base \leftarrow A + 1$

Sliding Window Protocol: Sender



■ `application_send(pkt1)`

- ▶ `send([pkt1, next_seq_num])`
- ▶ $next_seq_num \leftarrow next_seq_num + 1$

■ `recv([ACK, A])`

- ▶ $base \leftarrow A + 1$
- ▶ notice that acknowledgements are *cumulative*

- Concepts

- Concepts

- ▶ *sequence numbers*

- Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*

- Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*
- ▶ *cumulative acknowledgements*

■ Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*
- ▶ *cumulative acknowledgements*
- ▶ *checksums, timeouts, and sender-initiated retransmission*

■ Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*
- ▶ *cumulative acknowledgements*
- ▶ *checksums, timeouts, and sender-initiated retransmission*

■ Advantages: *simple, minimal state*

■ Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*
- ▶ *cumulative acknowledgements*
- ▶ *checksums, timeouts, and sender-initiated retransmission*

■ Advantages: *simple, minimal state*

- ▶ the sender maintains *two counters* and *one timer*, plus *packet buffer*
- ▶ the receiver maintains *one counter, no packet buffer*

- Concepts
 - ▶ *sequence numbers*
 - ▶ *sliding window*
 - ▶ *cumulative acknowledgements*
 - ▶ *checksums, timeouts*, and *sender-initiated retransmission*
- Advantages: *simple, minimal state*
 - ▶ the sender maintains *two counters* and *one timer*, plus *packet buffer*
 - ▶ the receiver maintains *one counter, no packet buffer*
- Disadvantages: *not optimal, not adaptive*

■ Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*
- ▶ *cumulative acknowledgements*
- ▶ *checksums, timeouts*, and *sender-initiated retransmission*

■ Advantages: *simple, minimal state*

- ▶ the sender maintains *two counters* and *one timer*, plus *packet buffer*
- ▶ the receiver maintains *one counter, no packet buffer*

■ Disadvantages: *not optimal, not adaptive*

- ▶ the sender can fill the window without filling the pipeline

■ Concepts

- ▶ *sequence numbers*
- ▶ *sliding window*
- ▶ *cumulative acknowledgements*
- ▶ *checksums, timeouts, and sender-initiated retransmission*

■ Advantages: *simple, minimal state*

- ▶ the sender maintains *two counters* and *one timer*, plus *packet buffer*
- ▶ the receiver maintains *one counter, no packet buffer*

■ Disadvantages: *not optimal, not adaptive*

- ▶ the sender can fill the window without filling the pipeline
- ▶ the receiver *could* buffer out-of-order packets...

- What is a good value for W ?

- What is a good value for W ?
 - ▶ W that achieves the *maximum utilization* of the connection

- What is a good value for W ?

- ▶ W that achieves the *maximum utilization* of the connection

ℓ = *stream*

d = *500ms*

R = *1Mb/s*

W = ?

- What is a good value for W ?

- ▶ W that achieves the *maximum utilization* of the connection

$$\ell = \text{stream}$$

$$d = 500\text{ms}$$

$$R = 1\text{Mb/s}$$

$$W = ?$$

- The problem may seem a bit underspecified. What is the (average) packet size?

$$\ell_{pkt} = 1\text{Kb}$$

$$d = 500\text{ms}$$

$$R = 1\text{Mb/s}$$

$$W = \frac{2d \times R}{\ell_{pkt}} = 1000$$

- The RTT-rate product ($2d \times R$) is the crucial factor

- The RTT-rate product ($2d \times R$) is the crucial factor
 - ▶ $W \times \ell_{pkt} \leq 2d \times R$
 - ▶ why $W \times \ell_{pkt} > 2d \times R$ doesn't make much sense?

- The RTT-rate product ($2d \times R$) is the crucial factor
 - ▶ $W \times \ell_{pkt} \leq 2d \times R$
 - ▶ why $W \times \ell_{pkt} > 2d \times R$ doesn't make much sense?
 - ▶ maximum channel utilization when $W \times \ell_{pkt} = 2d \times R$
 - ▶ $2d \times R$ can be thought of as the *capacity* of a connection

- Let's consider a fully utilized connection

- Let's consider a fully utilized connection

$$\ell_{pkt} = 1Kb$$

$$d = 500ms$$

$$R = 1Mb/s$$

$$W = \frac{R \times d}{\ell_{pkt}} = 1000$$

- Let's consider a fully utilized connection

$$\ell_{pkt} = 1Kb$$

$$d = 500ms$$

$$R = 1Mb/s$$

$$W = \frac{R \times d}{\ell_{pkt}} = 1000$$

- What happens if the first packet (or acknowledgement) is lost?

- Let's consider a fully utilized connection

$$\ell_{pkt} = 1Kb$$

$$d = 500ms$$

$$R = 1Mb/s$$

$$W = \frac{R \times d}{\ell_{pkt}} = 1000$$

- What happens if the first packet (or acknowledgement) is lost?
- Sender retransmits the entire content of its buffers

- Let's consider a fully utilized connection

$$\ell_{pkt} = 1Kb$$

$$d = 500ms$$

$$R = 1Mb/s$$

$$W = \frac{R \times d}{\ell_{pkt}} = 1000$$

- What happens if the first packet (or acknowledgement) is lost?
- Sender retransmits the entire content of its buffers
 - ▶ $W \times \ell_{pkt} = 2d \times R = 1Mb$
 - ▶ retransmitting 1Mb to recover 1Kb worth of data isn't exactly the best solution. Not to mention congestions...

- Let's consider a fully utilized connection

$$\ell_{pkt} = 1Kb$$

$$d = 500ms$$

$$R = 1Mb/s$$

$$W = \frac{R \times d}{\ell_{pkt}} = 1000$$

- What happens if the first packet (or acknowledgement) is lost?
- Sender retransmits the entire content of its buffers
 - ▶ $W \times \ell_{pkt} = 2d \times R = 1Mb$
 - ▶ retransmitting 1Mb to recover 1Kb worth of data isn't exactly the best solution. Not to mention congestions...
- Is there a better way to deal with retransmissions?

- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted

- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted
 - ▶ sender maintains a vector of acknowledgement flags

- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted
 - ▶ sender maintains a vector of acknowledgement flags
 - ▶ receiver maintains a vector of acknowledged flags

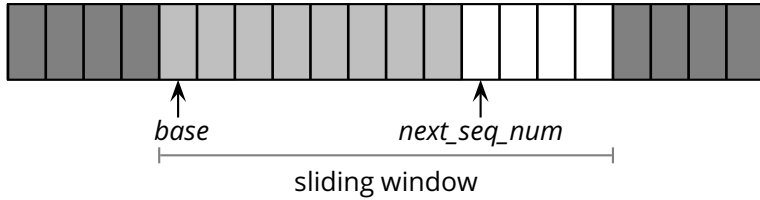
- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted
 - ▶ sender maintains a vector of acknowledgement flags
 - ▶ receiver maintains a vector of acknowledged flags
 - ▶ in fact, receiver maintains a buffer of out-of-order packets

- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted
 - ▶ sender maintains a vector of acknowledgement flags
 - ▶ receiver maintains a vector of acknowledged flags
 - ▶ in fact, receiver maintains a buffer of out-of-order packets
 - ▶ sender maintains a timer for each pending packet

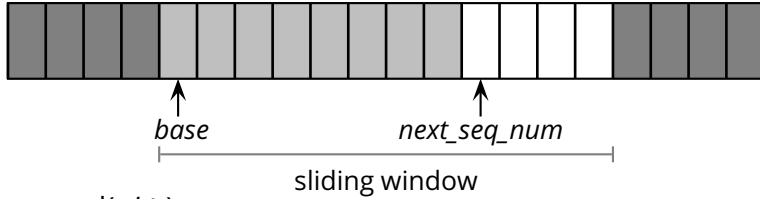
- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted
 - ▶ sender maintains a vector of acknowledgement flags
 - ▶ receiver maintains a vector of acknowledged flags
 - ▶ in fact, receiver maintains a buffer of out-of-order packets
 - ▶ sender maintains a timer for each pending packet
 - ▶ sender resends a packet when its timer expires

- **Idea:** have the sender retransmit only those packets that it suspects were lost or corrupted
 - ▶ sender maintains a vector of acknowledgement flags
 - ▶ receiver maintains a vector of acknowledged flags
 - ▶ in fact, receiver maintains a buffer of out-of-order packets
 - ▶ sender maintains a timer for each pending packet
 - ▶ sender resends a packet when its timer expires
 - ▶ sender slides the window when the lowest pending sequence number is acknowledged

Selective Repeat: Sender

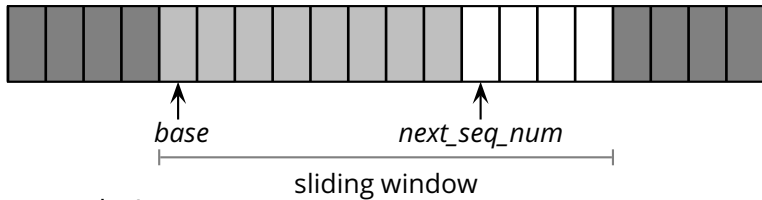


Selective Repeat: Sender



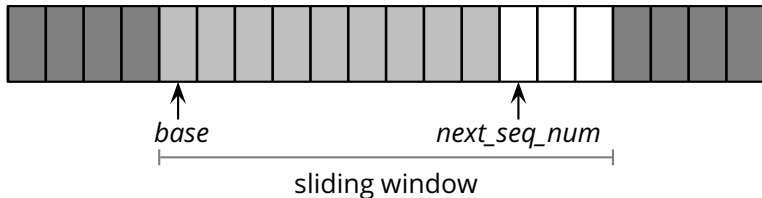
■ application_send(pkt_1)

Selective Repeat: Sender



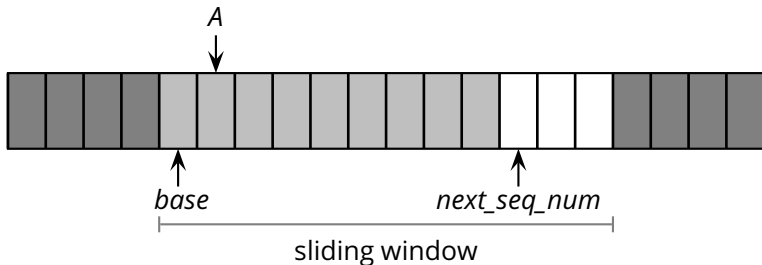
- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`
 - ▶ `start_timer(next_seq_num)`

Selective Repeat: Sender



- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`
 - ▶ `start_timer(next_seq_num)`
 - ▶ $next_seq_num \leftarrow next_seq_num + 1$

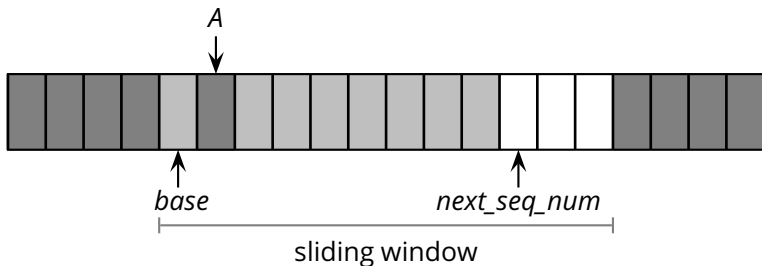
Selective Repeat: Sender



- `application_send(pkt1)`
 - ▶ `send([pkt1, next_seq_num])`
 - ▶ `start_timer(next_seq_num)`
 - ▶ $next_seq_num \leftarrow next_seq_num + 1$

- `rcv([ACK, A])`

Selective Repeat: Sender



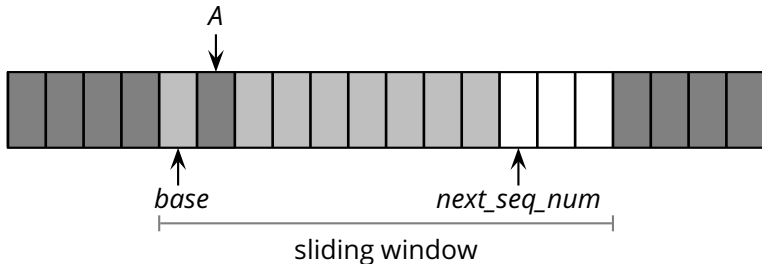
■ application_send(pkt_1)

- ▶ send($[pkt_1, next_seq_num]$)
- ▶ start_timer($next_seq_num$)
- ▶ $next_seq_num \leftarrow next_seq_num + 1$

■ rcv($[ACK, A]$)

- ▶ $acks[A] \leftarrow 1$ // remember that A was ACK'd

Selective Repeat: Sender



■ application_send(pkt_1)

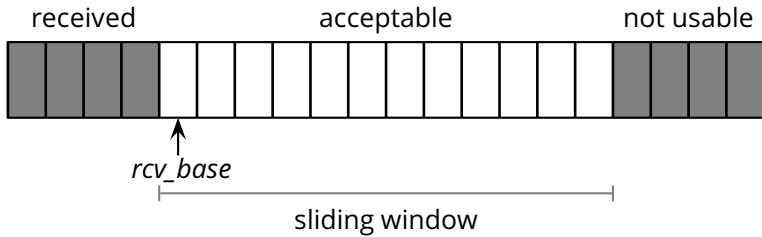
- ▶ send($[pkt_1, next_seq_num]$)
- ▶ start_timer($next_seq_num$)
- ▶ $next_seq_num \leftarrow next_seq_num + 1$

■ recv($[ACK, A]$)

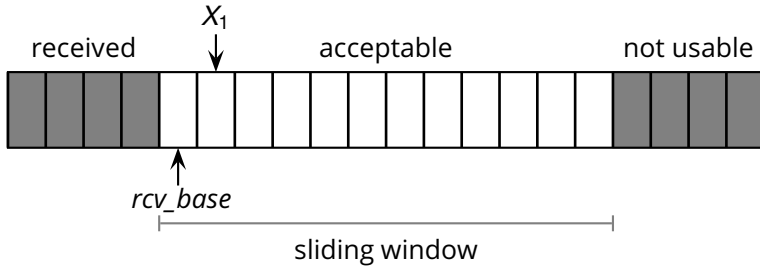
- ▶ $acks[A] \leftarrow 1$ // remember that A was ACK'd
- ▶ acknowledgements are *no longer cumulative*

Selective Repeat: Receiver

Selective Repeat: Receiver

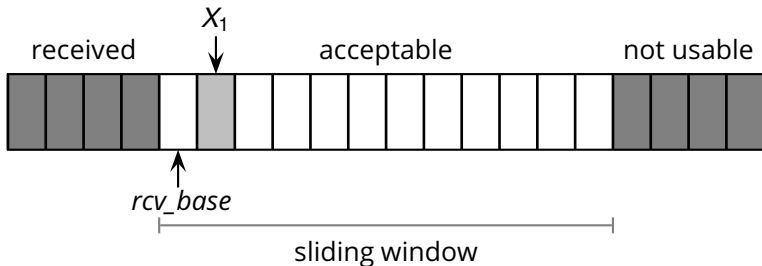


Selective Repeat: Receiver



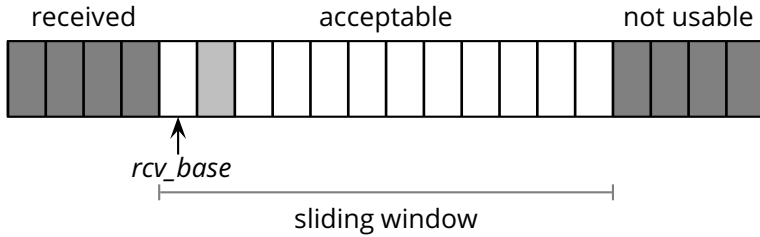
- $rcv([pkt_1, X_1])$ and $rcv_base \leq X_1 < rcv_base + W$

Selective Repeat: Receiver

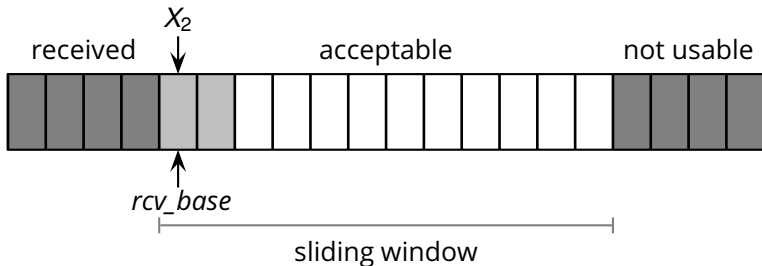


- $rcv([pkt_1, X_1])$ **and** $rcv_base \leq X_1 < rcv_base + W$
 - ▶ $buffer[X_1] \leftarrow pkt_1$
 - ▶ $send([ACK, X_1]^*)$ // no longer a "cumulative" ACK

Selective Repeat: Receiver

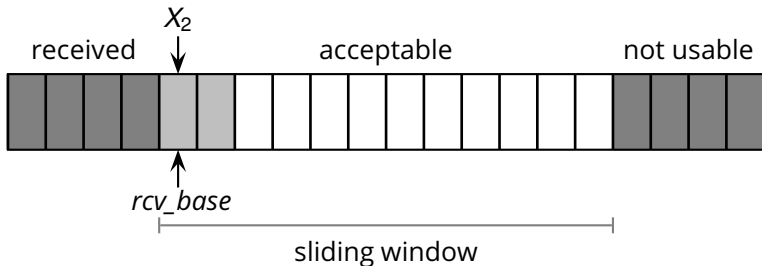


Selective Repeat: Receiver



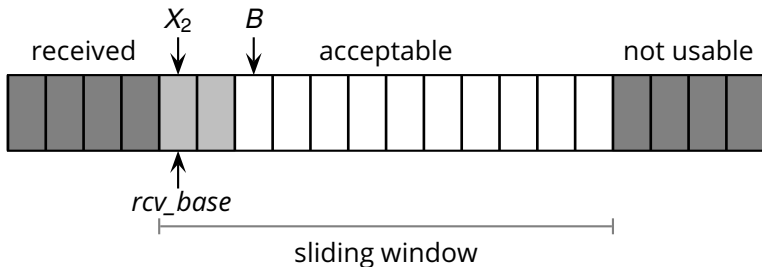
- $rcv([pkt_2, X_2])$ and $rcv_base \leq X_2 < rcv_base + W$
 - ▶ $buffer[X_2] \leftarrow pkt_2$
 - ▶ $send([ACK, X_2]^*)$

Selective Repeat: Receiver



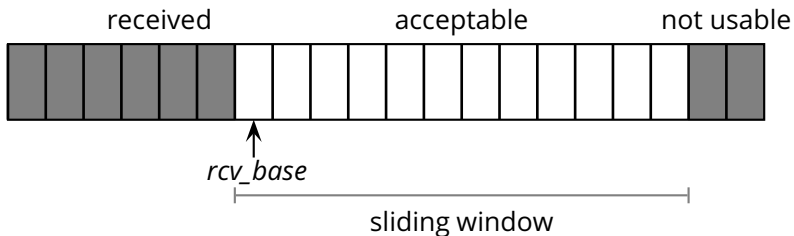
- $recv([pkt_2, X_2])$ **and** $rcv_base \leq X_2 < rcv_base + W$
 - ▶ $buffer[X_2] \leftarrow pkt_2$
 - ▶ $send([ACK, X_2]^*)$
 - ▶ **if** $X_2 = rcv_base$:

Selective Repeat: Receiver



- $recv([pkt_2, X_2])$ and $rcv_base \leq X_2 < rcv_base + W$
 - ▶ $buffer[X_2] \leftarrow pkt_2$
 - ▶ $send([ACK, X_2]^*)$
 - ▶ **if** $X_2 = rcv_base$:
 - $B \leftarrow first_missing_seq_num()$
 - foreach** i in $rcv_base \dots B - 1$:
 - $deliver(buffer[i])$

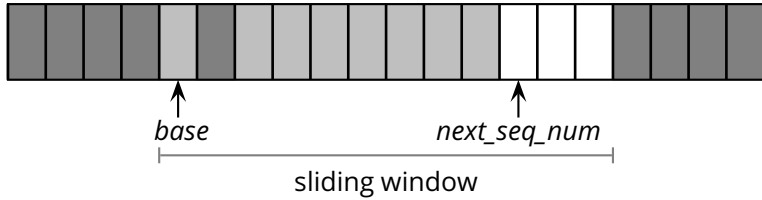
Selective Repeat: Receiver



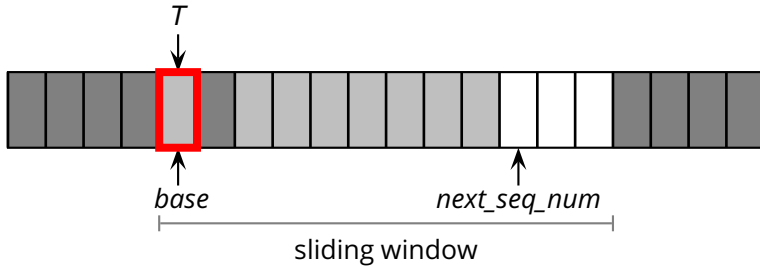
- $recv([pkt_2, X_2])$ and $rcv_base \leq X_2 < rcv_base + W$
 - ▶ $buffer[X_2] \leftarrow pkt_2$
 - ▶ $send([ACK, X_2]^*)$
 - ▶ **if** $X_2 = rcv_base$:
 - $B \leftarrow first_missing_seq_num()$
 - foreach** i in $rcv_base \dots B - 1$:
 - $deliver(buffer[i])$
 - $rcv_base \leftarrow B$

Selective Repeat: Sender

Selective Repeat: Sender

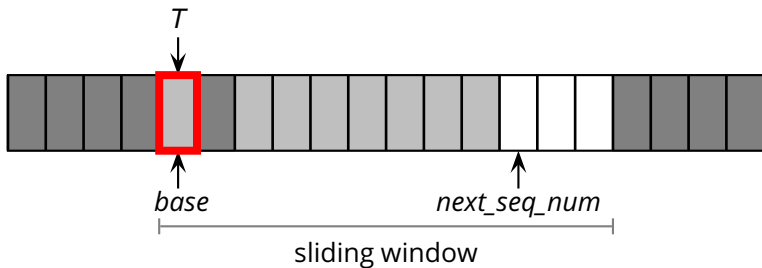


Selective Repeat: Sender



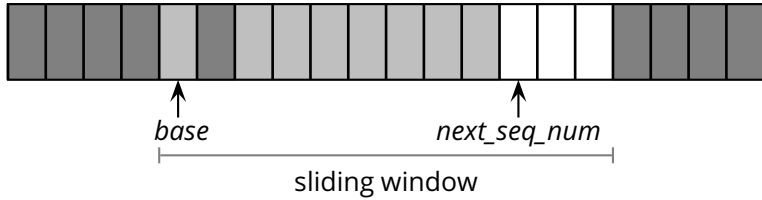
- Timeout for sequence number T

Selective Repeat: Sender

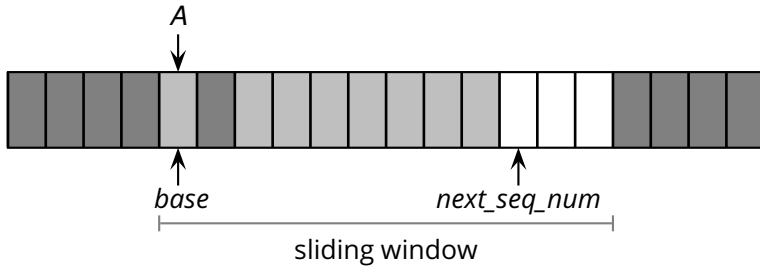


- Timeout for sequence number T
 - ▶ $send([pkt[T], T]^*)$

Selective Repeat: Sender

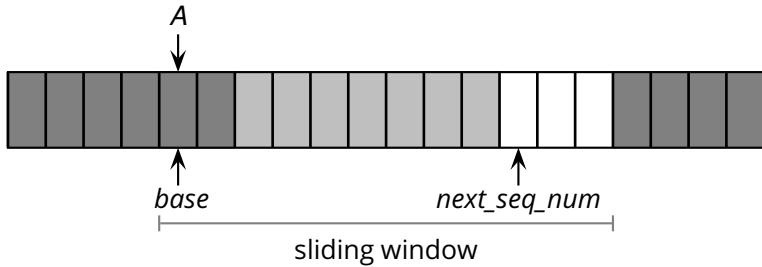


Selective Repeat: Sender



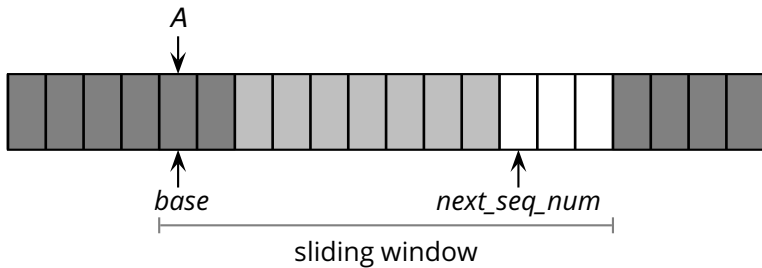
■ `recv([ACK,A])`

Selective Repeat: Sender



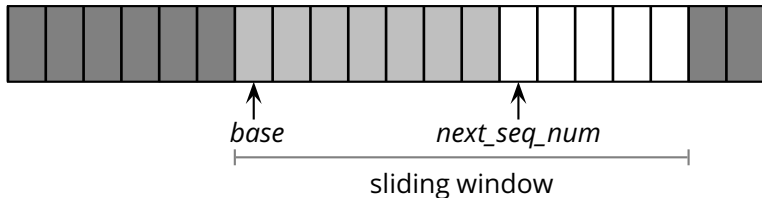
- `recv([ACK,A])`
 - ▶ `acks[A] ← 1`

Selective Repeat: Sender



- `recv([ACK,A])`
 - ▶ $acks[A] \leftarrow 1$
 - ▶ **if** $A = base$:

Selective Repeat: Sender



- `recv([ACK,A])`
 - ▶ `acks[A] ← 1`
 - ▶ **if** $A = base$:
 - $base \leftarrow first_missing_ack_num()$

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

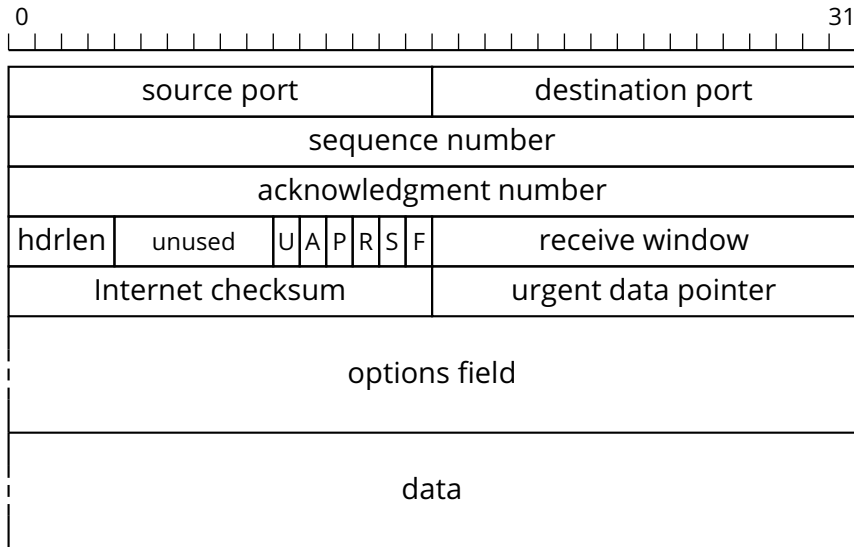
Preliminary Definitions

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

- **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)

- **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



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

- *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

- *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

- *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

- *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

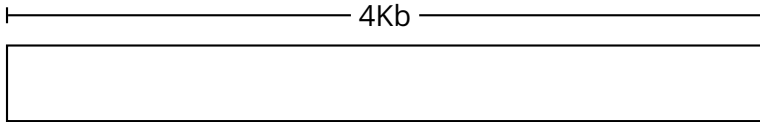
Sequence Numbers

- 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

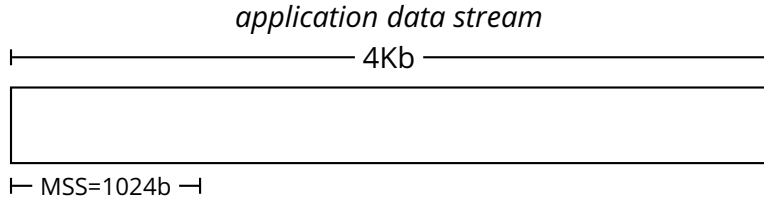
- Sequence numbers are associated with *bytes* in the data stream
 - ▶ not with segments, as we have used them before
- The ***sequence number*** in a TCP segment indicates ***the sequence number of the first byte carried by that segment***

application data stream



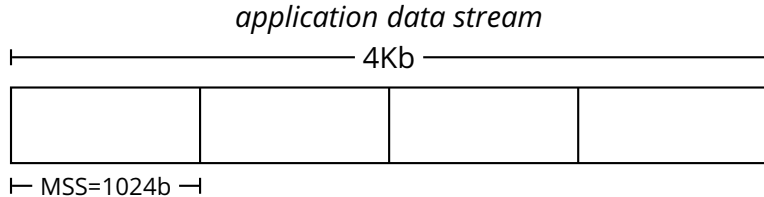
Sequence Numbers

- 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***



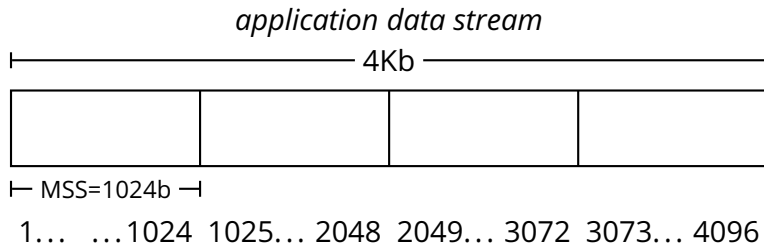
Sequence Numbers

- 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***



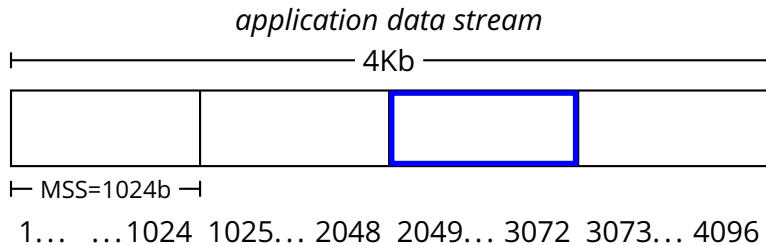
Sequence Numbers

- 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***

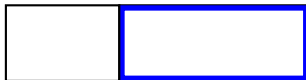


Sequence Numbers

- 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***

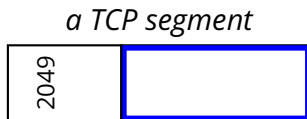
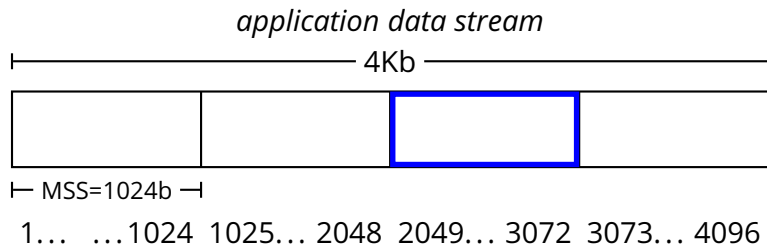


a TCP segment



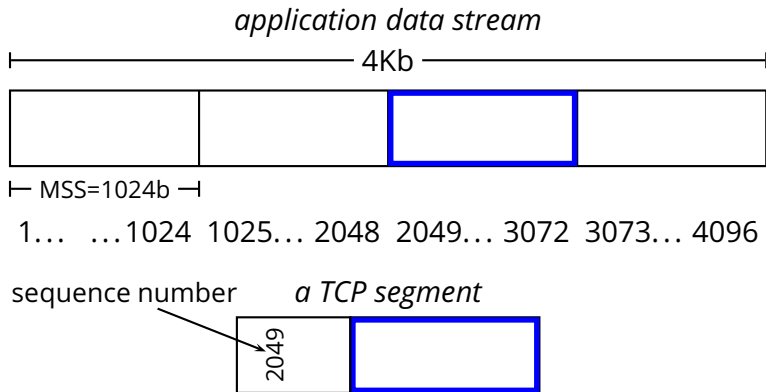
Sequence Numbers

- 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***



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***



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*

A

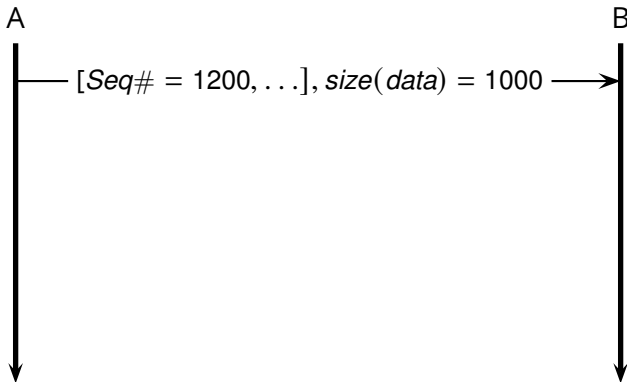


B



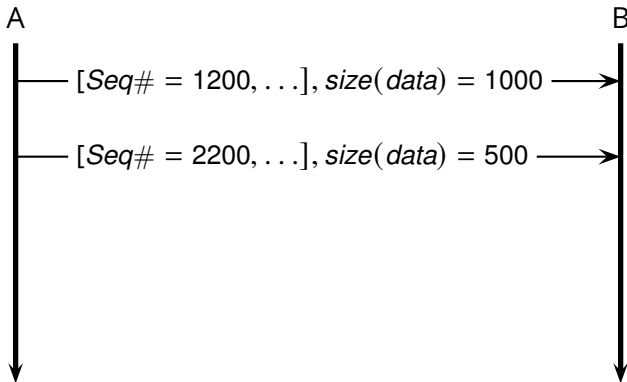
Acknowledgment Numbers

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



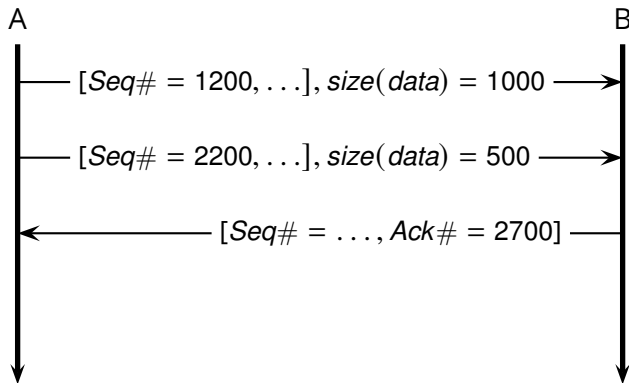
Acknowledgment Numbers

- 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:

A

A vertical black arrow pointing downwards, starting below the letter 'A' and ending with a triangular arrowhead.

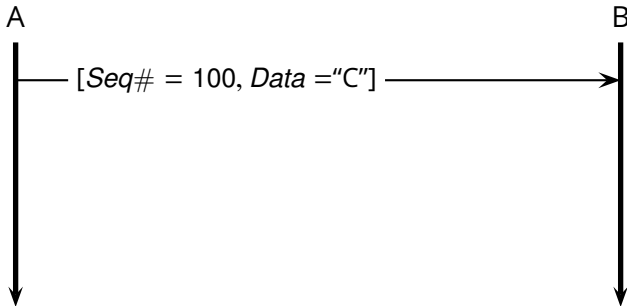
B

A vertical black arrow pointing downwards, starting below the letter 'B' and ending with a triangular arrowhead.

Sequence Numbers and ACK Numbers

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

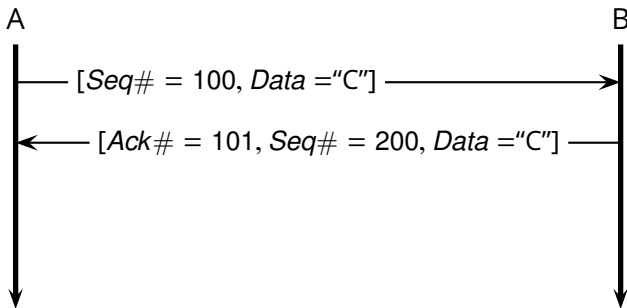
E.g., consider a simple “Echo” application:



Sequence Numbers and ACK Numbers

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

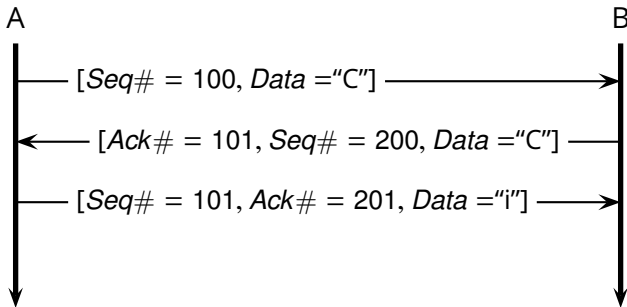
E.g., consider a simple “Echo” application:



Sequence Numbers and ACK Numbers

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

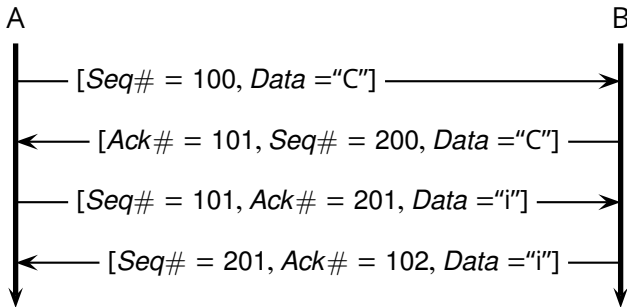
E.g., consider a simple “Echo” application:



Sequence Numbers and ACK Numbers

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

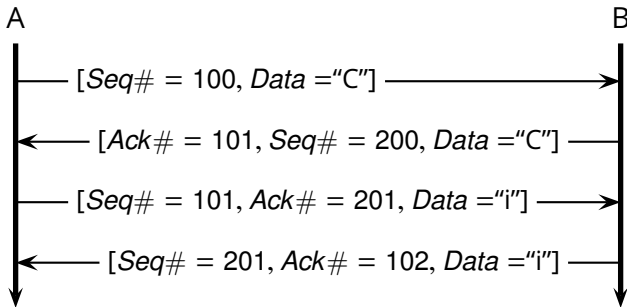
E.g., consider a simple “Echo” application:



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:



- Acknowledgments are “piggybacked” on data segments

Reliability and Timeout

- Duplicate acknowledgments to detect lost segments
 - ▶ receiver notices a missing packet → duplicate ACKs → retransmission by sender
- A *timer* to detect lost segments
 - ▶ timeout without an ACK → lost packet → retransmission

Reliability and Timeout

- Duplicate acknowledgments to detect lost segments
 - ▶ receiver notices a missing packet → duplicate ACKs → retransmission by sender
- A *timer* to detect lost segments
 - ▶ timeout without an ACK → lost packet → retransmission
- How long to wait for acknowledgments?

Reliability and Timeout

- Duplicate acknowledgments to detect lost segments
 - ▶ receiver notices a missing packet → duplicate ACKs → retransmission by sender
- A *timer* to detect lost segments
 - ▶ timeout without an ACK → lost packet → 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

- Duplicate acknowledgments to detect lost segments
 - ▶ receiver notices a missing packet → duplicate ACKs → retransmission by sender
- A *timer* to detect lost segments
 - ▶ timeout without an ACK → lost packet → 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 $\beta = 0.25$

- 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

- 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} :

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

A simplified TCP sender

■ `application_send(data)`

if (timer not running)

`start_timer()`

`send([data, next_seq_num])`

$next_seq_num \leftarrow next_seq_num + length(data)$

A simplified TCP sender

- application_send(*data*)

- if** (timer not running)

- start_timer()

- send(*[data,next_seq_num]*)

- $next_seq_num \leftarrow next_seq_num + length(data)$

- timeout

- send(pending segment with smallest sequence number)

- start_timer()

A simplified TCP sender

- application_send(*data*)

- if** (timer not running)

- start_timer()

- send(*[data, next_seq_num]*)

- $next_seq_num \leftarrow next_seq_num + length(data)$

- timeout

- send(pending segment with smallest sequence number)

- start_timer()

- 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)

■ `recv([ACK,y])`

if ($y > base$)

$base \leftarrow y$

if (there are pending segments)

`start_timer()`

- `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 ($ack_counter[y] = 3$)

`send(segment with sequence number y)`

Three-way handshake

Connection Setup

Three-way handshake

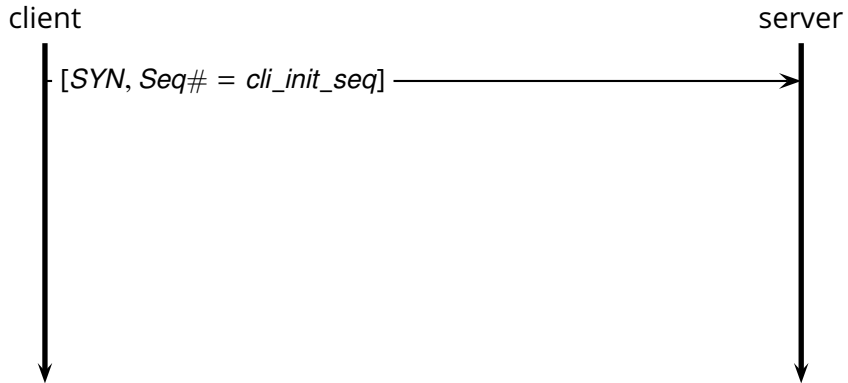
client



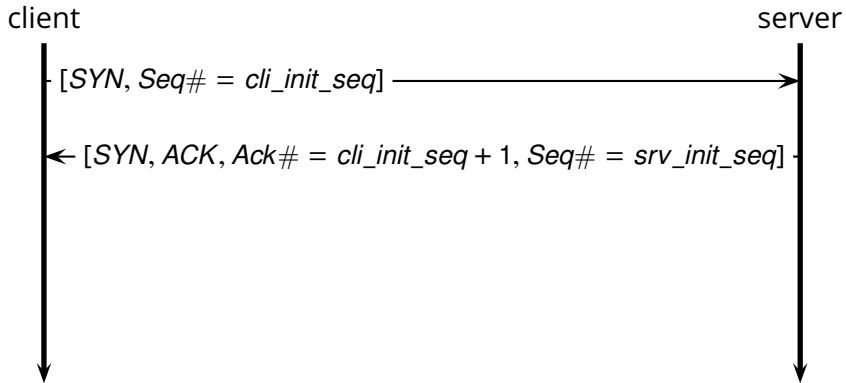
server



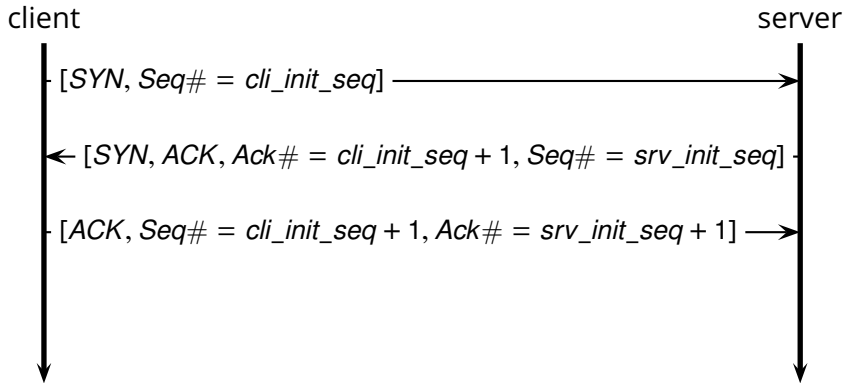
Three-way handshake



Three-way handshake



Three-way handshake



Connection Shutdown

"This is it."

"Okay, Bye now."

"Bye."

Connection Shutdown

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

client



server

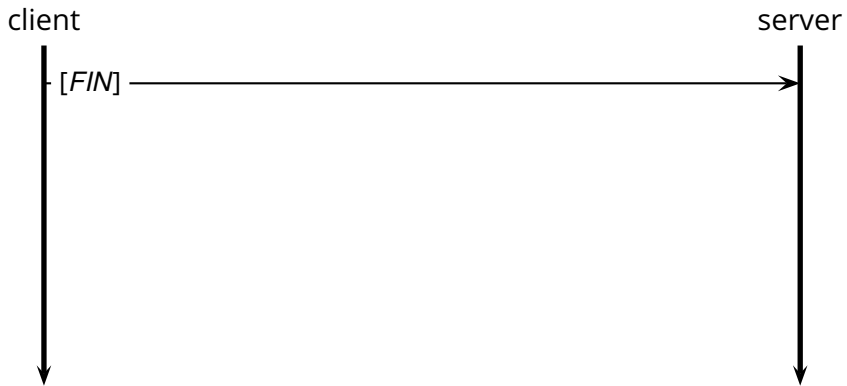


Connection Shutdown

"This is it."

"Okay, Bye now."

"Bye."

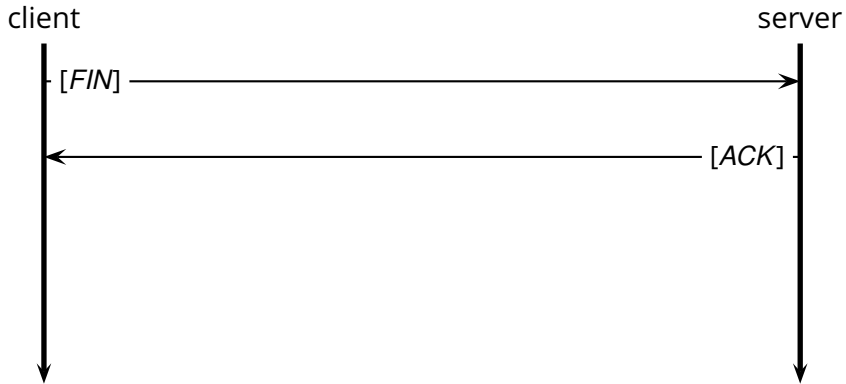


Connection Shutdown

"This is it."

"Okay, Bye now."

"Bye."

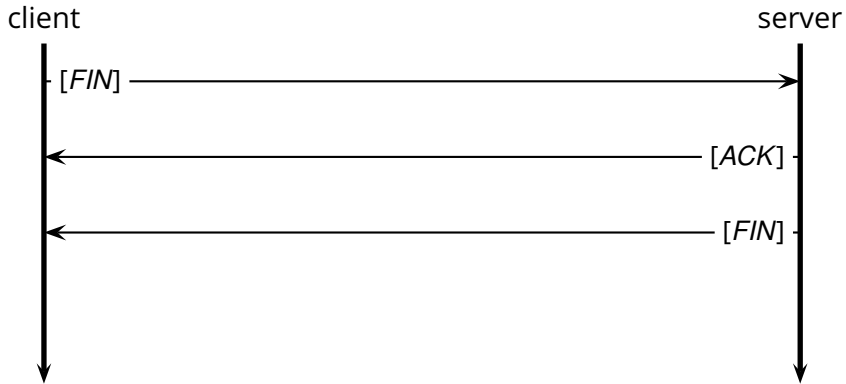


Connection Shutdown

"This is it."

"Okay, Bye now."

"Bye."

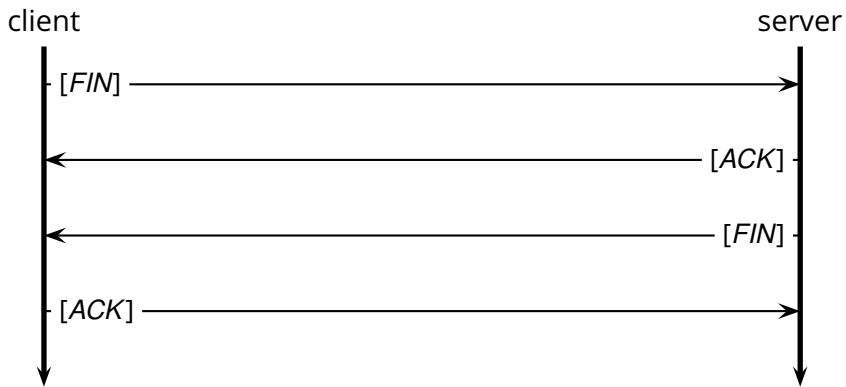


Connection Shutdown

"This is it."

"Okay, Bye now."

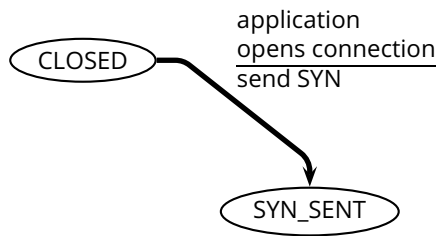
"Bye."



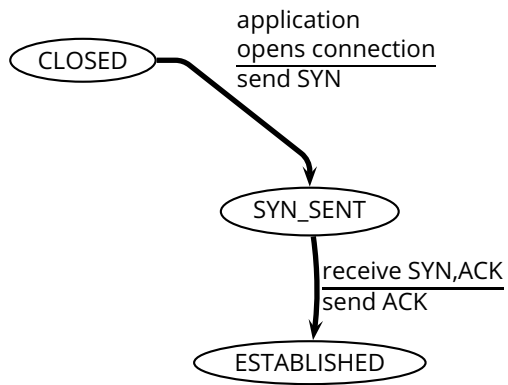
The TCP State Machine (Client)

CLOSED

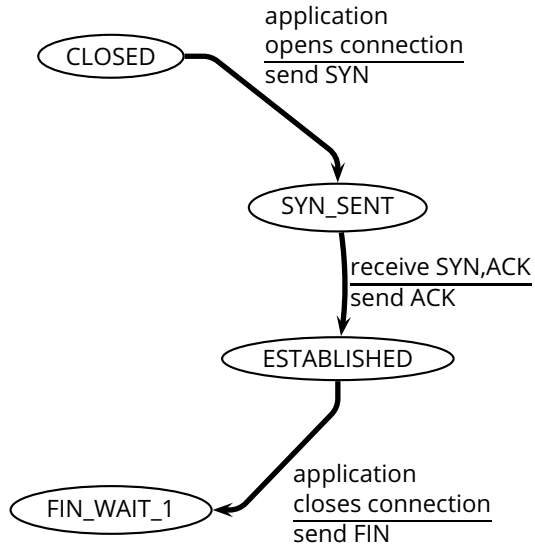
The TCP State Machine (Client)



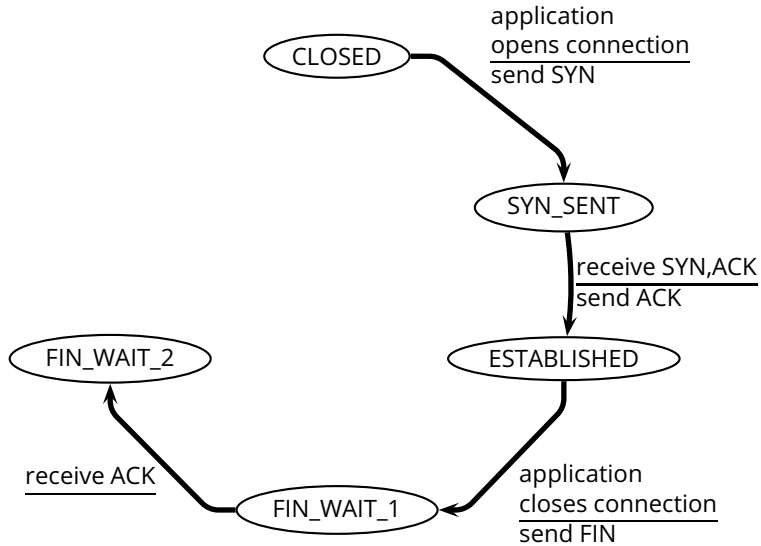
The TCP State Machine (Client)



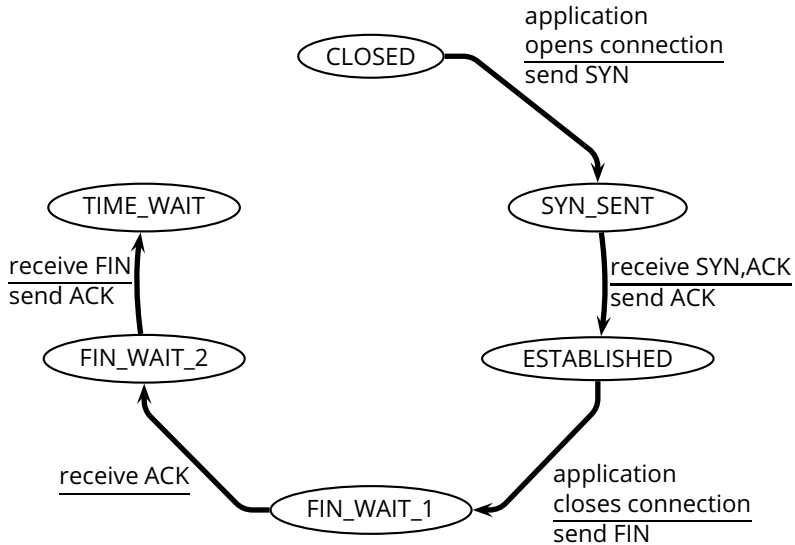
The TCP State Machine (Client)



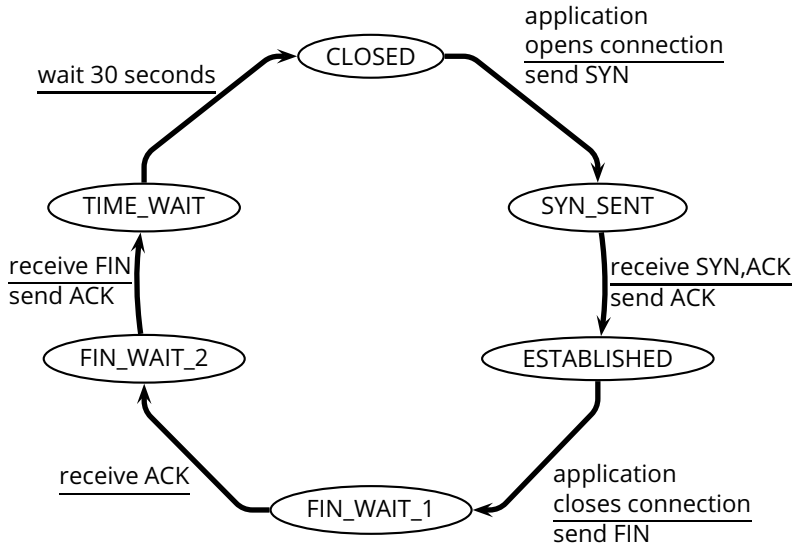
The TCP State Machine (Client)



The TCP State Machine (Client)



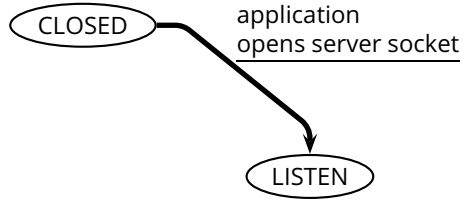
The TCP State Machine (Client)



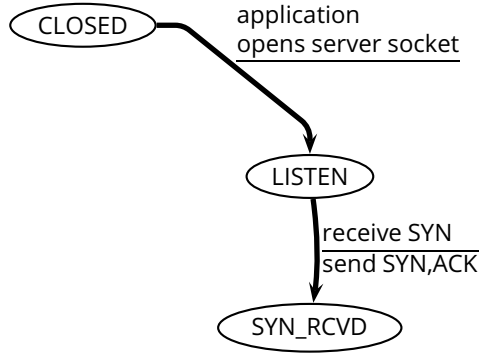
The TCP State Machine (Server)

CLOSED

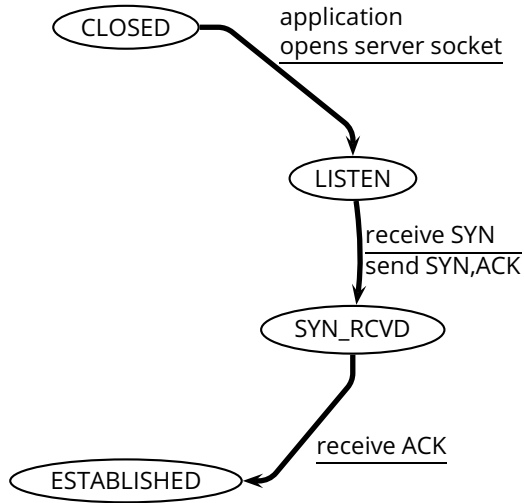
The TCP State Machine (Server)



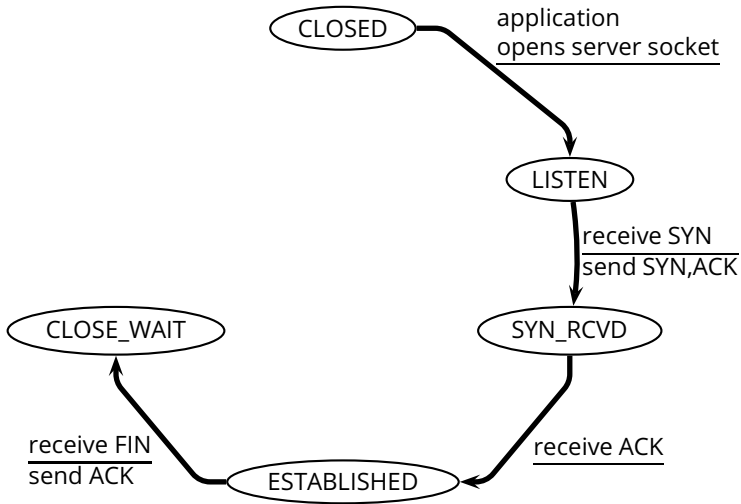
The TCP State Machine (Server)



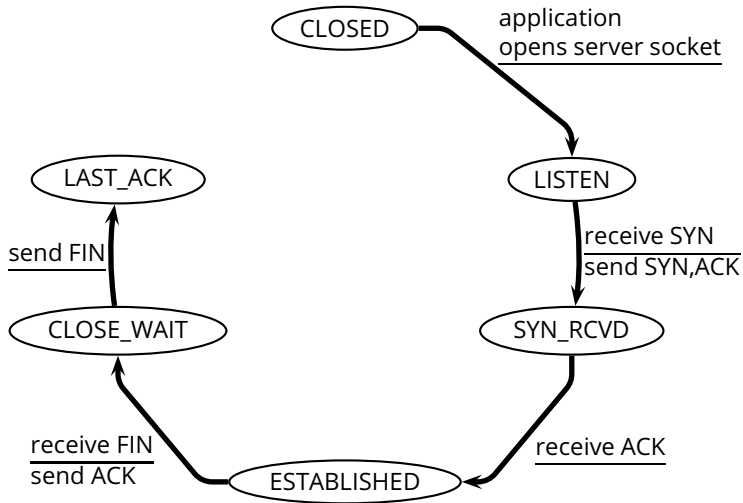
The TCP State Machine (Server)



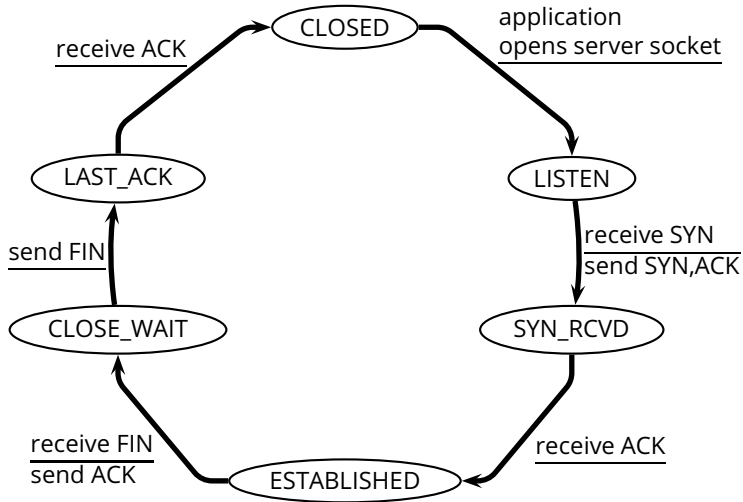
The TCP State Machine (Server)



The TCP State Machine (Server)



The TCP State Machine (Server)



Part VI

Congestion Control

Understanding Congestion

- A router behaves a lot like a kitchen sink

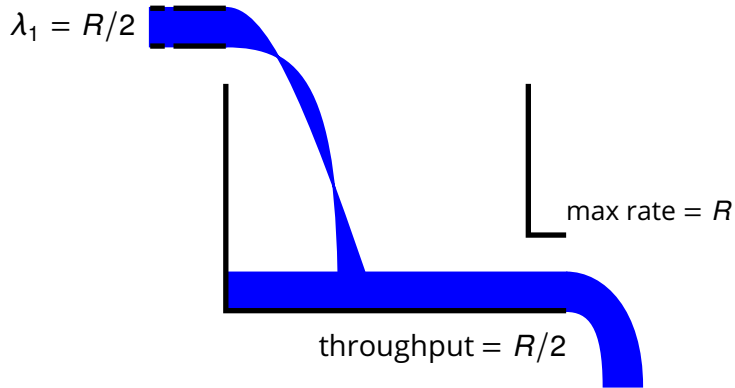
Understanding Congestion

- A router behaves a lot like a kitchen sink



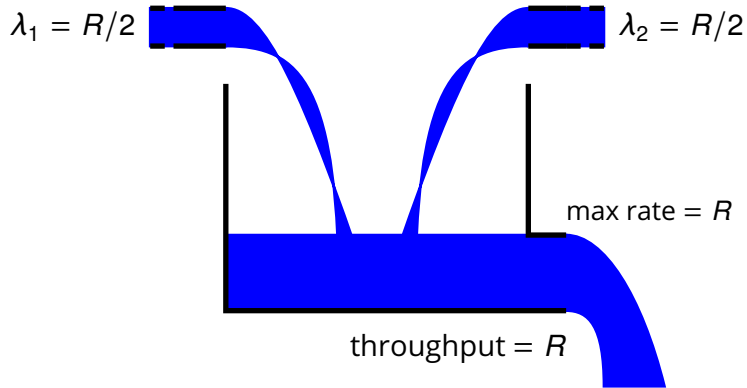
Understanding Congestion

- A router behaves a lot like a kitchen sink



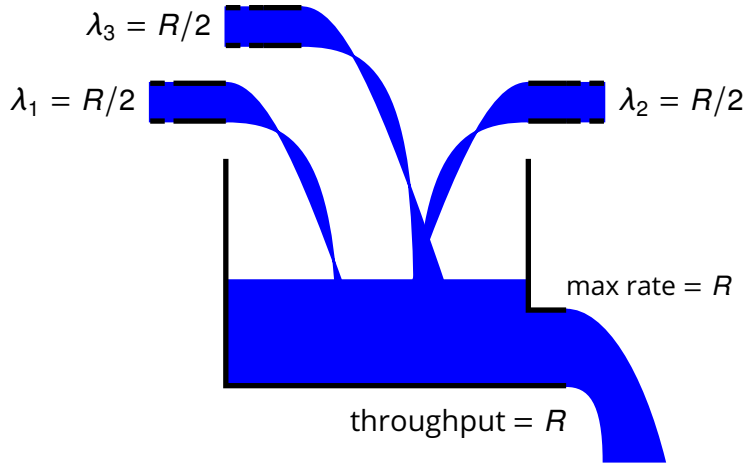
Understanding Congestion

- A router behaves a lot like a kitchen sink



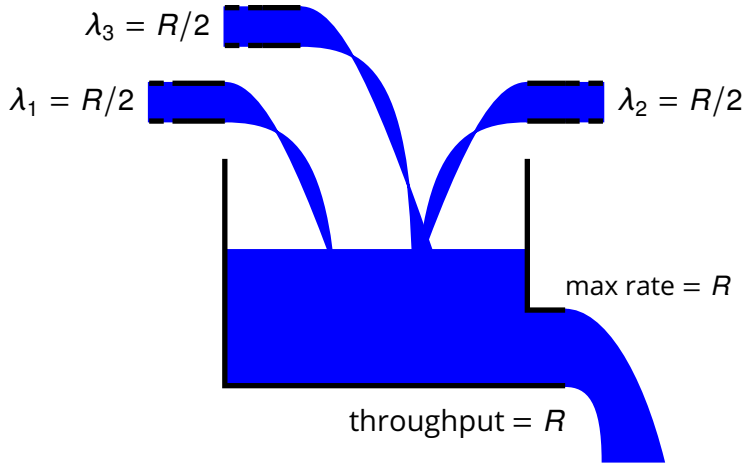
Understanding Congestion

- A router behaves a lot like a kitchen sink



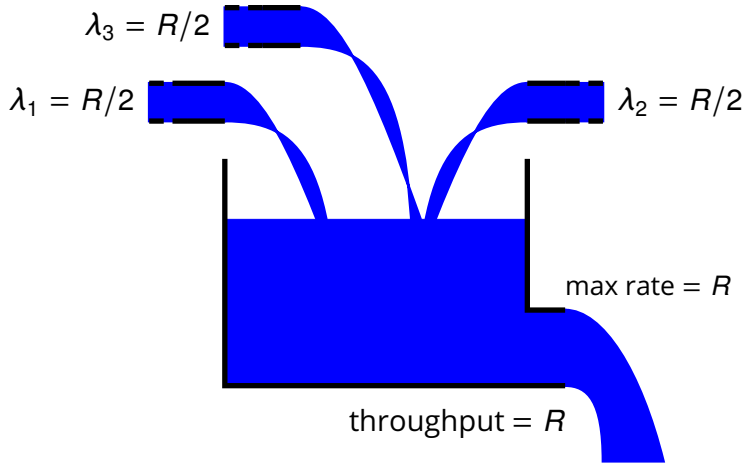
Understanding Congestion

- A router behaves a lot like a kitchen sink



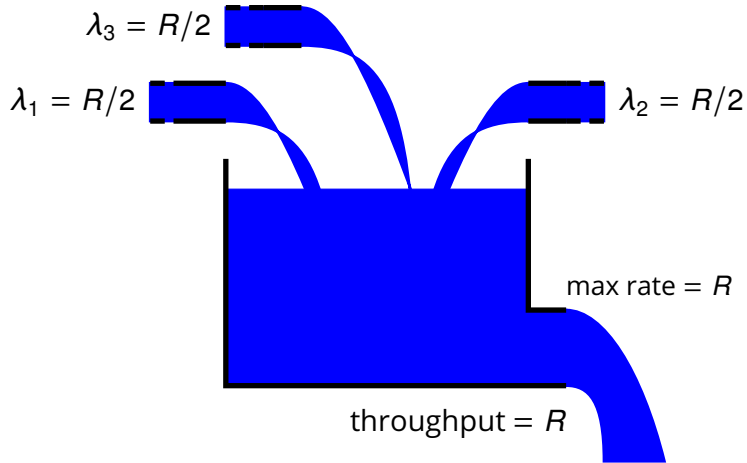
Understanding Congestion

- A router behaves a lot like a kitchen sink



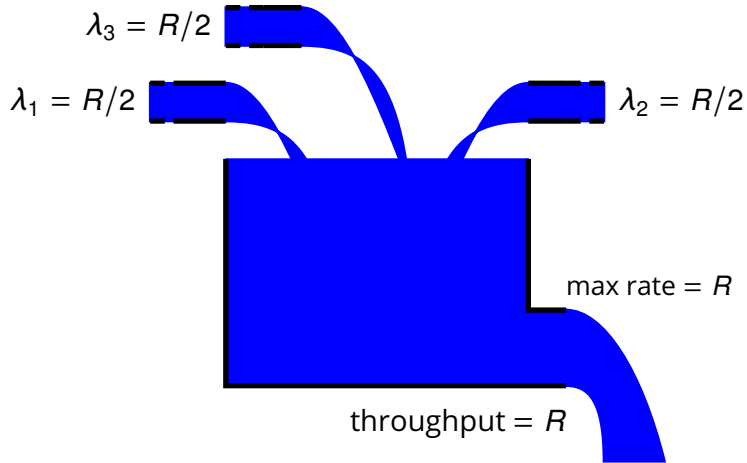
Understanding Congestion

- A router behaves a lot like a kitchen sink



Understanding Congestion

- A router behaves a lot like a kitchen sink



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

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

- 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)

- 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

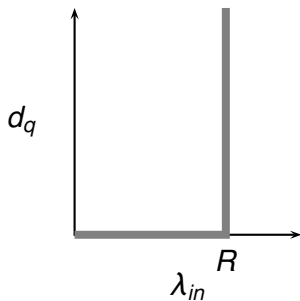
$$d_q = \frac{\lambda_{in} - R}{R}t$$

- Steady-state queuing delay

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

- Steady-state queuing delay

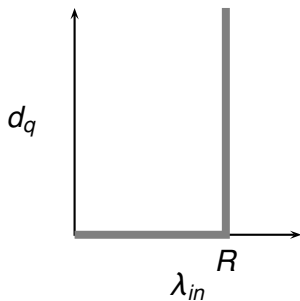
$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in} - R}{R} t & \lambda_{in} > R \end{cases}$$



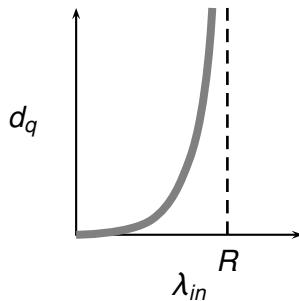
ideal input flow
 λ_{in} constant

■ 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
 λ_{in} constant

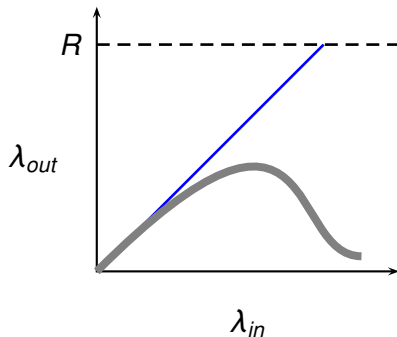


realistic input flow
 λ_{in} variable

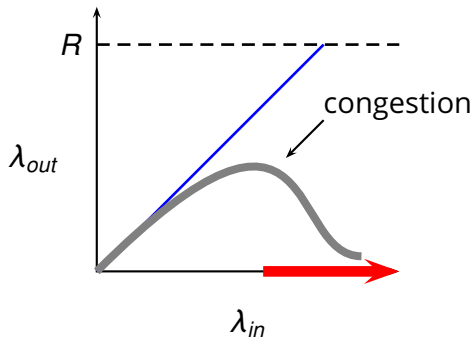
- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays

- **Conclusion:** as the input rate λ_{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

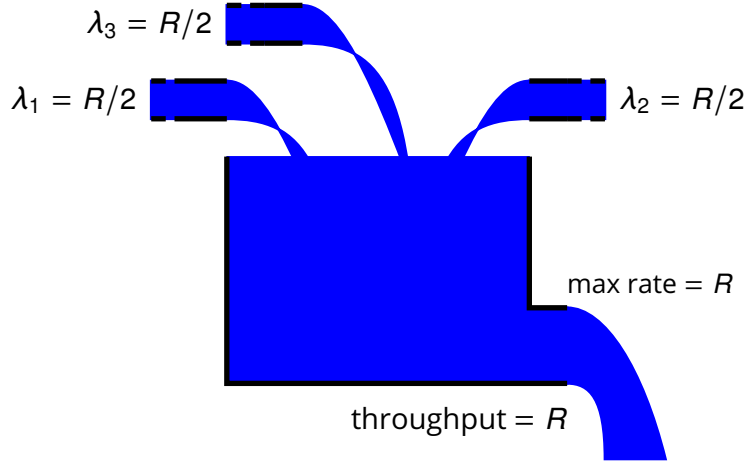
- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays
- More realistic assumptions and models
 - ▶ finite queue length (buffers) in routers
 - ▶ effects of retransmission overhead
 - ▶ full queues along multi-hops paths



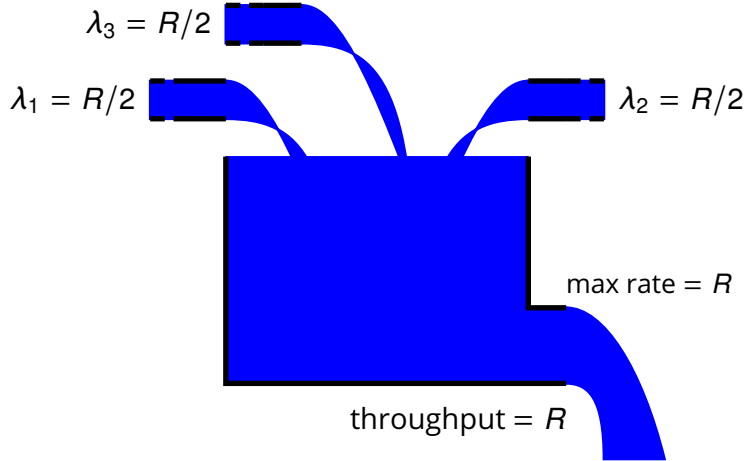
- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays
- More realistic assumptions and models
 - ▶ finite queue length (buffers) in routers
 - ▶ effects of retransmission overhead
 - ▶ full queues along multi-hops paths



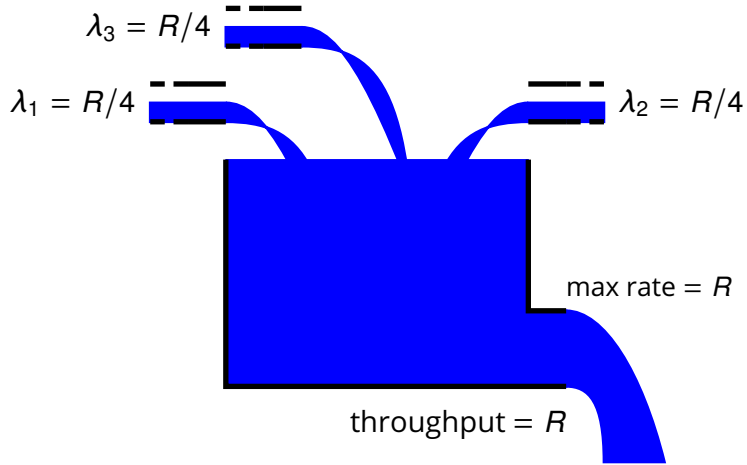
- What to do when the network is congested?



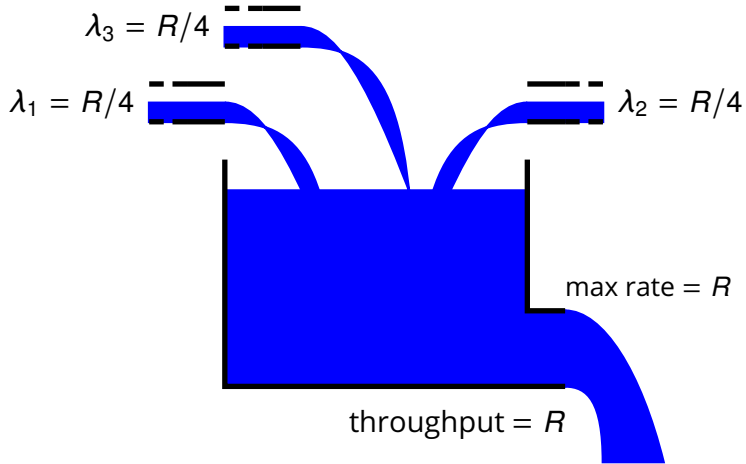
- What to do when the network is congested? **BACK OFF!**



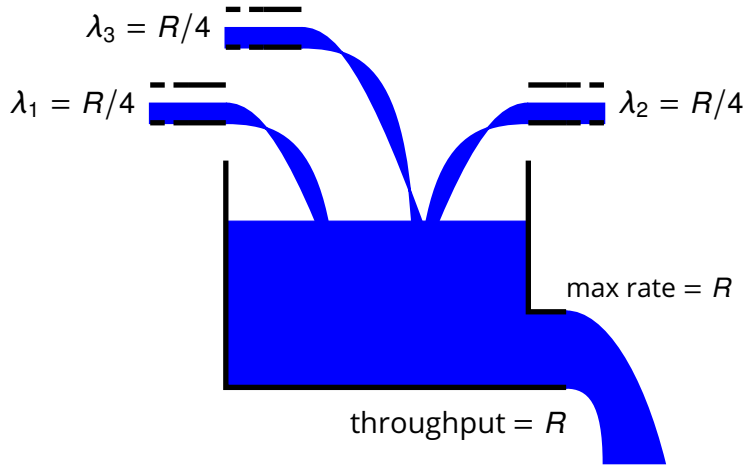
- What to do when the network is congested? **BACK OFF!**



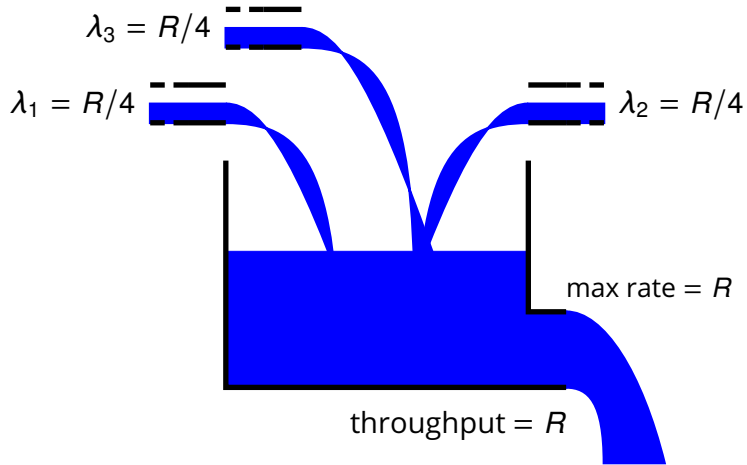
- What to do when the network is congested? **BACK OFF!**



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



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



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 (λ_{in})

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 (λ_{in})

Ingredients:

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

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 (λ_{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

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 (λ_{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 (Accelerator/Brakes)

- The sender maintains a *congestion window* W

Congestion Window (Accelerator/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 (Accelerator/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

$$LastByteSent - LastByteAked \leq W$$

where

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

Congestion Window (Accelerator/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

$$LastByteSent - LastByteAked \leq W$$

where

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

- 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

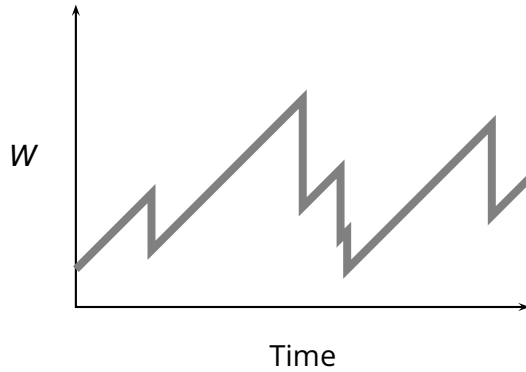
- **How W is reduced:** at every *loss* event, TCP halves the congestion window
 - ▶ e.g., suppose the window size W is currently 20Kb, and a loss is detected
 - ▶ TCP reduces W to 10Kb
- **How W is increased:** at every (good) acknowledgment, TCP increments W by $1MSS/W$, so as to increase W by MSS 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

Additive-Increase/Multiplicative-Decrease

■ Window size W over time



- What is the initial value of W ?

- 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

- 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*)

- 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

- 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

- 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*

- 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

- 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

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

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*

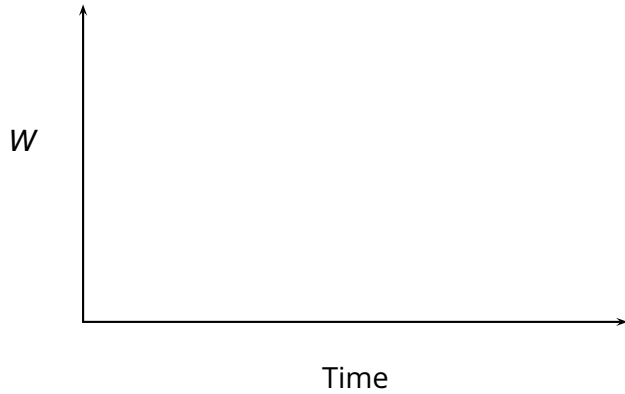
Assuming the current window size is $W = \overline{W}$

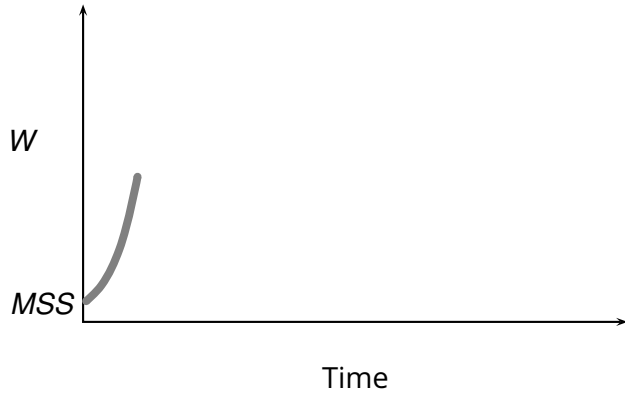
■ *Timeout*

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

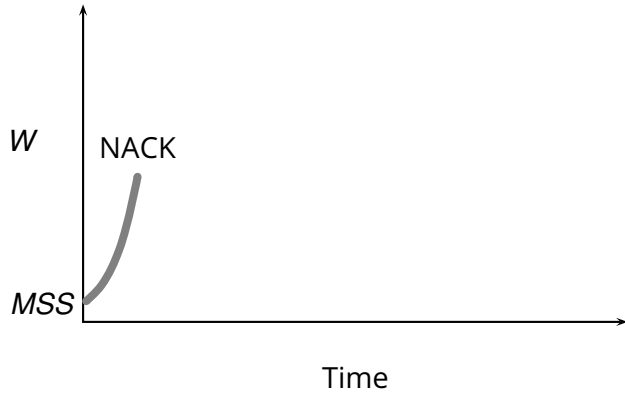
■ NACK (i.e., triple duplicate-ack)

- ▶ 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***

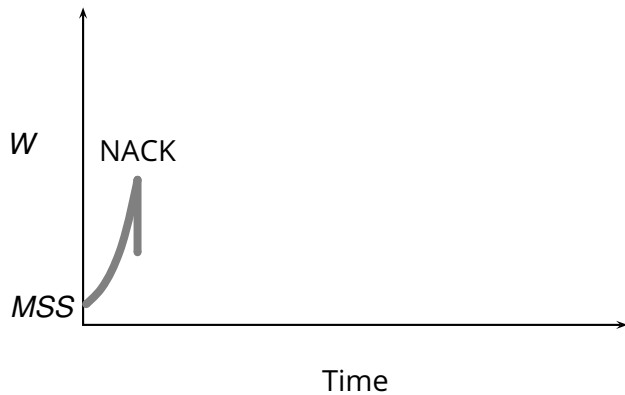




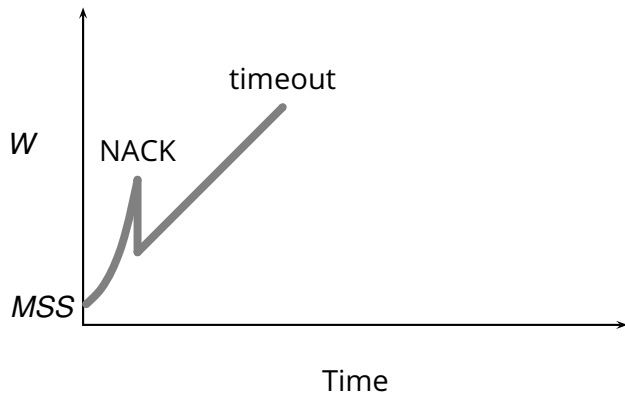
Sender Behavior



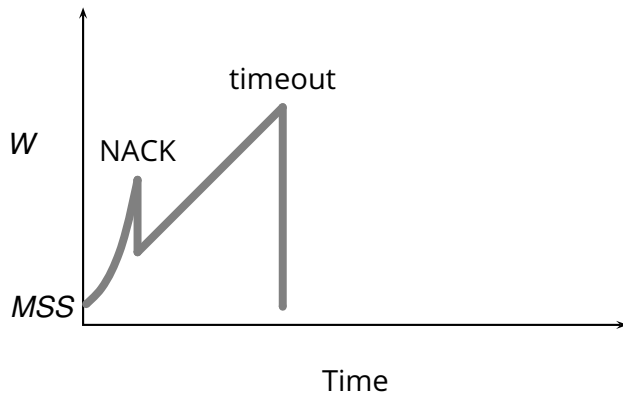
Sender Behavior



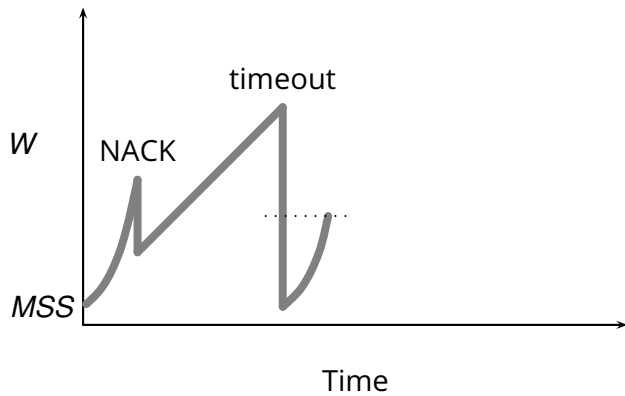
Sender Behavior



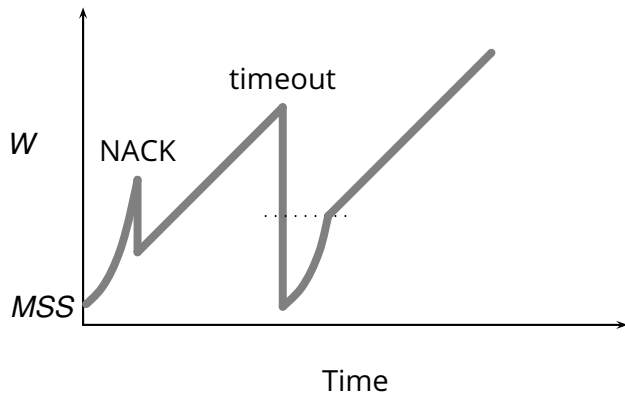
Sender Behavior



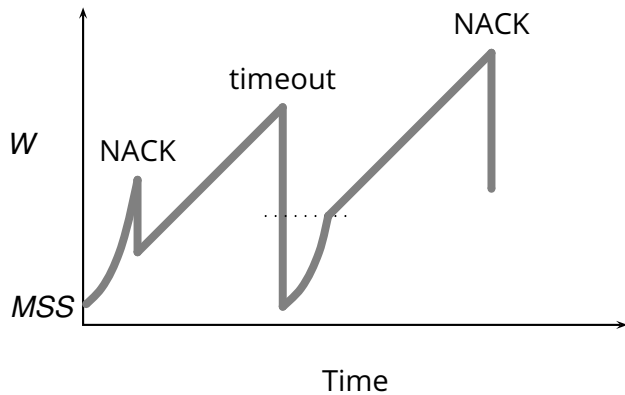
Sender Behavior



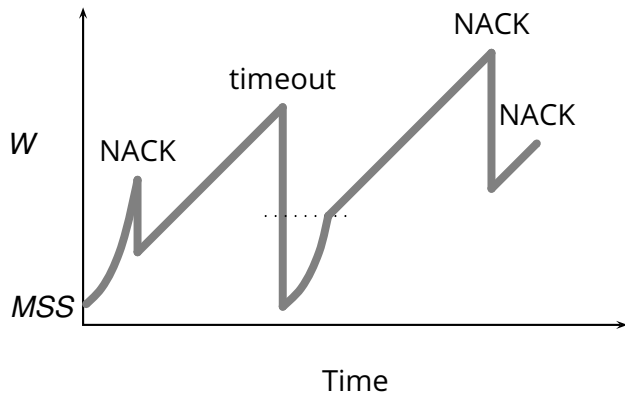
Sender Behavior



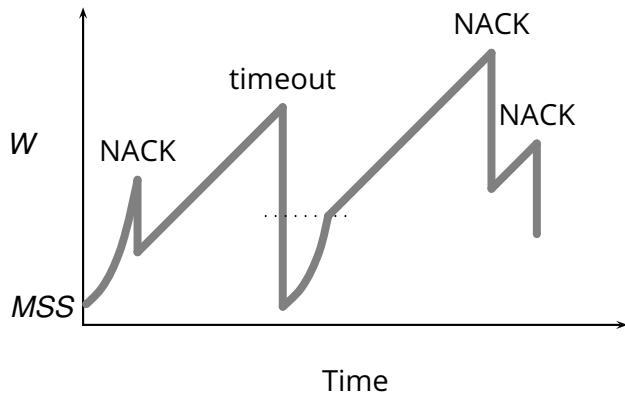
Sender Behavior



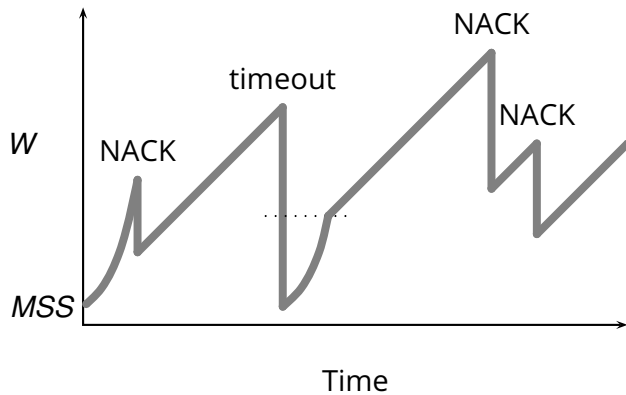
Sender Behavior



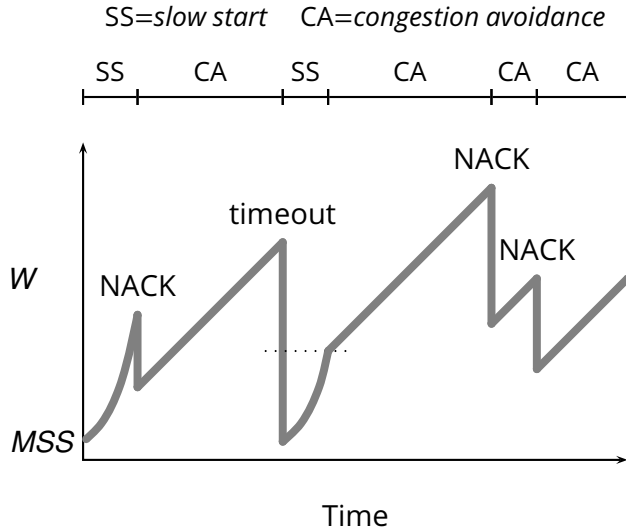
Sender Behavior



Sender Behavior



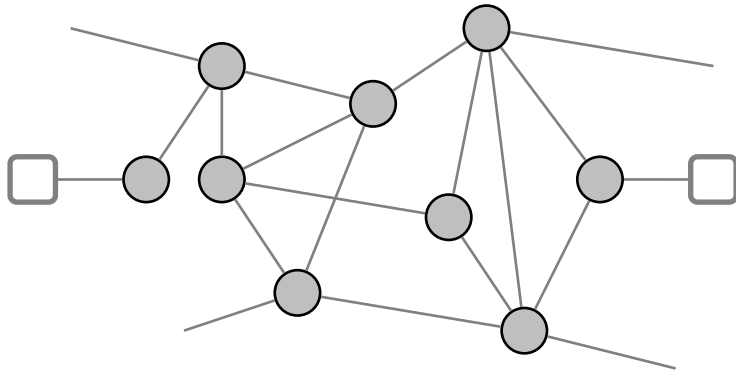
Sender Behavior



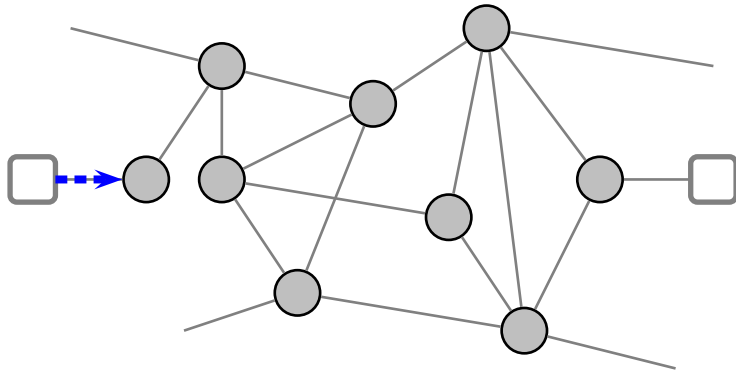
Part VII

Network Layer

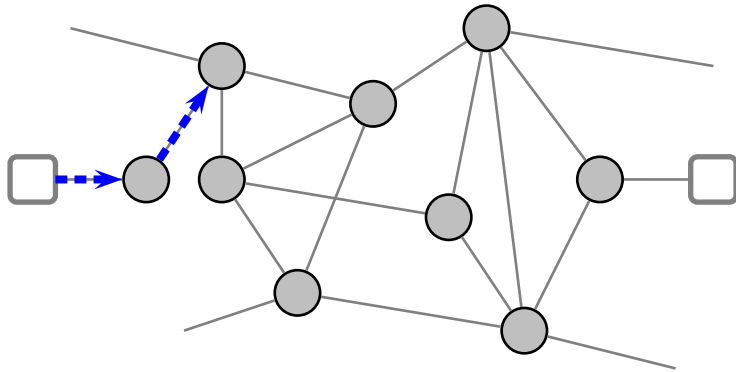
Datagram Network



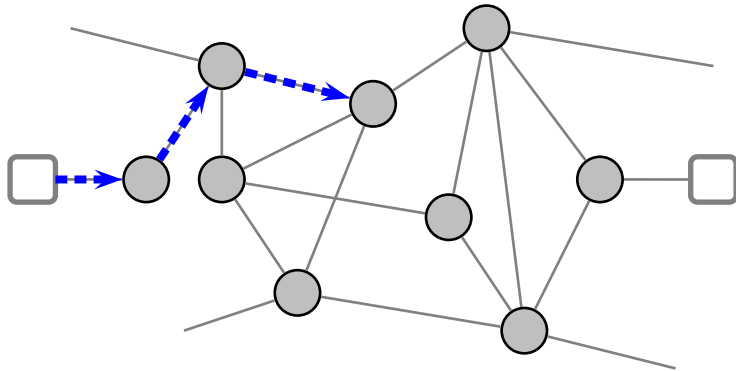
Datagram Network



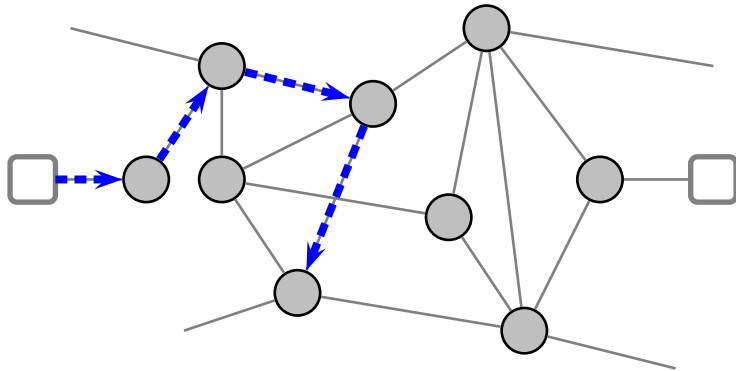
Datagram Network



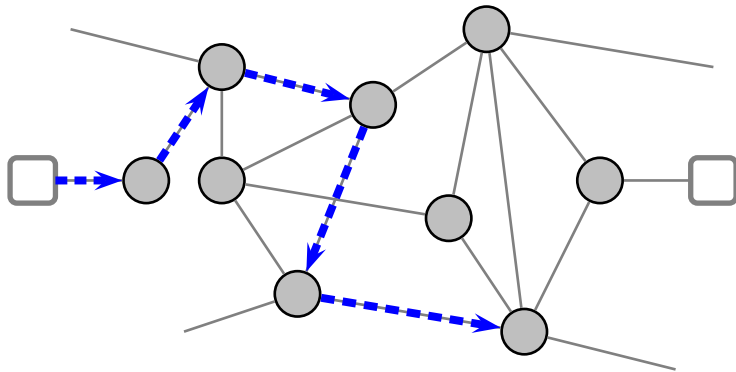
Datagram Network



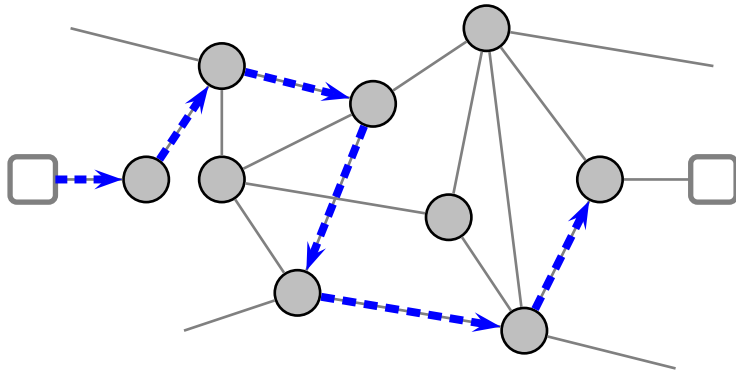
Datagram Network



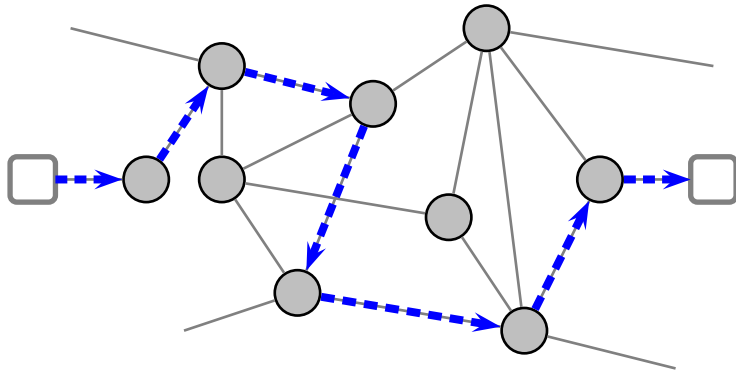
Datagram Network



Datagram Network

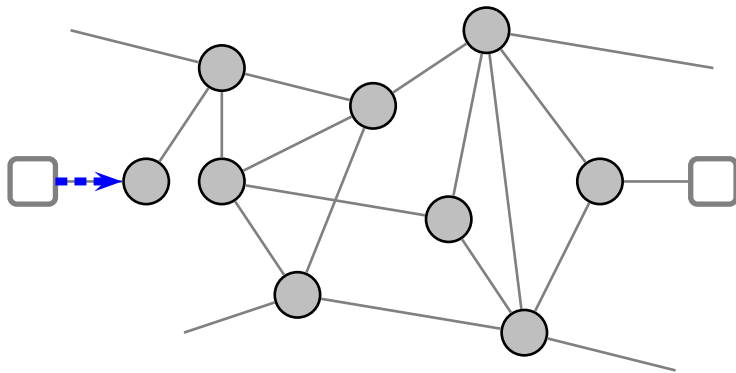


Datagram Network



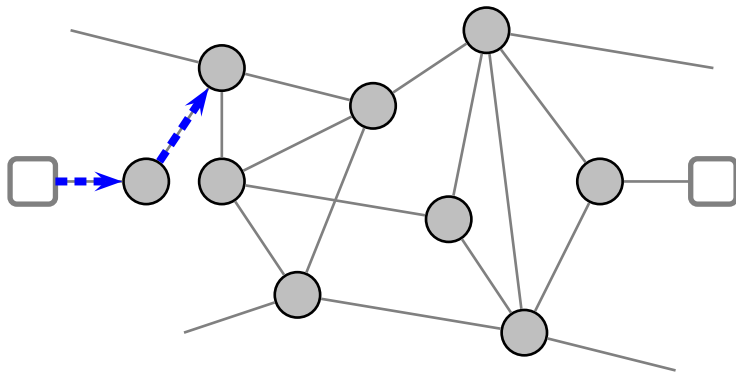
- Potentially *multiple paths* for the same source/destination

Datagram Network



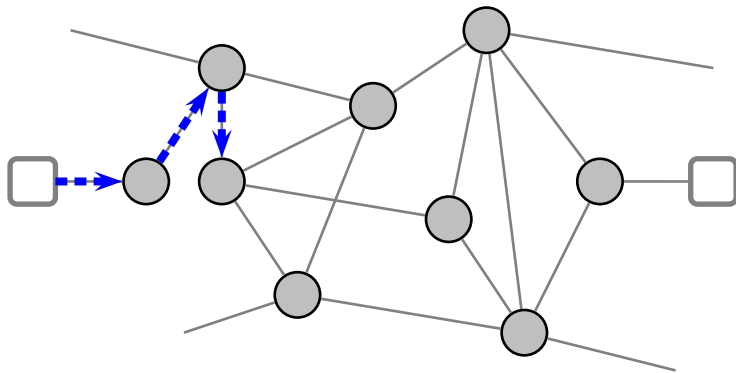
- Potentially *multiple paths* for the same source/destination

Datagram Network



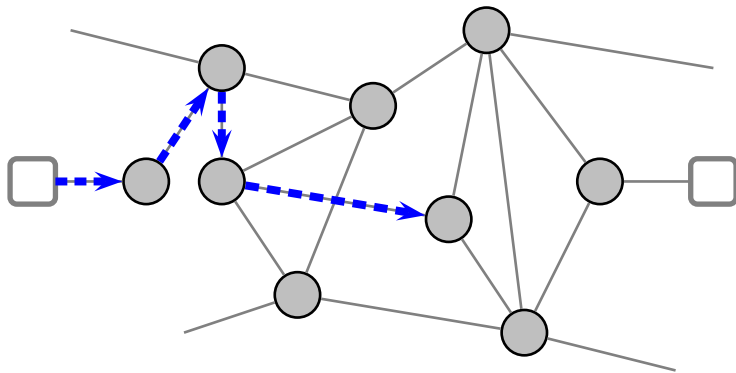
- Potentially *multiple paths* for the same source/destination

Datagram Network



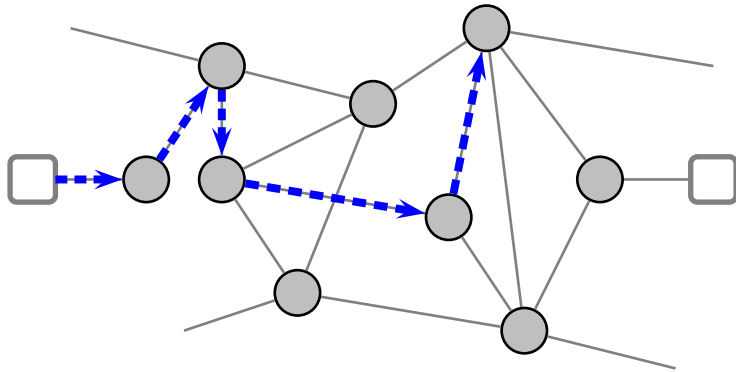
- Potentially *multiple paths* for the same source/destination

Datagram Network



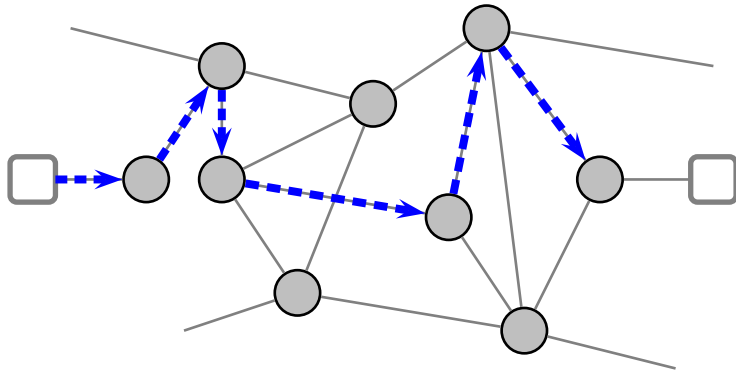
- Potentially *multiple paths* for the same source/destination

Datagram Network



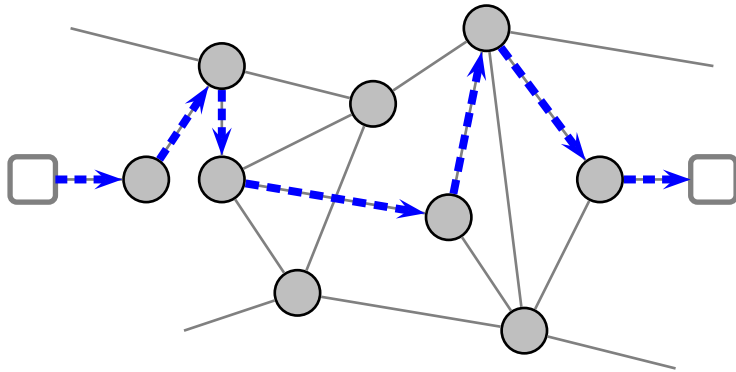
- Potentially *multiple paths* for the same source/destination

Datagram Network



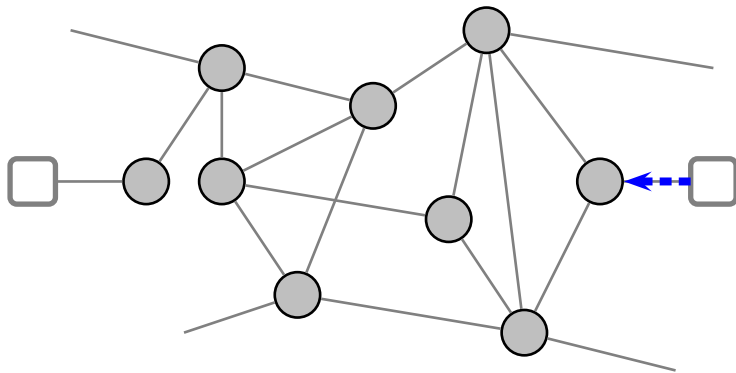
- Potentially *multiple paths* for the same source/destination

Datagram Network



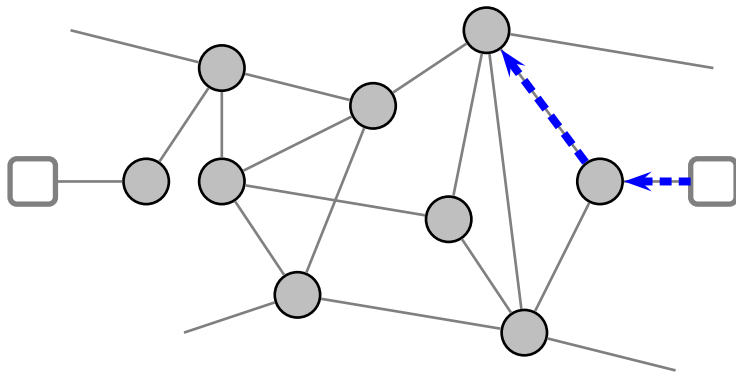
- Potentially *multiple paths* for the same source/destination

Datagram Network



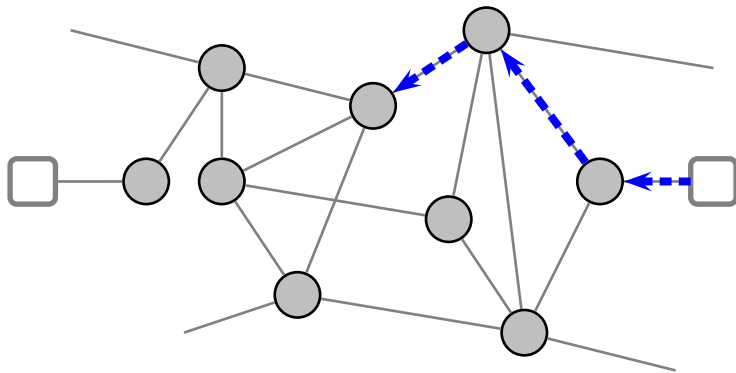
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



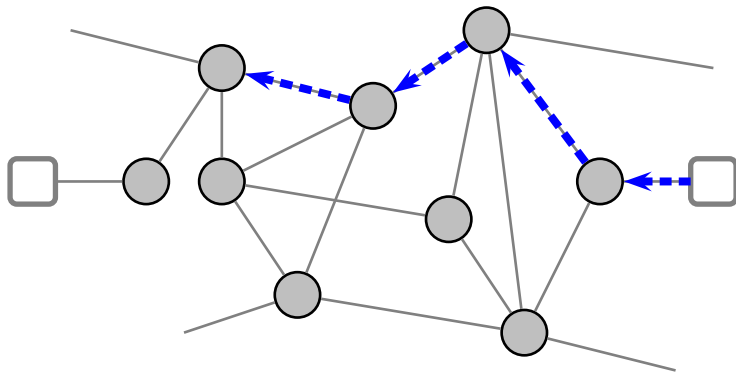
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



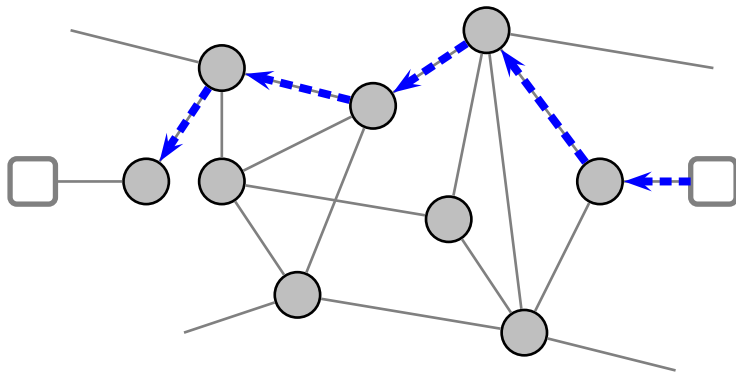
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



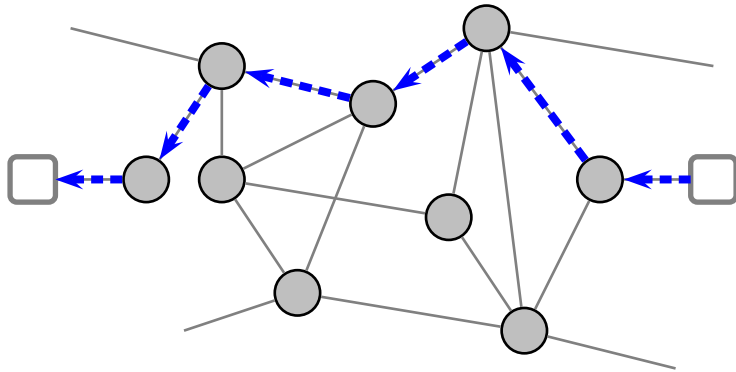
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



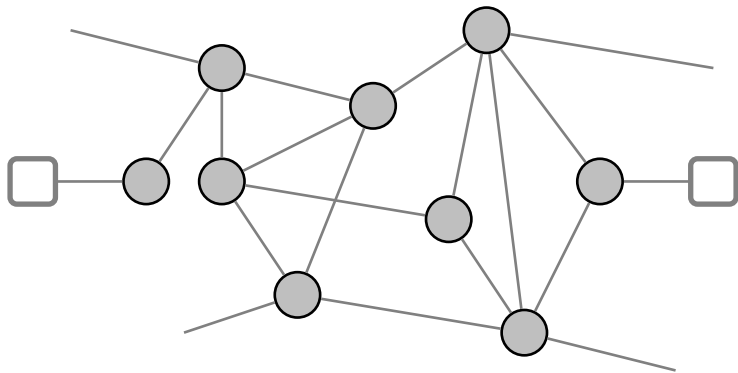
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



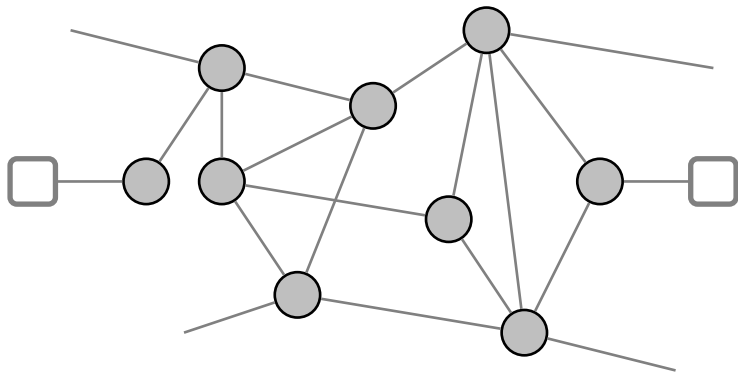
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*
- No connection, each packet handled independently

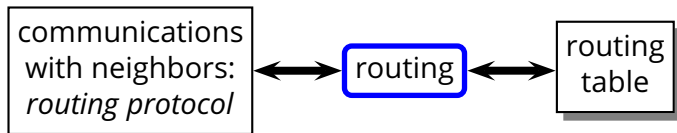
Datagram Network



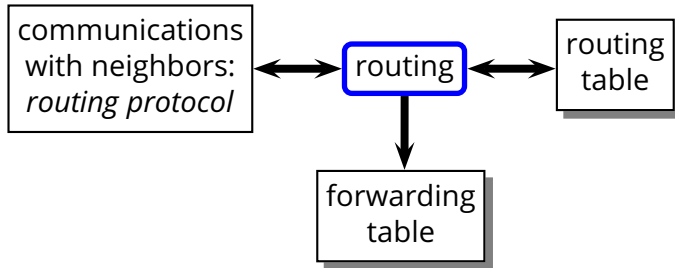
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*
- No connection, each packet handled independently
- No connection, each packet handled independently

Network/Router Functions

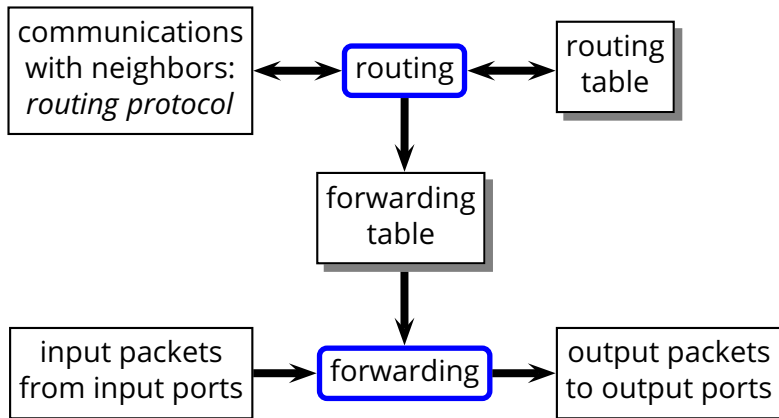
Network/Router Functions



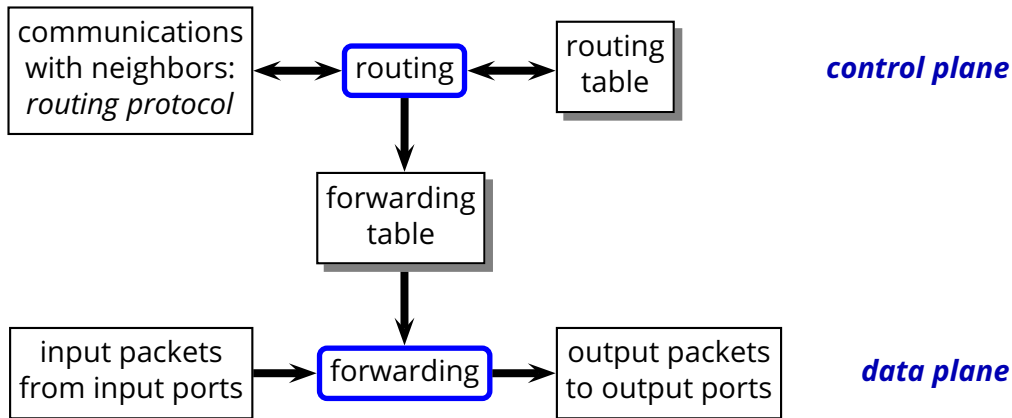
Network/Router Functions

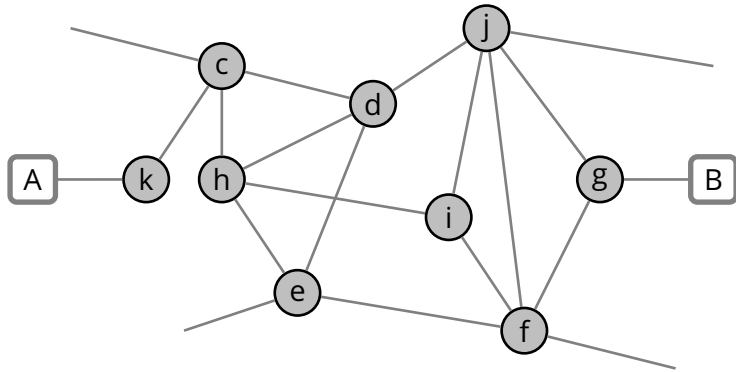


Network/Router Functions

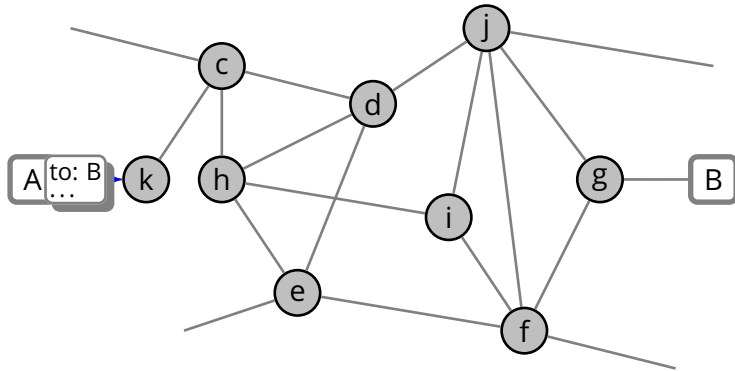


Network/Router Functions

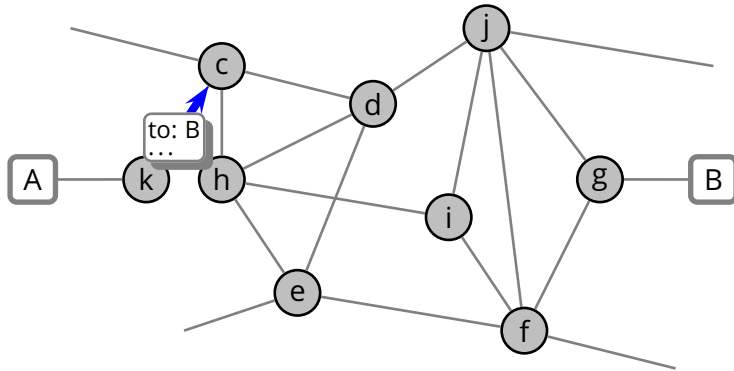




- A sends a datagram to B

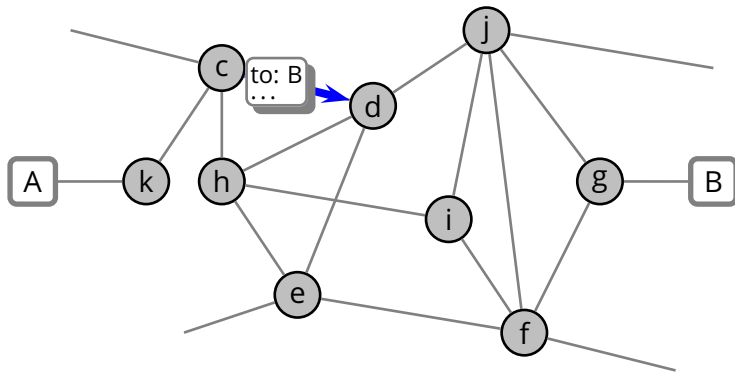


- A sends a datagram to *B*
- The datagram is **forwarded** towards *B*

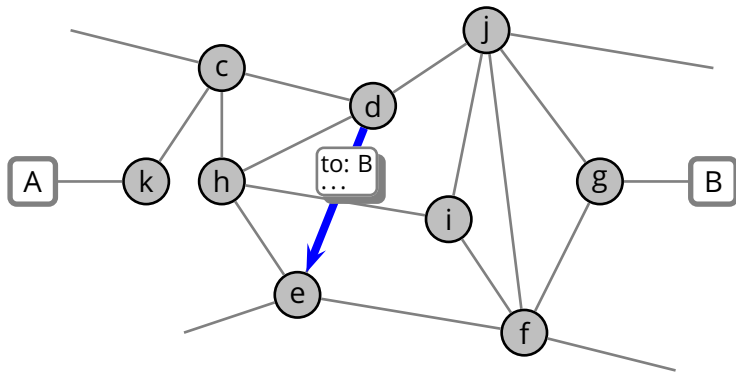


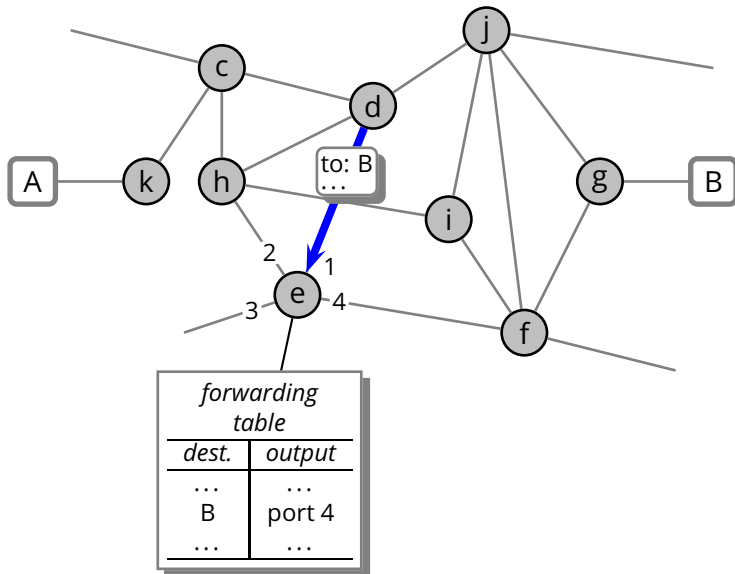
- A sends a datagram to *B*
- The datagram is **forwarded** towards *B*

Forwarding

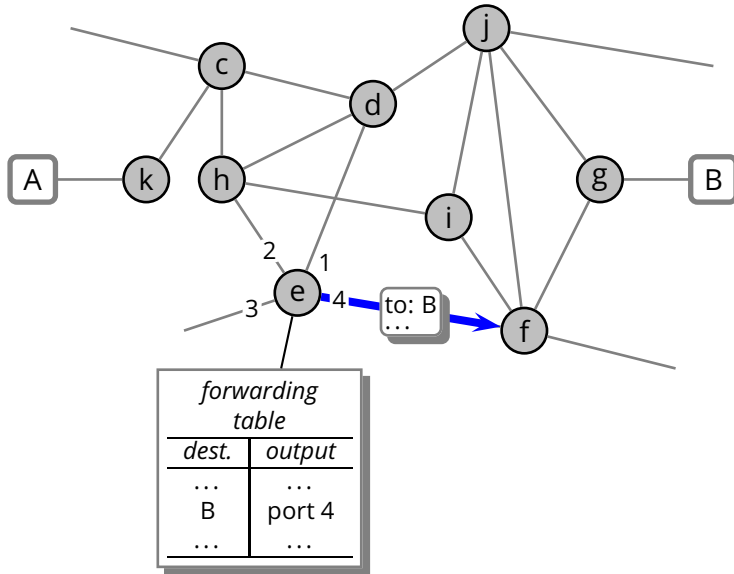


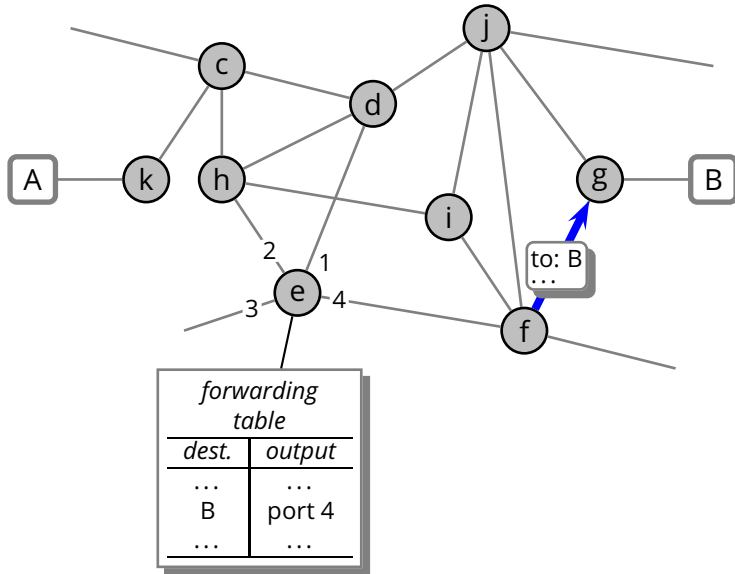
Forwarding

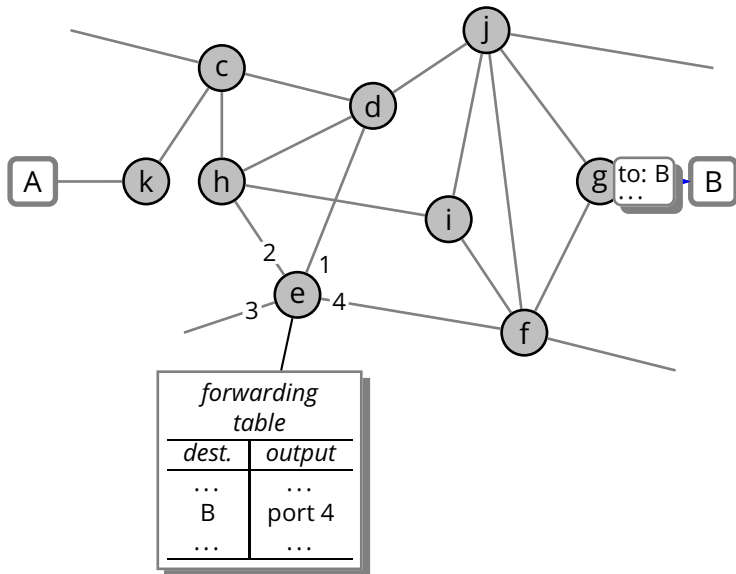




Forwarding







- *Input:* datagram destination

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

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

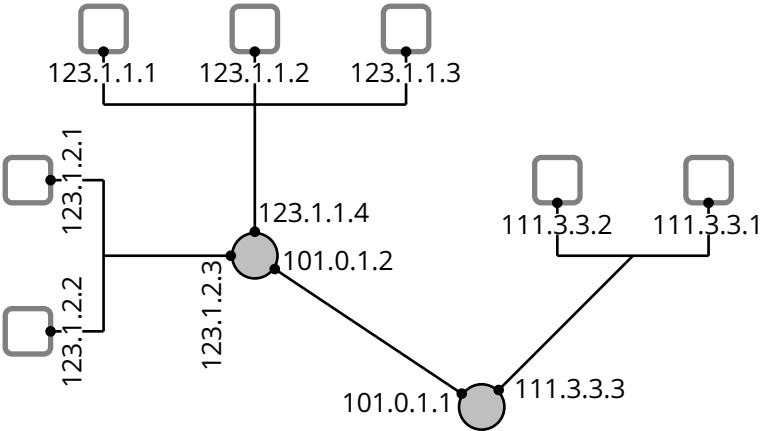
- *Input*: datagram destination
- *Output*: output port
- Simple design: “forwarding table”
- Issues

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

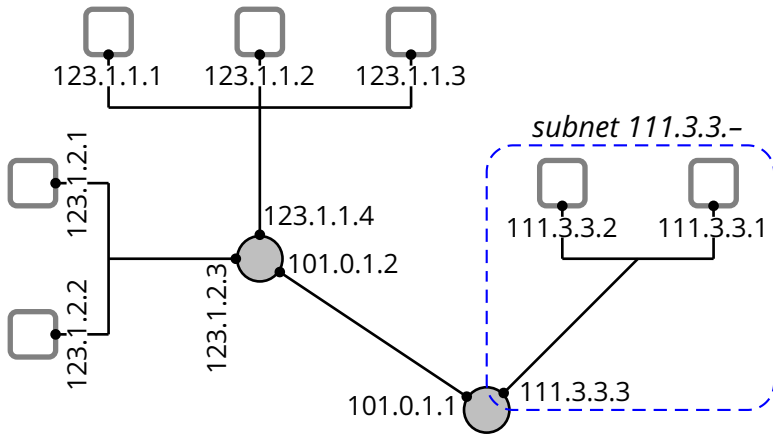
- *Input*: datagram destination
- *Output*: output port
- Simple design: “forwarding table”
- Issues
 - ▶ how big is the forwarding table?
 - ▶ how fast does the router have to forward datagrams?

- *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?

Interconnection of Networks



Interconnection of Networks



- 32-bit *addresses*

- 32-bit *addresses*
- An IP address is associated with an *interface*, not a host
 - ▶ a host with more than one interface may have more than one IP address

- 32-bit *addresses*
- An IP address is associated with an *interface*, not a host
 - ▶ a host with more than one interface may have more than one IP address
- The assignment of addresses over an Internet topology is crucial to limit the complexity of routing and forwarding

- 32-bit *addresses*
- An IP address is associated with an *interface*, not a host
 - ▶ a host with more than one interface may have more than one IP address
- The assignment of addresses over an Internet topology is crucial to limit the complexity of routing and forwarding
- The key idea is to assign addresses with the *same prefix* to interfaces that are on the *same subnet*

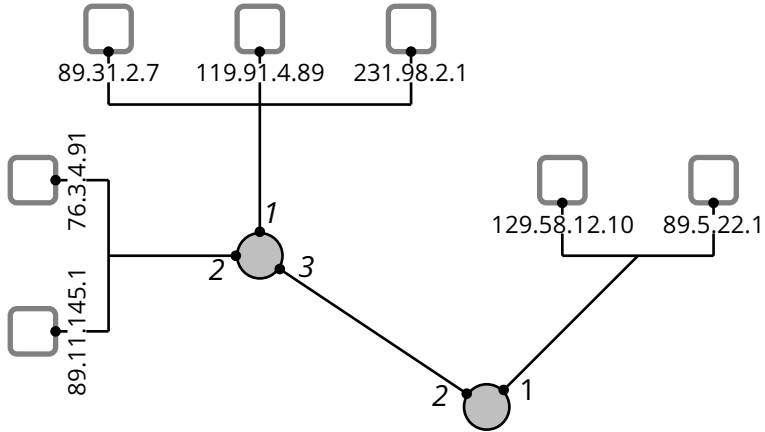
- 32-bit *addresses*
- An IP address is associated with an *interface*, not a host
 - ▶ a host with more than one interface may have more than one IP address
- The assignment of addresses over an Internet topology is crucial to limit the complexity of routing and forwarding
- The key idea is to assign addresses with the *same prefix* to interfaces that are on the *same subnet*
- Why is the idea of the common prefix so important?

- 32-bit *addresses*
- An IP address is associated with an *interface*, not a host
 - ▶ a host with more than one interface may have more than one IP address
- The assignment of addresses over an Internet topology is crucial to limit the complexity of routing and forwarding
- The key idea is to assign addresses with the *same prefix* to interfaces that are on the *same subnet*
- Why is the idea of the common prefix so important?

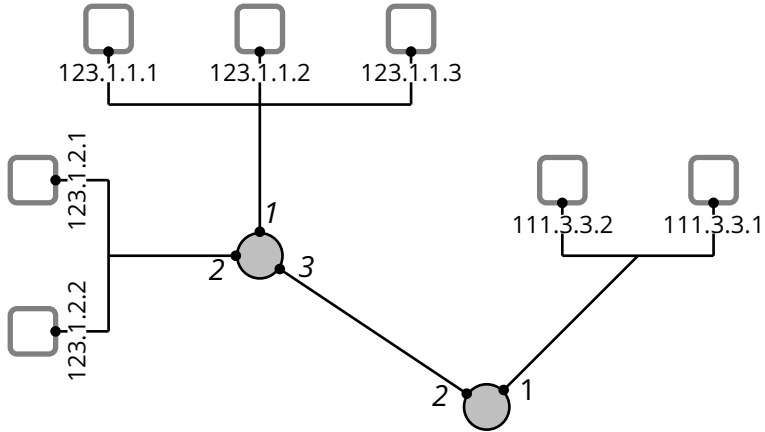
Because it compresses the forwarding tables by an exponential factor!

 - ▶ there might be some 64 thousands hosts in 128.138.-.-
but they all appear as *one table entry* from the outside

Example: Bad Address Allocation



Example: Good Address Allocation



Classless Interdomain Routing

Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
 - ▶ e.g., in the previous example we have 123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—

Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
 - ▶ e.g., in the previous example we have
123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—
- Network addresses prefix-length notation: ***address/prefix-length***

Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
 - ▶ e.g., in the previous example we have
123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—
- Network addresses prefix-length notation: ***address/prefix-length***
 - ▶ e.g., 123.1.1.0/24, 123.1.1.0/24, 101.0.1.0/24, and 111.3.3.0/24

Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
 - ▶ e.g., in the previous example we have 123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—
- Network addresses prefix-length notation: ***address/prefix-length***
 - ▶ e.g., 123.1.1.0/24, 123.1.1.0/24, 101.0.1.0/24, and 111.3.3.0/24
 - ▶ 123.1.1.0/24 means that all the addresses share the same leftmost 24 bits with address 123.1.1.0

Classless Interdomain Routing

- All interfaces in the same subnet share the same *address prefix*
 - ▶ e.g., in the previous example we have 123.1.1.—, 123.1.2.—, 101.0.1.—, and 111.3.3.—
- Network addresses prefix-length notation: ***address/prefix-length***
 - ▶ e.g., 123.1.1.0/24, 123.1.1.0/24, 101.0.1.0/24, and 111.3.3.0/24
 - ▶ 123.1.1.0/24 means that all the addresses share the same leftmost 24 bits with address 123.1.1.0
- This addressing scheme is not limited to entire bytes. For example, a network address might be 128.138.207.160/27
 - ▶ as opposed to the original scheme which divided the address space in “classes”

<i>address class</i>	<i>prefix length</i>
A	8
B	16
C	24

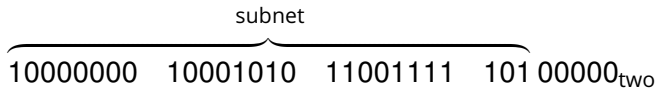
- Network address 128.138.207.160/27

- Network address 128.138.207.160/27

subnet

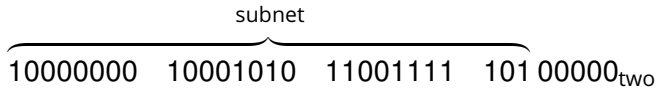
10000000 10001010 11001111 10100000_{two}

- Network address 128.138.207.160/27



128.138.207.185?

- Network address 128.138.207.160/27



128.138.207.185?

10000000 10001010 11001111 10111001_{two}

- Network address 128.138.207.160/27

subnet

$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 10100000}_{\text{two}}$

128.138.207.185?

10000000 10001010 11001111 10111001_{two}

128.138.207.98?

- Network address 128.138.207.160/27

subnet

$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 10100000}^{\text{subnet}}_{\text{two}}$

128.138.207.185?

10000000 10001010 11001111 10111001_{two}

128.138.207.98?

10000000 10001010 11001111 01100010_{two}

- Network address 128.138.207.160/27

subnet

$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 10100000}_{\text{two}}$

128.138.207.185?

10000000 10001010 11001111 10111001_{two}

128.138.207.98?

10000000 10001010 11001111 01100010_{two}

128.138.207.194?

- Network address 128.138.207.160/27

subnet

$\overbrace{10000000 \quad 10001010 \quad 11001111 \quad 10100000}_{\text{two}}$

128.138.207.185?

10000000 10001010 11001111 10111001_{two}

128.138.207.98?

10000000 10001010 11001111 01100010_{two}

128.138.207.194?

10000000 10001010 11001111 11000010_{two}

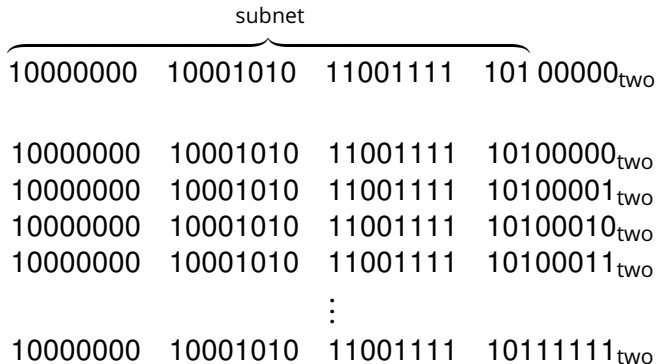
- What is the range of addresses in 128.138.207.160/27?

- What is the range of addresses in 128.138.207.160/27?

subnet

10000000 10001010 11001111 10100000_{two}

- What is the range of addresses in 128.138.207.160/27?



- What is the range of addresses in 128.138.207.160/27?

subnet			
10000000	10001010	11001111	101 00000 _{two}
10000000	10001010	11001111	10100000 _{two}
10000000	10001010	11001111	10100001 _{two}
10000000	10001010	11001111	10100010 _{two}
10000000	10001010	11001111	10100011 _{two}
		⋮	
10000000	10001010	11001111	10111111 _{two}

128.138.207.160–128.138.207.191

- Network addresses, *mask* notation: ***address/mask***

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
- ▶ 127.0.0.1/8=?

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
- ▶ 127.0.0.1/8=127.0.0.1/255.0.0.0

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
- ▶ 127.0.0.1/8=127.0.0.1/255.0.0.0
- ▶ 192.168.0.3/24=?

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., $128.138.207.160/27=128.138.207.160/255.255.255.224$
- ▶ $127.0.0.1/8=127.0.0.1/255.0.0.0$
- ▶ $192.168.0.3/24=192.168.0.3/255.255.255.0$

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
- ▶ 127.0.0.1/8=127.0.0.1/255.0.0.0
- ▶ 192.168.0.3/24=192.168.0.3/255.255.255.0
- ▶ 195.176.181.11/32=?

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
- ▶ 127.0.0.1/8=127.0.0.1/255.0.0.0
- ▶ 192.168.0.3/24=192.168.0.3/255.255.255.0
- ▶ 195.176.181.11/32=195.176.181.11/255.255.255.255

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
 - ▶ 127.0.0.1/8=127.0.0.1/255.0.0.0
 - ▶ 192.168.0.3/24=192.168.0.3/255.255.255.0
 - ▶ 195.176.181.11/32=195.176.181.11/255.255.255.255
- In Java:

- Network addresses, *mask* notation: ***address/mask***
- A prefix of length p corresponds to a mask

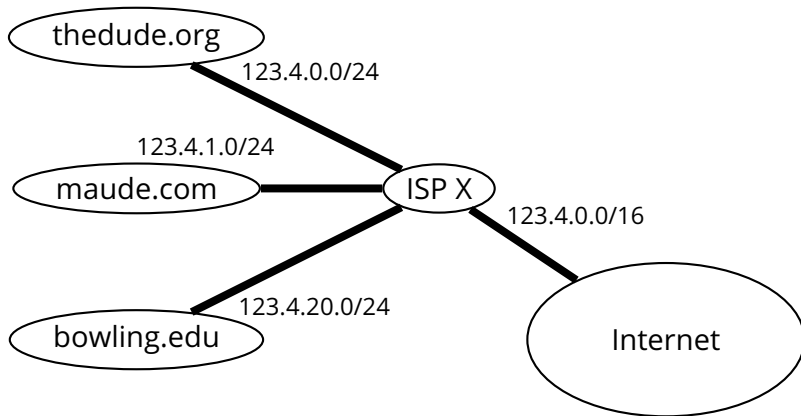
$$M = \overbrace{11 \cdots 1}^{p \text{ times}} \overbrace{00 \cdots 0}^{32-p \text{ times}}_{\text{two}}$$

- ▶ e.g., 128.138.207.160/27=128.138.207.160/255.255.255.224
 - ▶ 127.0.0.1/8=127.0.0.1/255.0.0.0
 - ▶ 192.168.0.3/24=192.168.0.3/255.255.255.0
 - ▶ 195.176.181.11/32=195.176.181.11/255.255.255.255
- In Java:

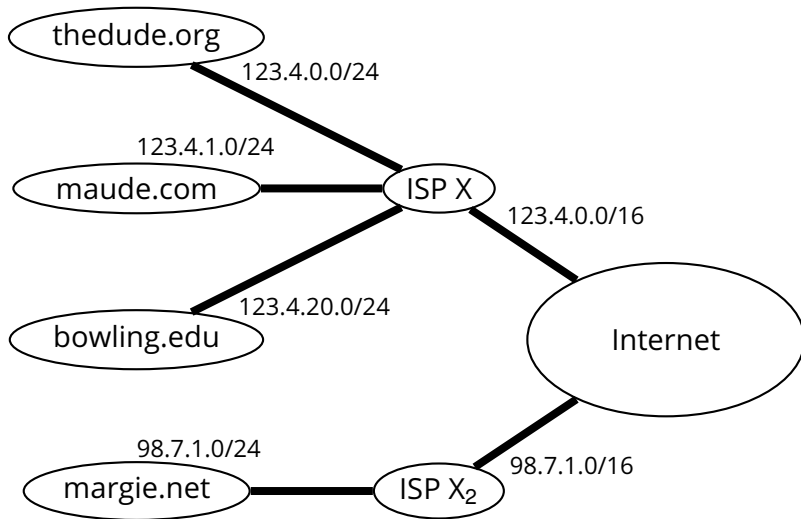
```
boolean match(int address, int network, int mask) {  
    return (address & mask) == (network & mask);  
}
```

Allocation of Address Blocks

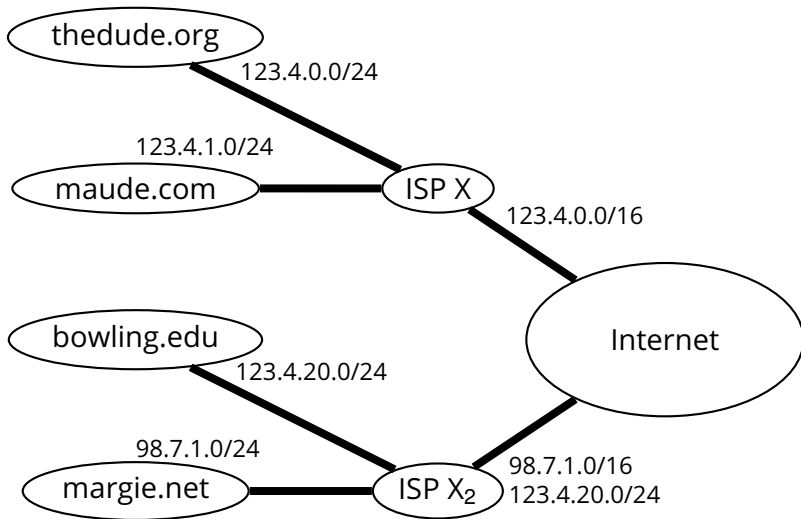
Allocation of Address Blocks



Allocation of Address Blocks



Allocation of Address Blocks



Longest-Prefix Matching

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69→?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → ?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → ?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → ?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3
- ▶ 128.138.207.167 → ?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3
- ▶ 128.138.207.167 → 4

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3
- ▶ 128.138.207.167 → 4
- ▶ 123.4.20.11 → ?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3
- ▶ 128.138.207.167 → 4
- ▶ 123.4.20.11 → 2

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3
- ▶ 128.138.207.167 → 4
- ▶ 123.4.20.11 → 2
- ▶ 123.4.21.10 → ?

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

Longest-Prefix Matching

- In choosing where to forward a datagram, a router chooses the entry that matches the destination address with the longest prefix

E.g.,

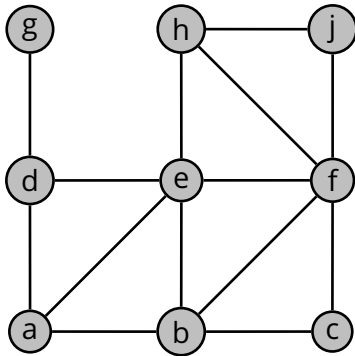
- ▶ 123.4.1.69 → 1
- ▶ 68.142.226.44 → 4
- ▶ 98.7.2.71 → 2
- ▶ 200.100.2.1 → 3
- ▶ 128.138.207.167 → 4
- ▶ 123.4.20.11 → 2
- ▶ 123.4.21.10 → 1

<i>forwarding table</i>	
<i>network</i>	<i>port</i>
123.4.0.0/16	1
98.7.1.0/16	2
123.4.20.0/24	2
128.0.0.0/1	3
66.249.0.0/16	3
0.0.0.0/1	4
128.138.0.0/16	4

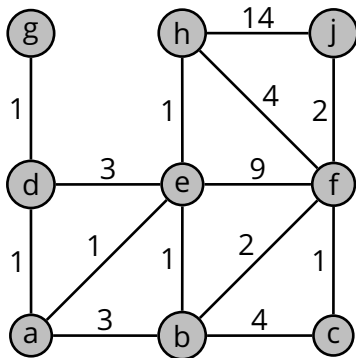
Routing Problem

- Finding paths through a network

- Finding paths through a network



- Finding paths through a network



- Example: $a \rightarrow j$?

- The network is modeled as a graph

$$G = (V, E)$$

- The network is modeled as a graph

$$G = (V, E)$$

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

- 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

- 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 $u, v \in N$

- 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 $c : E \rightarrow \mathbb{R}$
 - ▶ 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 $P_{u \rightarrow v} = u, x_1, x_2, \dots, x_n, v$ such that

Routing in the Graph Model

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

Routing in the Graph Model

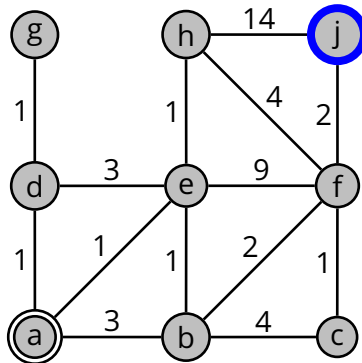
- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \rightarrow v} = u, x_1, x_2, \dots, x_n, v$ such that
 - ▶ $P_{u \rightarrow v}$ is completely contained in the network graph G . I.e., $(u, x_1) \in V, (x_1, x_2) \in V, \dots, (x_n, v) \in V$
 - ▶ $P_{u \rightarrow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

Routing in the Graph Model

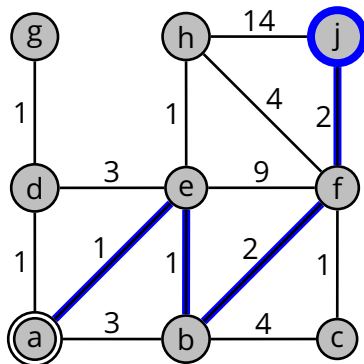
- For every router $u \in V$, for every other router $v \in V$, compute the path $P_{u \rightarrow v} = u, x_1, x_2, \dots, x_n, v$ such that
 - ▶ $P_{u \rightarrow v}$ is completely contained in the network graph G . I.e., $(u, x_1) \in V, (x_1, x_2) \in V, \dots, (x_n, v) \in V$
 - ▶ $P_{u \rightarrow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

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

- ▶ $A(v)$ 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 $P_{u \rightarrow v} = u, x_1, x_2, \dots, x_n, v$

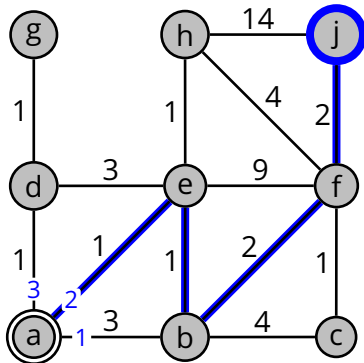


■ Example: $a \rightarrow j$



■ Example: $a \rightarrow j$

▶ least-cost path is $P_{a \rightarrow j} = a, e, b, f, j$



■ Example: $a \rightarrow j$

▶ least-cost path is $P_{a \rightarrow j} = a, e, b, f, j$

▶ a 's forwarding table will contain an entry $j \rightarrow 2$ 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

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

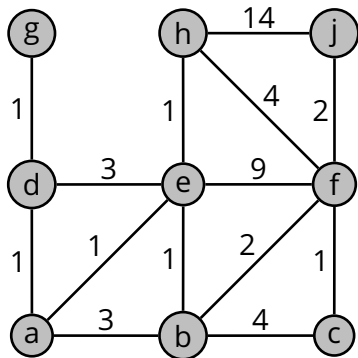
- Router u maintains a complete view of the network graph G (including all links and their costs)
 - ▶ every router v advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
 - ▶ ***link-state advertisements (LSAs)*** are broadcast through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

- 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

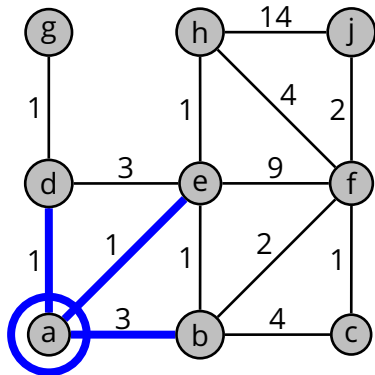
- Router u maintains a complete view of the network graph G (including all links and their costs)
 - ▶ every router v advertises its adjacent links (their costs) to every other router in the network; this information is called *link state*
 - ▶ ***link-state advertisements (LSAs)*** are broadcast through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

- 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

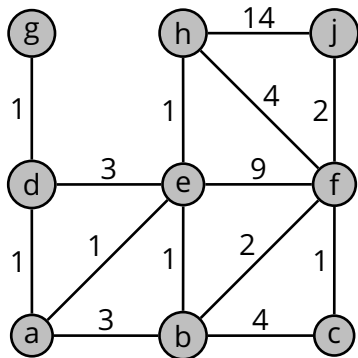


Link-State Advertisements

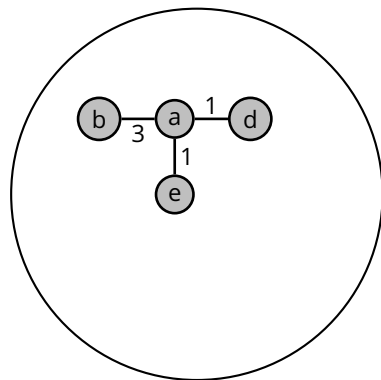


$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$

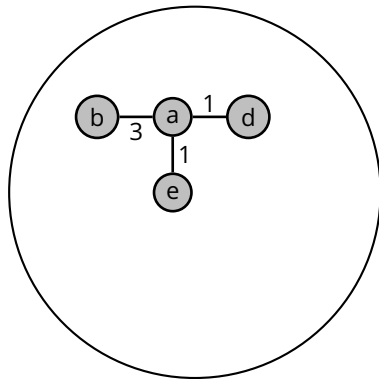
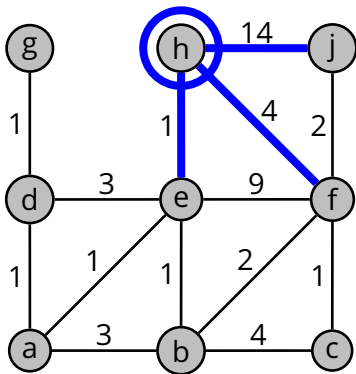
Link-State Advertisements



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



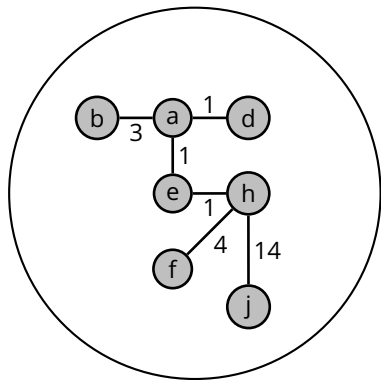
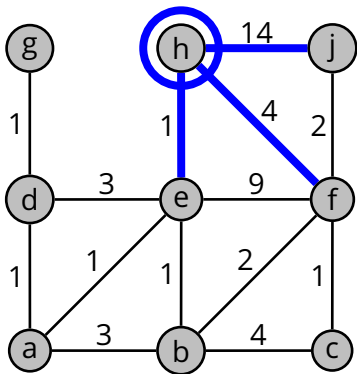
Link-State Advertisements



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

$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$

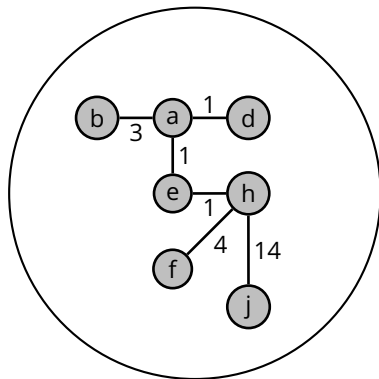
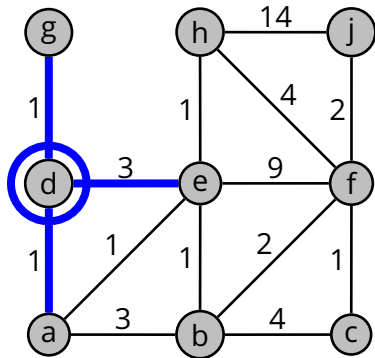
Link-State Advertisements



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

$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$

Link-State Advertisements

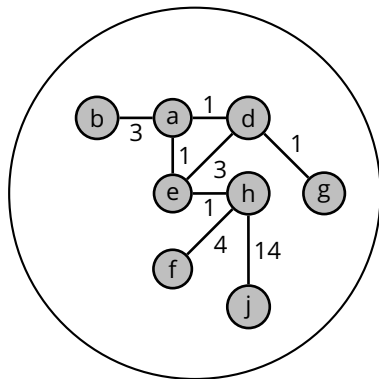
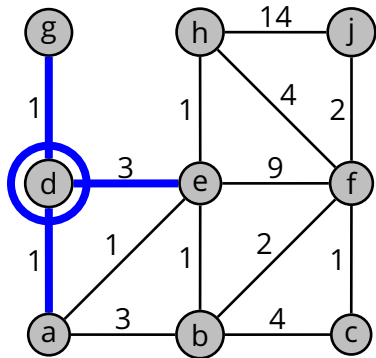


$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$

$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$

$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$

Link-State Advertisements

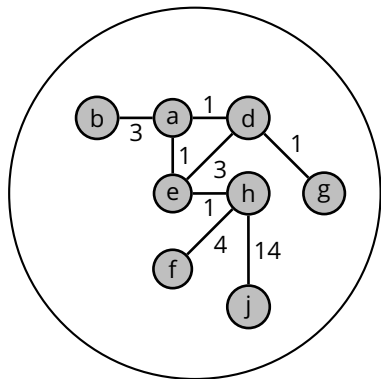
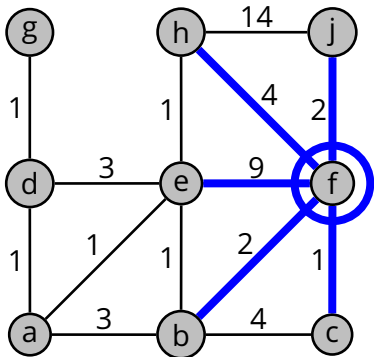


$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$

$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$

$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$

Link-State Advertisements



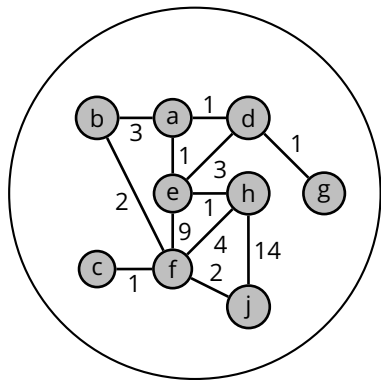
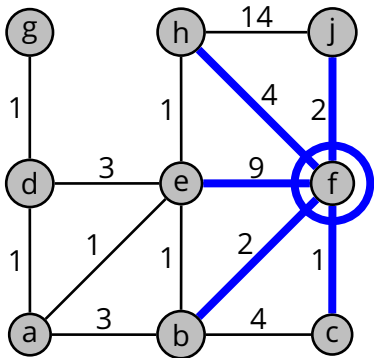
$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$

$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

Link-State Advertisements



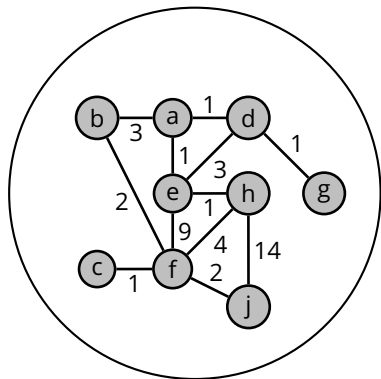
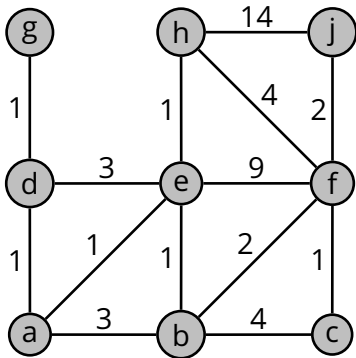
$$LSA_a = \{(a, b, 3), (a, e, 1), (a, d, 1)\}$$

$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

Link-State Advertisements



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

$$LSA_h = \{(h, e, 1), (h, f, 4), (h, j, 14)\}$$

$$LSA_d = \{(d, a, 1), (d, g, 1), (d, e, 3)\}$$

$$LSA_f = \{(f, c, 1), (f, b, 1), (f, e, 3), (f, h, 4), (f, j, 2)\}$$

...

Link-State 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***

■ *Flooding*

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

- ***Flooding***

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

- Simple and elegant

■ *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

■ *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?

■ ***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)

■ *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

■ *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

■ *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

■ *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?

■ *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

- *Sequence-number controlled flooding*

■ *Sequence-number controlled flooding*

- ▶ the originator s of a broadcast packet marks the packet with a sequence number n_s

■ *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

■ *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 : $seq(p) > n_s$

■ *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

■ *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

- *Scalability*

- ▶ hundreds of millions of hosts in today's Internet

■ *Scalability*

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

■ *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

■ *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*

■ *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

■ *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

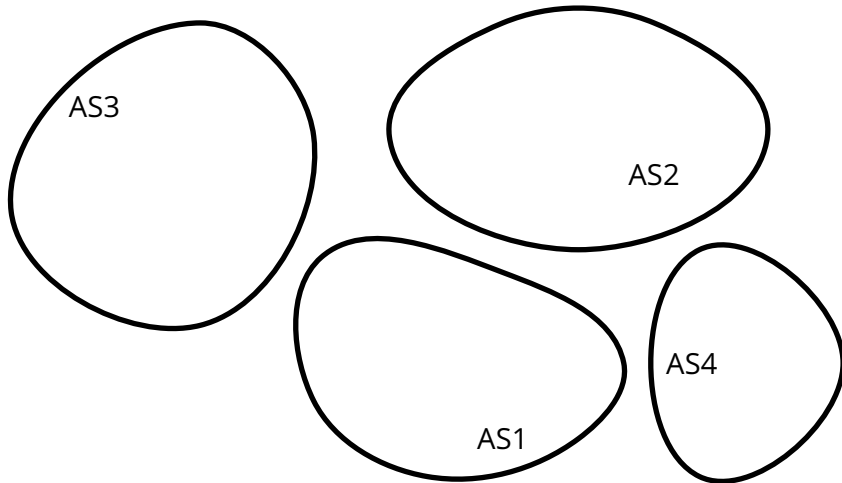
- Today's Internet is organized in *autonomous systems (ASs)*
 - ▶ independent administrative domains

Hierarchical Structure

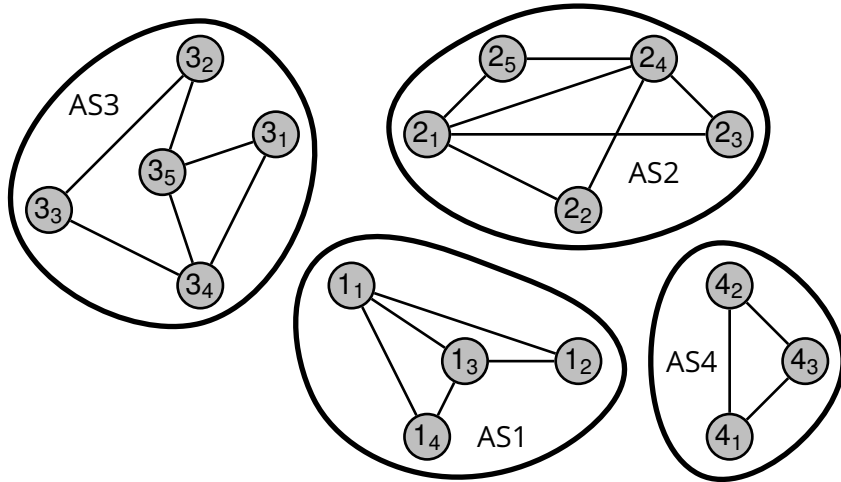
- Today's Internet is organized in ***autonomous systems (ASs)***
 - ▶ independent administrative domains
- ***Gateway routers*** connect an autonomous system with other autonomous systems

- 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

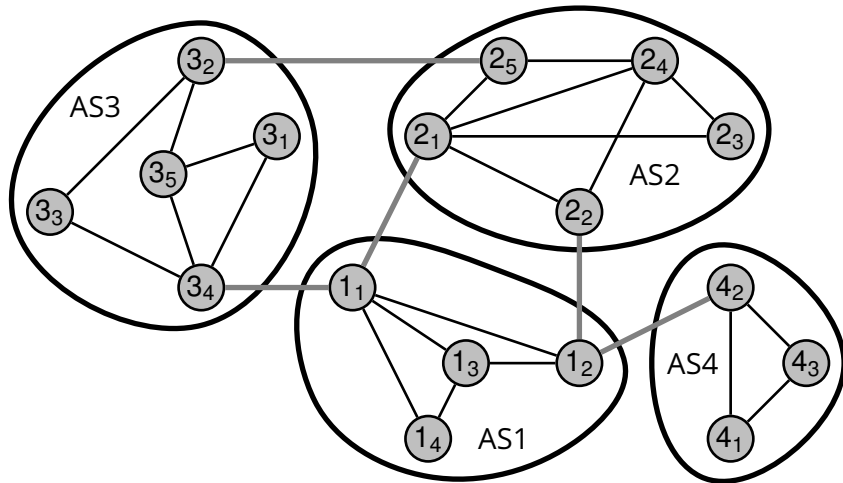
Hierarchical Structure



Hierarchical Structure

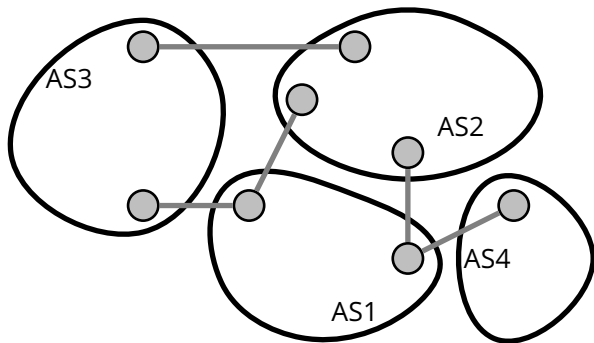


Hierarchical Structure



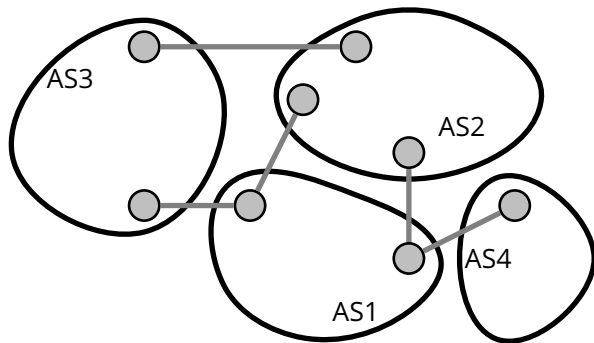
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level

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



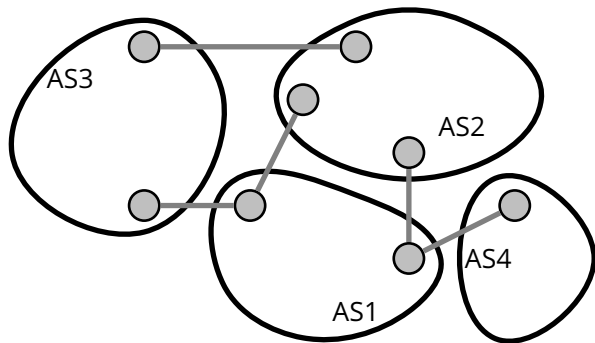
At AS3:
AS1 →

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



At AS3:
AS1 → AS1;

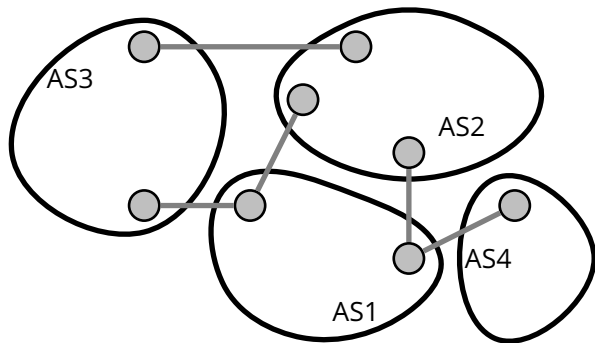
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:

AS1 → AS1; AS2 →

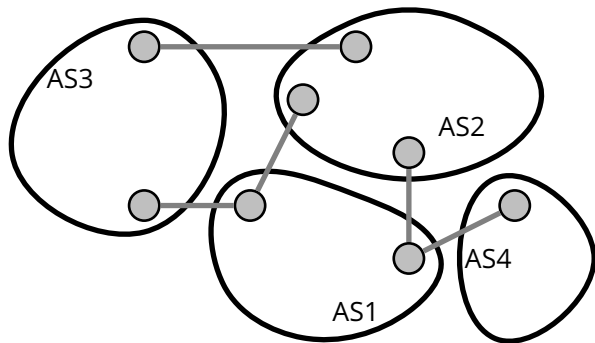
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:

AS1 \rightarrow AS1; AS2 \rightarrow AS2;

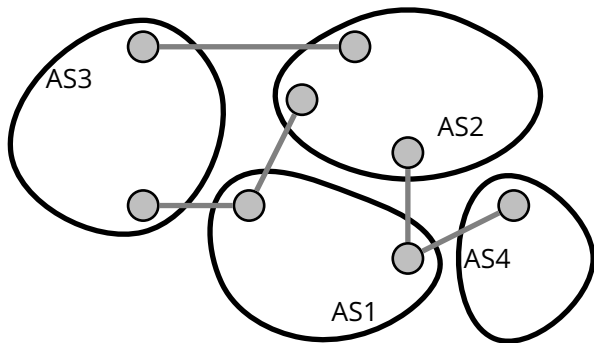
- An *inter-autonomous system routing protocol* determines routing at the autonomous-system level



At AS3:

AS1 → AS1; AS2 → AS2; AS4 →

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



At AS3:

AS1 \rightarrow AS1; AS2 \rightarrow AS2; AS4 \rightarrow AS1.

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

- All routers within an AS compute their *intra-AS* routing information
 - ▶ using an *intra-domain* routing protocol
- Gateway routers figure out *inter-AS* routing information
 - ▶ using an *inter-domain* routing protocol

- All routers within an AS compute their *intra-AS* routing information
 - ▶ using an *intra-domain* 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

- All routers within an AS compute their *intra-AS* routing information
 - ▶ using an *intra-domain* 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, \dots ?

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, \dots ?
 - ▶ use *intra-AS* routing information to determine the costs of the (least-cost) paths to G_x, G'_x, \dots
 - ▶ “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

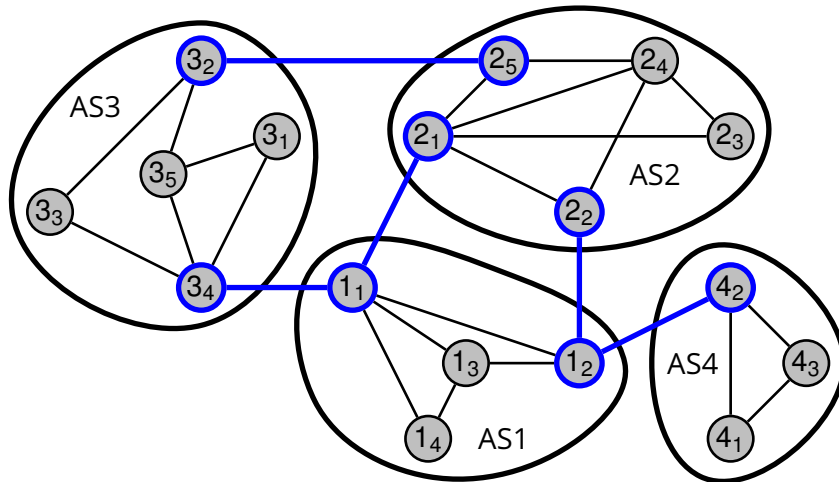
BGP Architecture and Terminology

- **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 *e*BGP



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.,

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

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.,

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

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

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.,

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

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

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 to the given destination AS

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 to the given destination AS
 - ▶ *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 to the given destination AS
 - ▶ *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*

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

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

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

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. ...