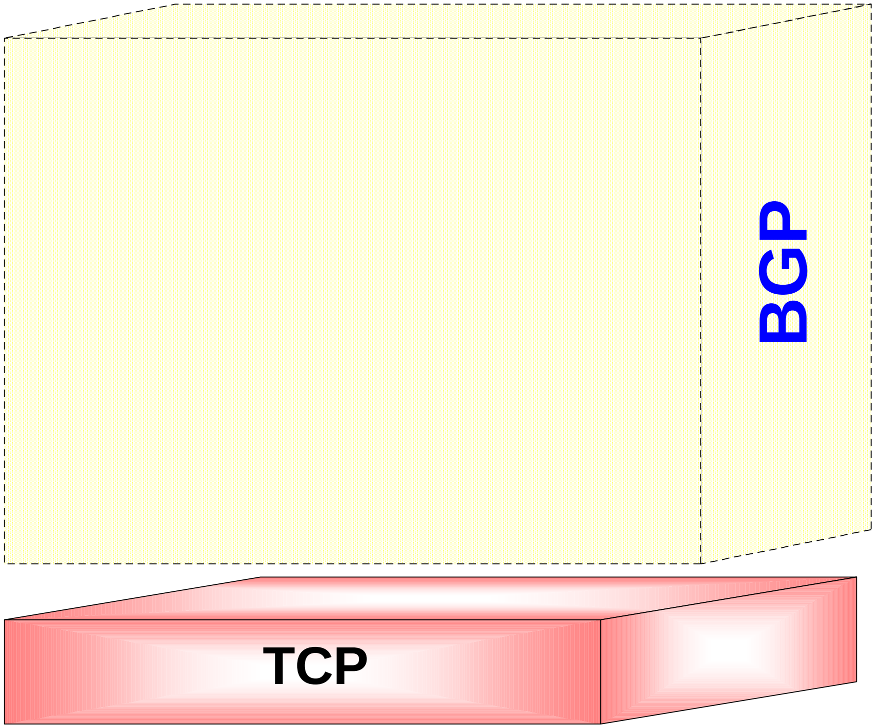
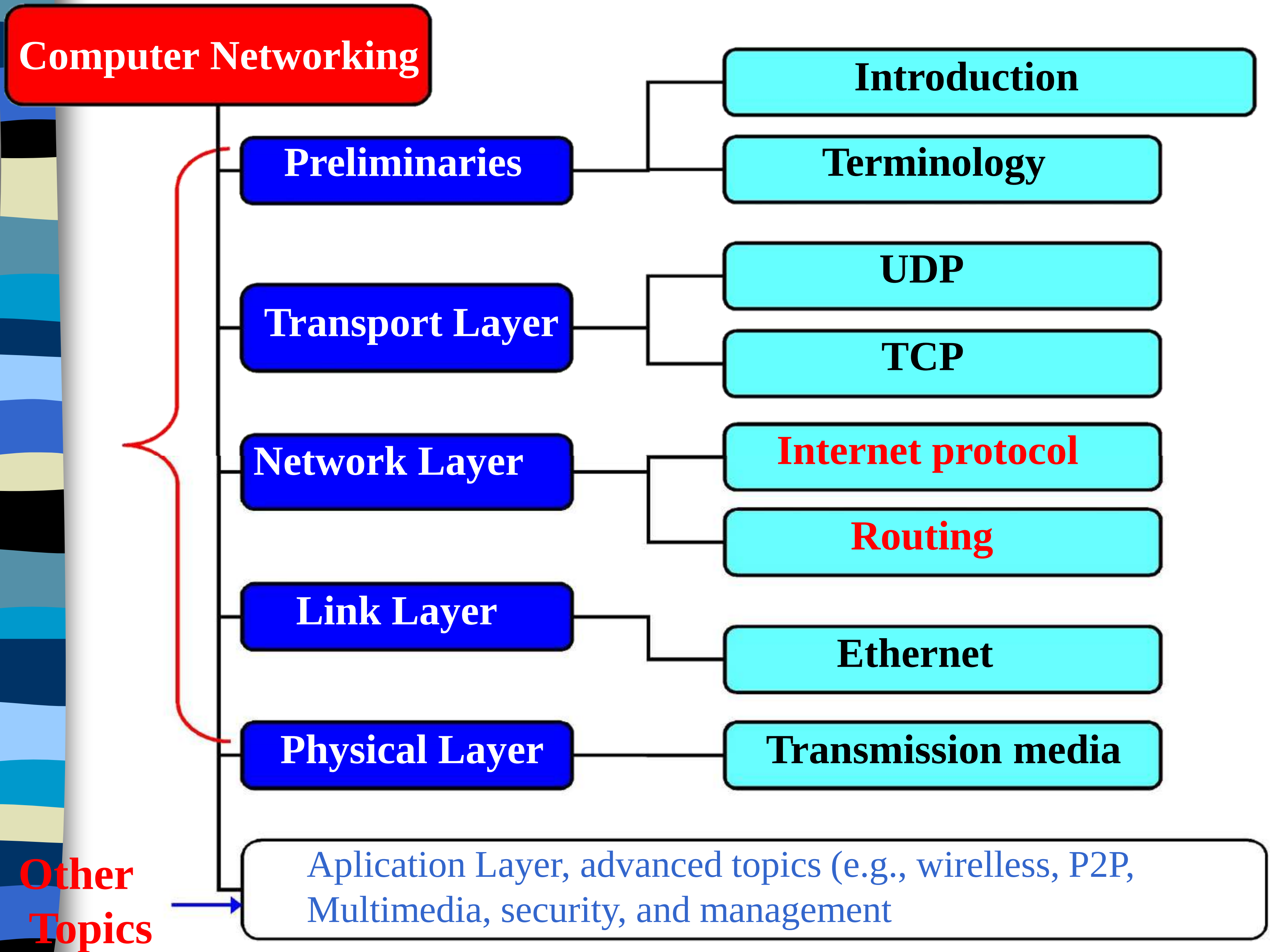
Protocoles et Interconnexions

Course Overview and Introduction

Dario Vieira
Department of Computer Science
EFREI

Routing Protocol





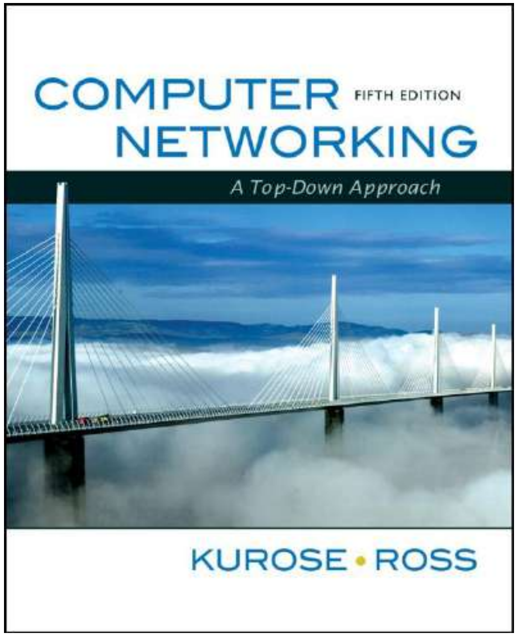
Computer Networking

- Preliminaries
  - Introduction
  - Terminology
- Transport Layer
  - UDP
  - TCP
- Network Layer
  - Internet protocol
  - Routing
- Link Layer
  - Ethernet
- Physical Layer
  - Transmission media

Other Topics

Aplication Layer, advanced topics (e.g., wirelless, P2P, Multimedia, security, and management

Chapter 3
Transport Layer

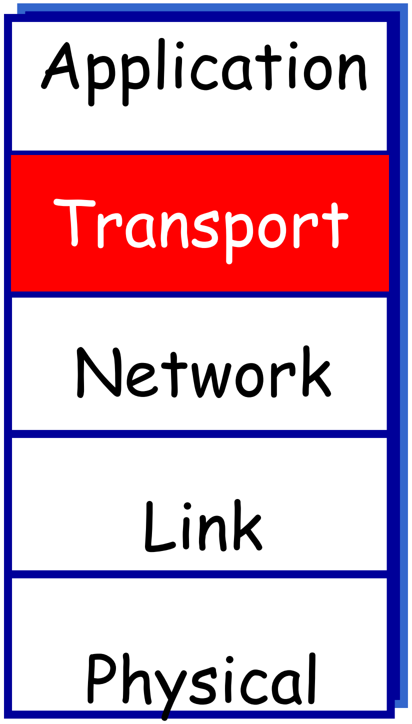


Computer Networking: A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Chapter 3: Transport Layer

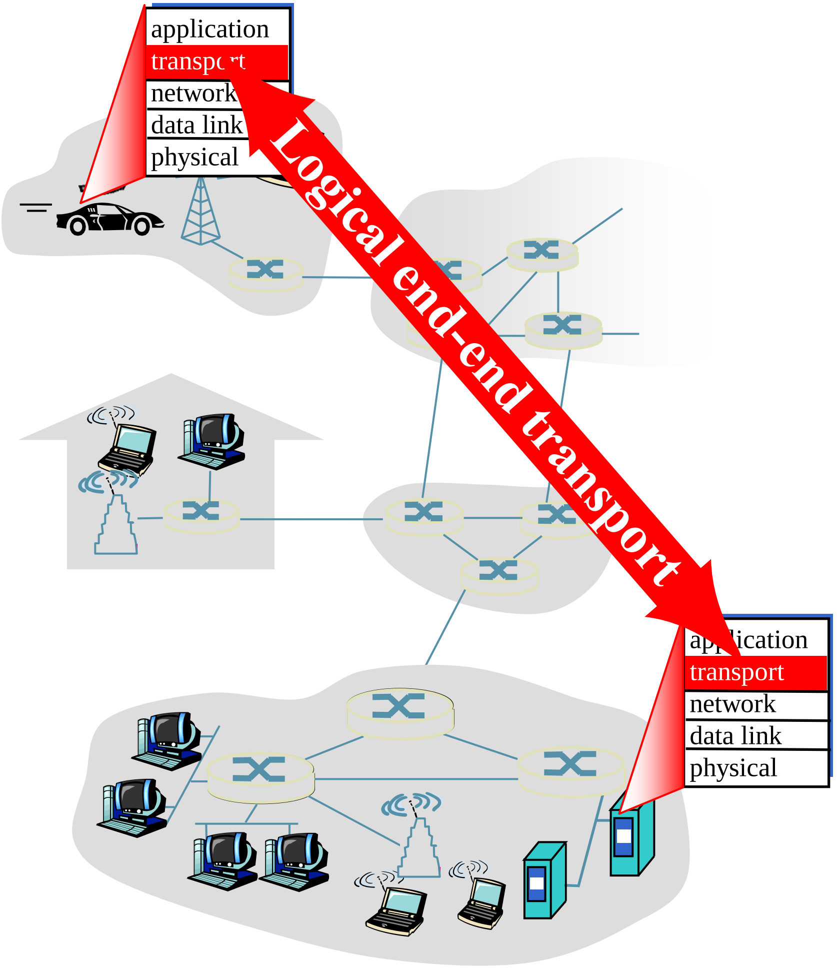
- Our Goals

  - ❖ Learn about transport layer protocols in the Internet
    - UDP: connectionless transport
    - TCP: connection-oriented transport



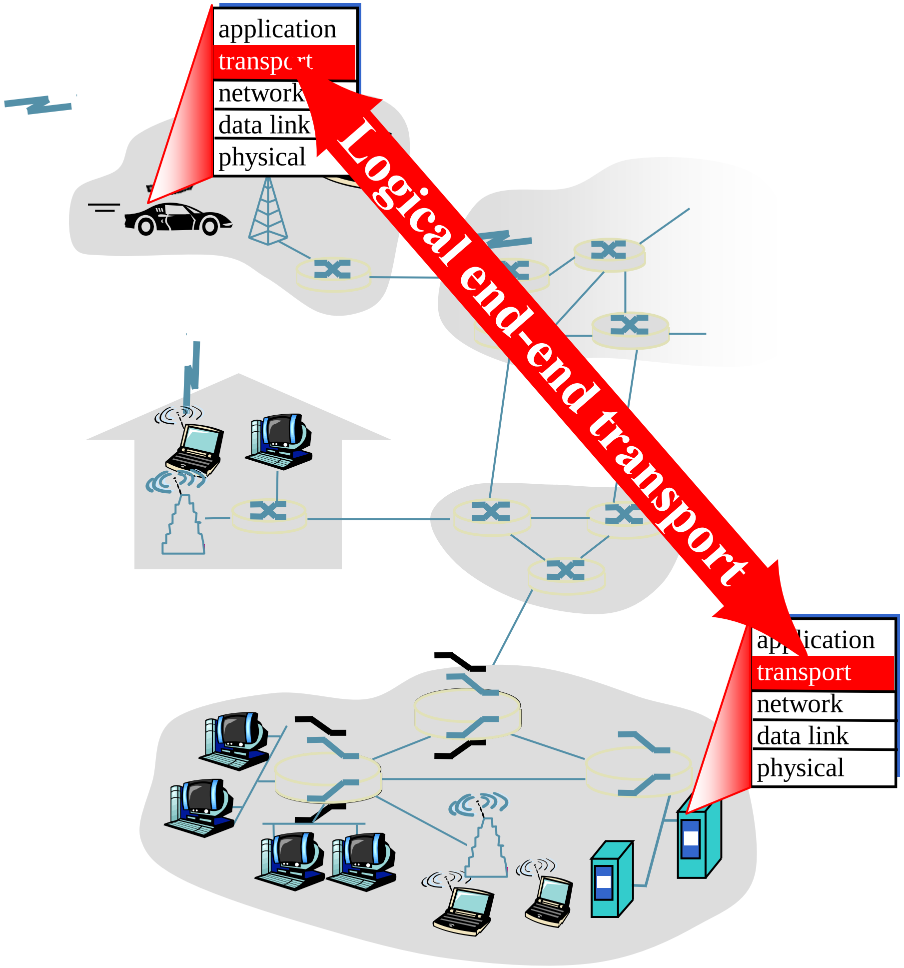
Transport services and protocols

- Transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps
  - Internet: TCP and UDP



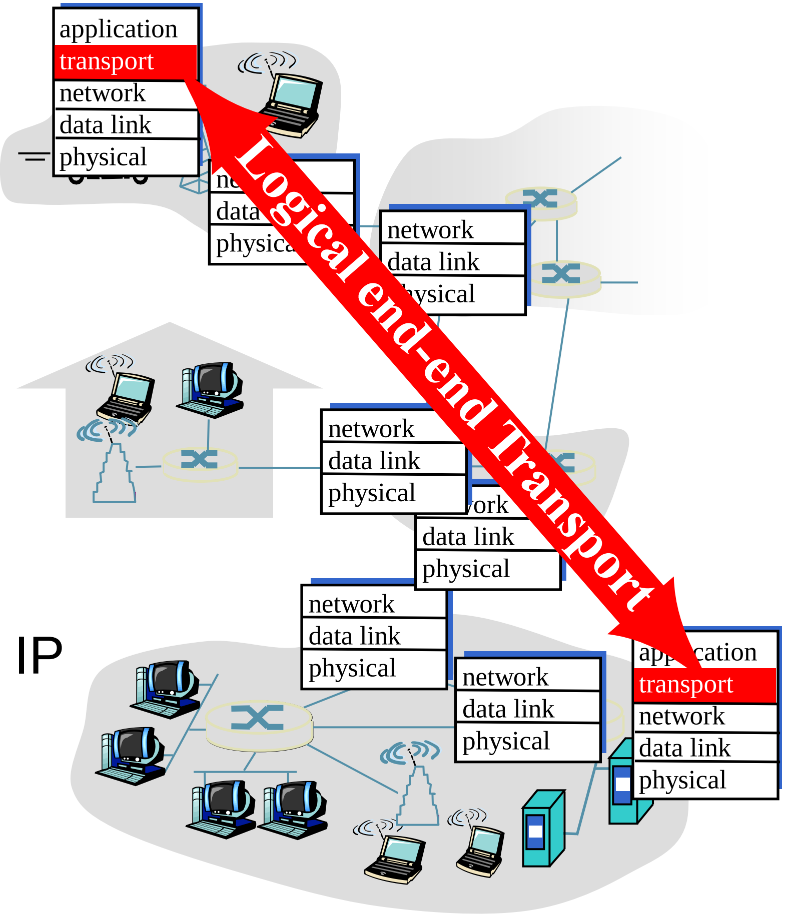
Transport vs. Network layer

- *Network layer*
  - Logical communication between **hosts**
- *Transport layer*
  - Logical communication between **processes**
  - Relies on, enhances, network layer services



Internet Transport-layer Protocols

- Reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- Unreliable, unordered delivery (UDP)
  - no-frills extension of “best-effort” IP
- Services not available
  - delay guarantees
  - bandwidth guarantees



Chapter 3 outline

UDP: User Datagram Protocol [RFC 768]

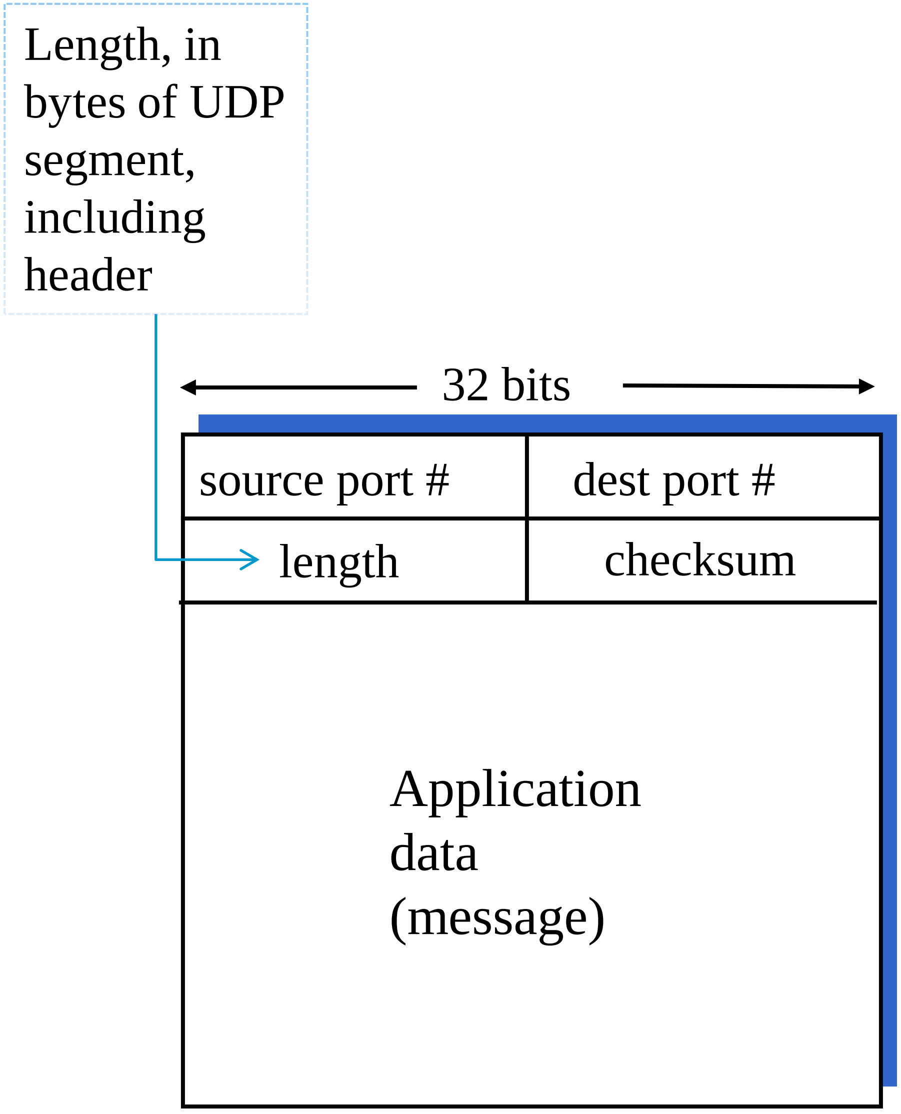
- “No frills,” “bare bones” Internet transport protocol
- “Best effort” service, UDP segments may be:
  - lost
  - delivered out of order to app
- **Connectionless**
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

Why is there a UDP?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small segment header
- No congestion control: UDP can blast away as fast as desired

UDP: more

- Often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- Other UDP uses
  - DNS
  - SNMP
- Reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

Chapter 3 outline

3.1 Transport-layer services

3.2 Connectionless Transport: UDP

3.3 Principles of reliable data transfer

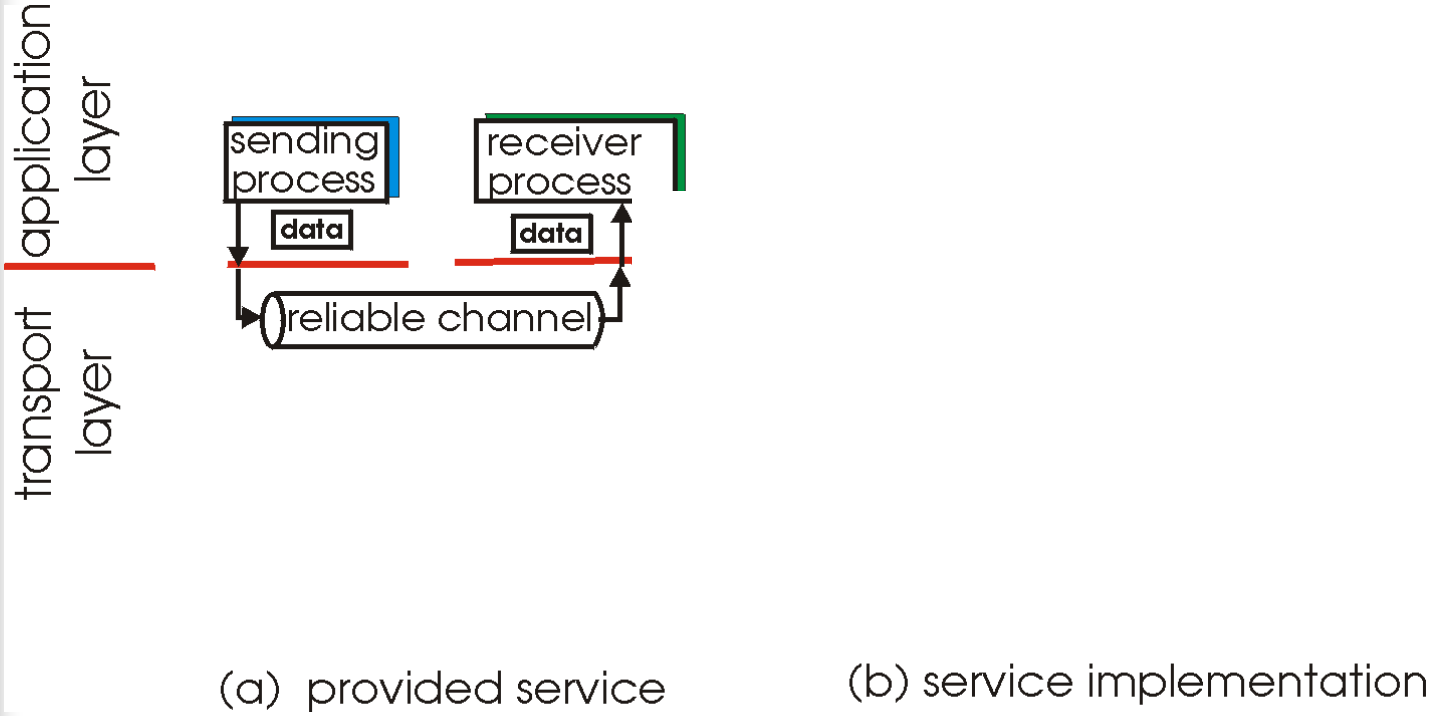
3.4 Connection-oriented transport: TCP

3.5 Principles of congestion control

3.6 TCP congestion control

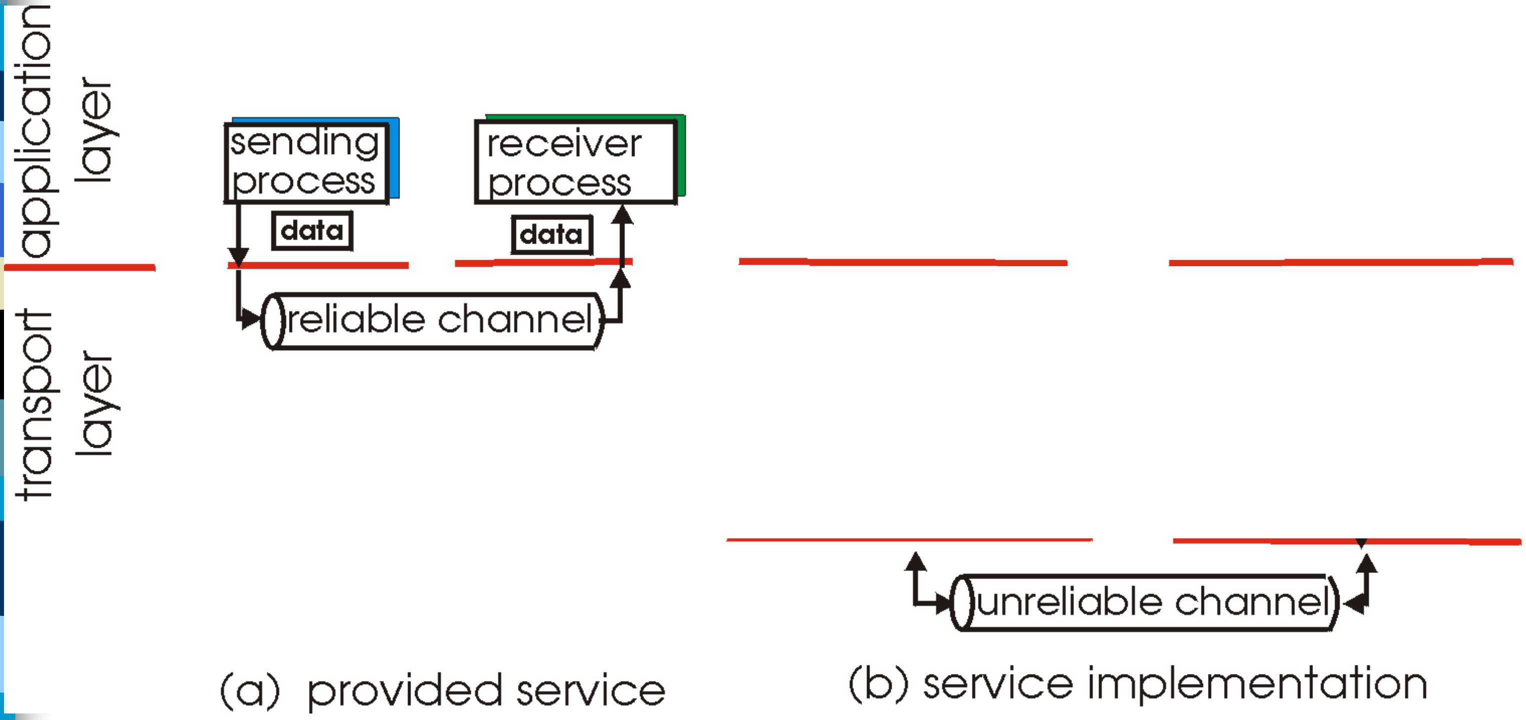
Principles of Reliable Data Transfer

- Important in app., transport, link layers
  - top-10 list of important networking topics!



Principles of Reliable Data Transfer

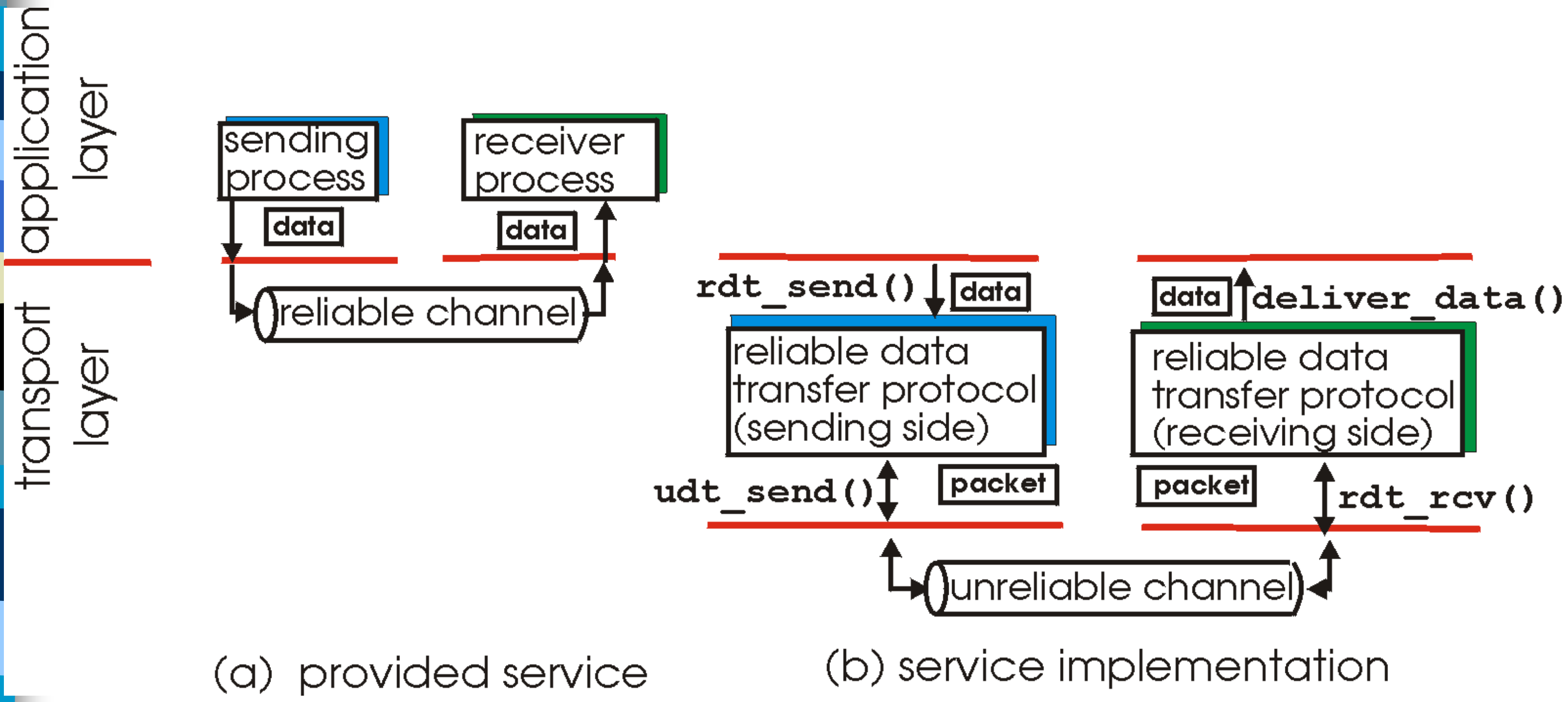
- Important in app., transport, link layers
  - top-10 list of important networking topics!



- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol

Principles of Reliable Data Transfer

- Important in app., transport, link layers
  - top-10 list of important networking topics!

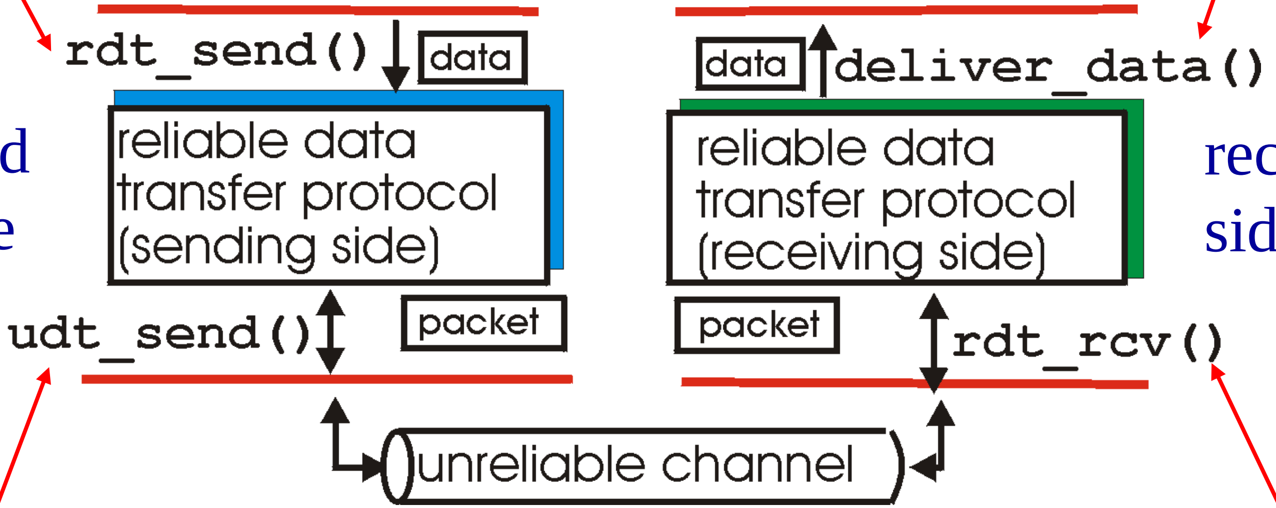


- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable Data Transfer: Getting Started

`rdt_send()`: called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

`deliver_data()`: called by **`rdt`** to deliver data to upper



send side

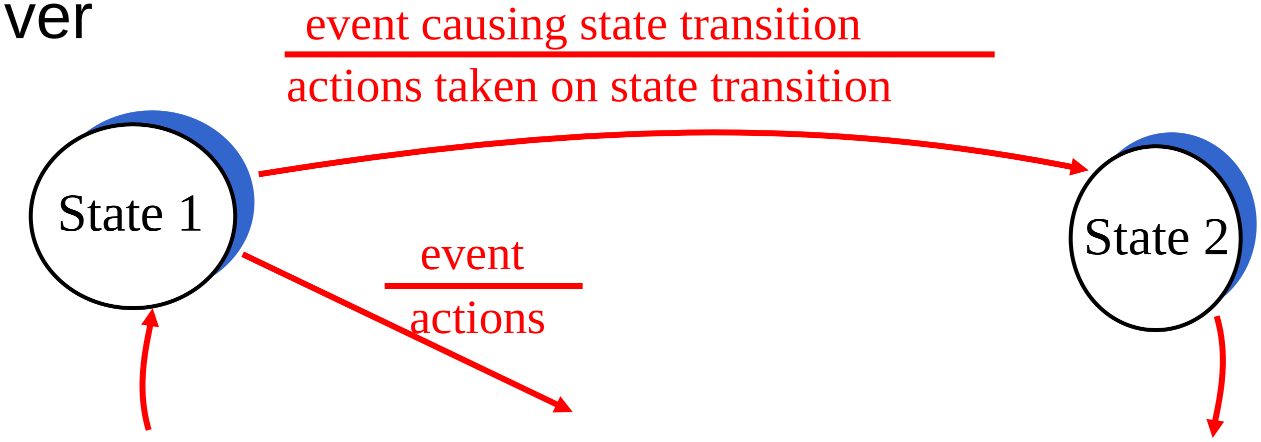
receive side

`udt_send()`: called by rdt, to transfer packet over unreliable channel to receiver

`rdt_rcv()`: called when packet arrives on rcv-side of channel

Reliable Data Transfer: Getting Started

- Incrementally develop sender, receiver sides of Reliable Data Transfer protocol (rdt)
- Consider only unidirectional data transfer
  - but control info will flow on both directions!
- Use Finite State Machines (FSM) to specify sender, receiver



event causing state transition

actions taken on state transition

State 1

event

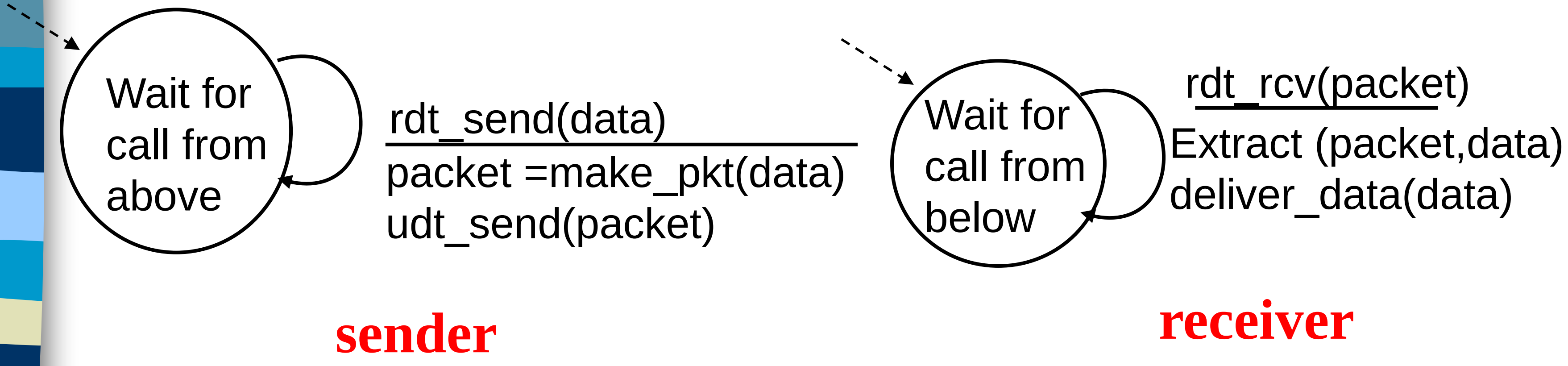
actions

State 2

state: when in this "state" next state uniquely determined by next event

Rdt1.0: Reliable Transfer over a Reliable Channel

- Underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- Separate FSMs for sender, receiver
  - sender sends data into underlying channel
  - receiver read data from underlying channel



Rdt2.0: channel with bit errors

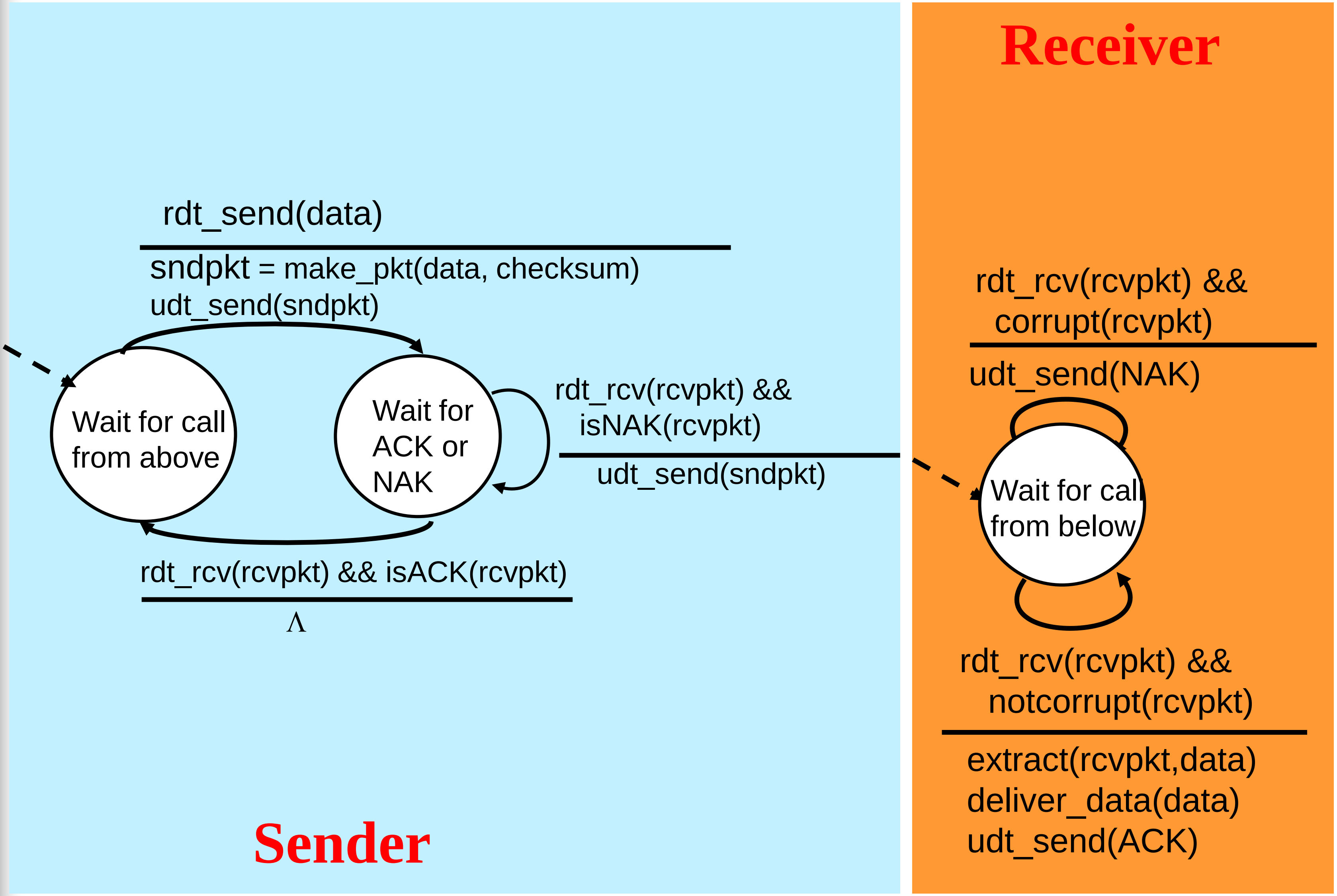
- Underlying channel may flip bits in packet
  - checksum to detect bit errors
- The question: how to recover from errors?

How do humans recover from "errors" during conversation?

Rdt2.0: channel with bit errors

- Underlying channel may flip bits in packet
  - checksum to detect bit errors
- The question: how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender retransmits pkt on receipt of NAK
- New mechanisms in **`rdt2.0`** (beyond **`rdt1.0`**):
  - error detection
  - receiver feedback: control msgs (ACK,NAK) rcvr->sender

rdt2.0: FSM specification



rdt_send(data)

sndpkt = make_pkt(data, checksum)
udt_send(sndpkt)

Wait for call from above

Wait for ACK or NAK

rdt_rcv(rcvpkt) && isNAK(rcvpkt)

udt_send(sndpkt)

rdt_rcv(rcvpkt) && isACK(rcvpkt)

Λ

Sender

Receiver

rdt_rcv(rcvpkt) && corrupt(rcvpkt)

udt_send(NAK)

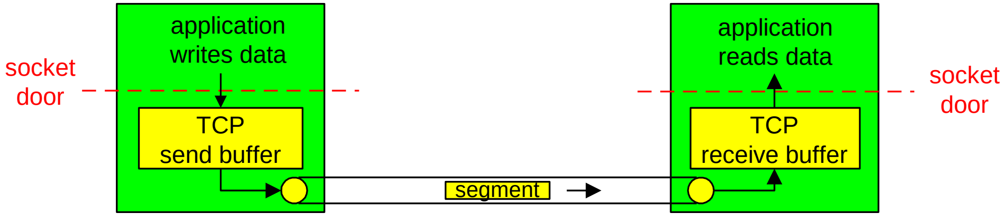
Wait for call from below

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)

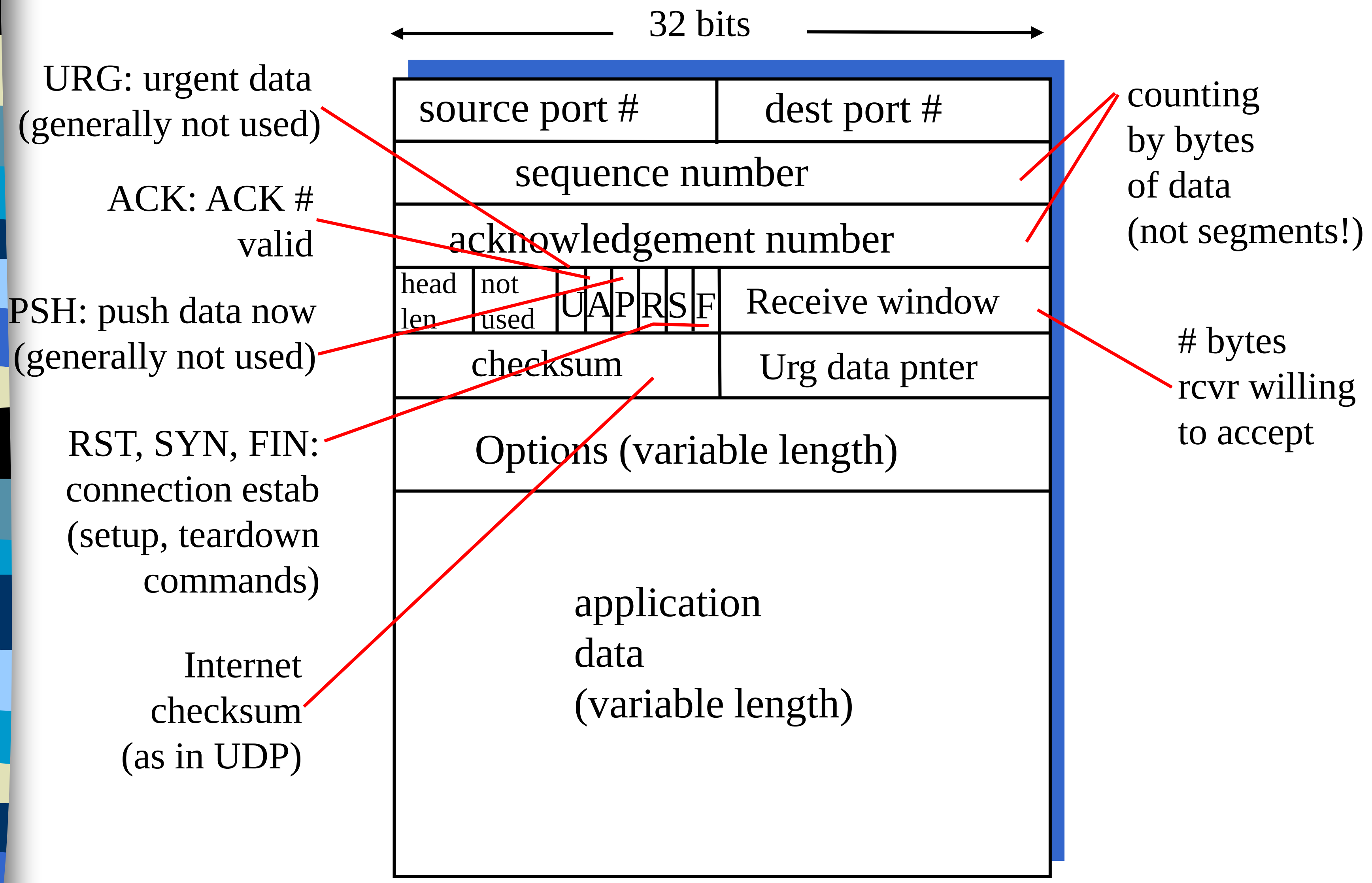
extract(rcvpkt,data)
deliver_data(data)
udt_send(ACK)

TCP: Overview

- Point-to-point:
  - one sender, one receiver
- Reliable, in-order *byte steam:*
  - no "message boundaries"
- Pipelined:
  - TCP congestion and flow control set window size
- *Send & receive buffers*
- Full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- Connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- Flow controlled:
  - sender will not overwhelm receiver



TCP segment structure



32 bits

source port #

dest port #

sequence number

acknowledgement number

head len

not used

U A P R S F

Receive window

checksum

Urg data pnter

Options (variable length)

application
data
(variable length)

URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)

counting by bytes of data (not segments!)

bytes rcvr willing to accept

TCP seq. #'s and ACKs

Seq. #'s:

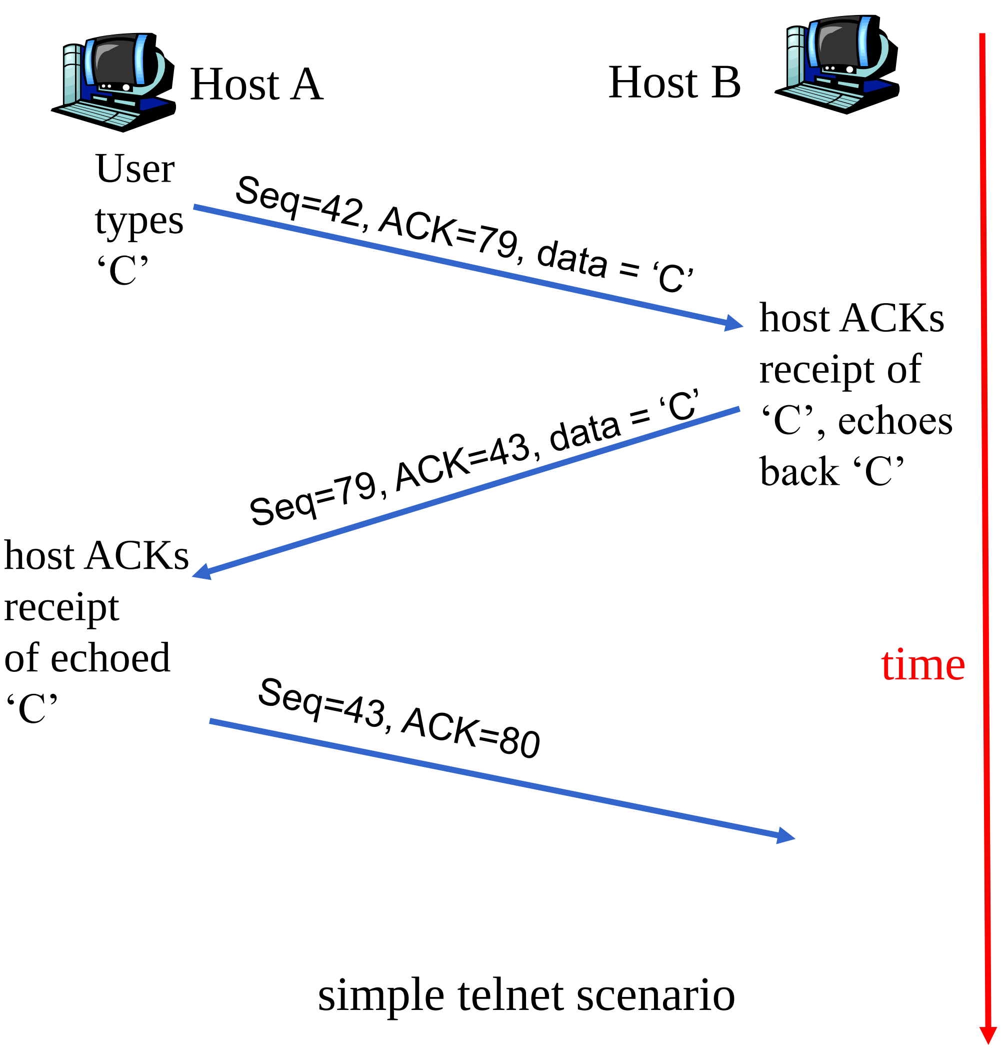
- byte stream "number" of first byte in segment's data

ACKs:

- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor



Host A

Host B

User types 'C'

Seq=42, ACK=79, data = 'C'

host ACKs receipt of 'C', echoes back 'C'

Seq=79, ACK=43, data = 'C'

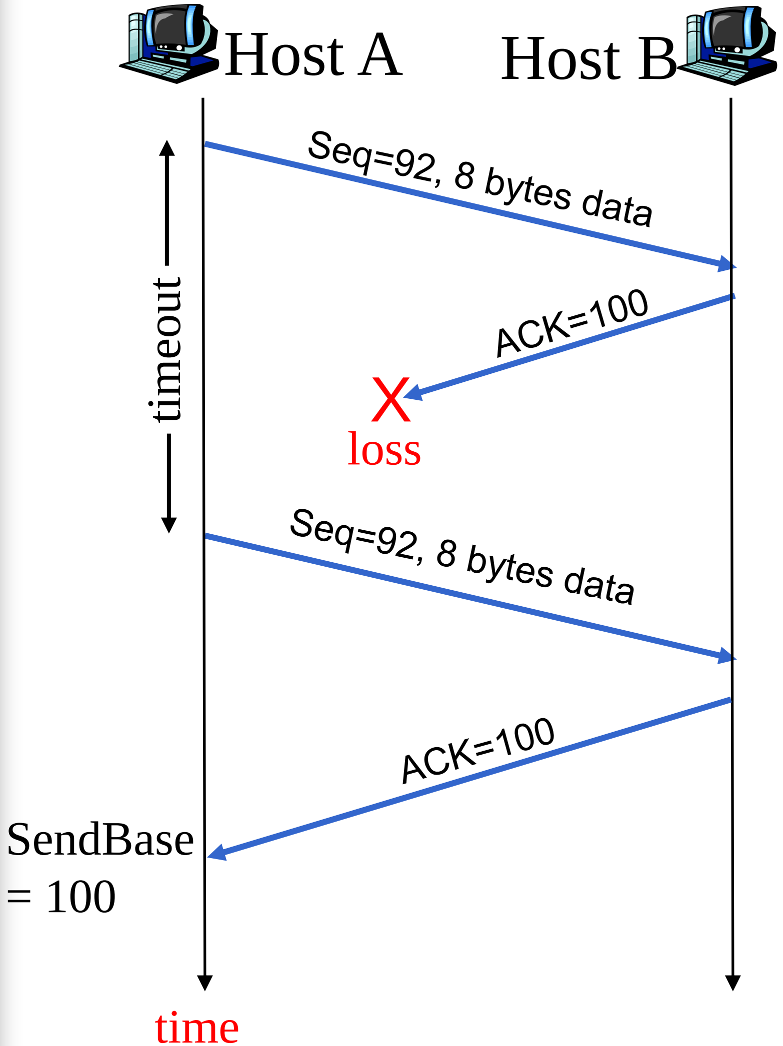
host ACKs receipt of echoed 'C'

Seq=43, ACK=80

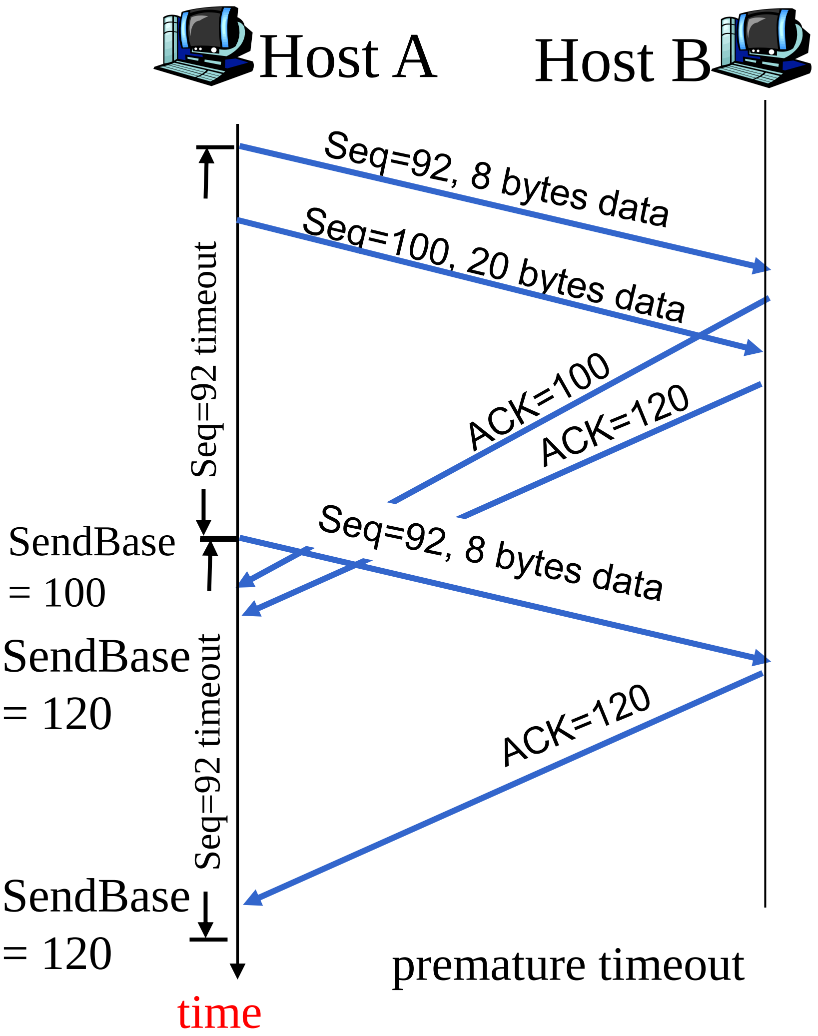
time

simple telnet scenario

TCP: Retransmission Scenarios

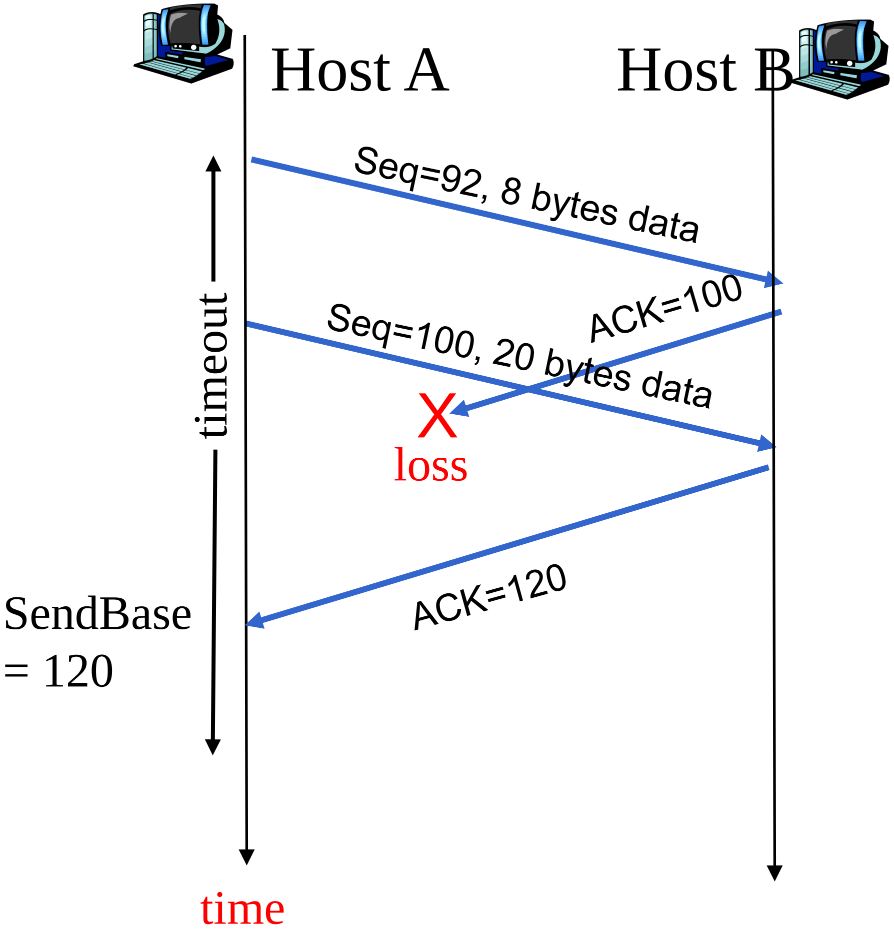


lost ACK scenario



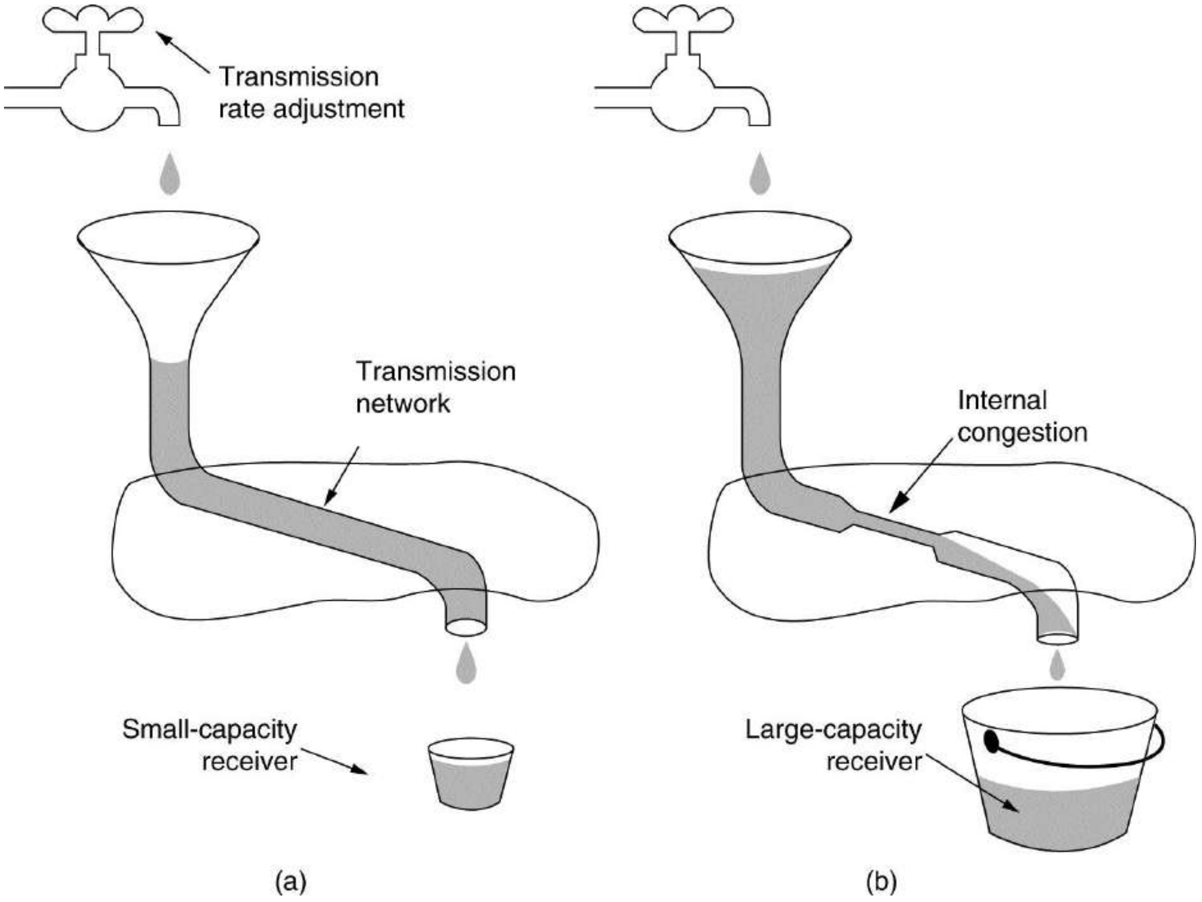
premature timeout

TCP Retransmission Scenarios (more)



Cumulative ACK scenario

TCP Congestion Control



(a) A fast network feeding a low capacity receiver.
(b) A slow network feeding a high-capacity receiver.

TCP Connection Management

Recall: TCP sender, receiver establish “connection” before exchanging data segments

- initialize TCP variables:
  - Initial seq. #s
  - Buffers, flow control info (e.g. **`RcvWindow`**)
- *client:* connection initiator

```
Socket clientSocket = new
 Socket("hostname","port
 number");
```

- *server:* contacted by client

```
Socket connectionSocket =
 welcomeSocket.accept();
```

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- specifies initial seq #
- no data

Step 2: server host receives SYN, replies with SYNACK segment
- server allocates buffers
- specifies server initial seq. #

Step 3: client receives SYNACK, replies with ACK segment, which may contain data

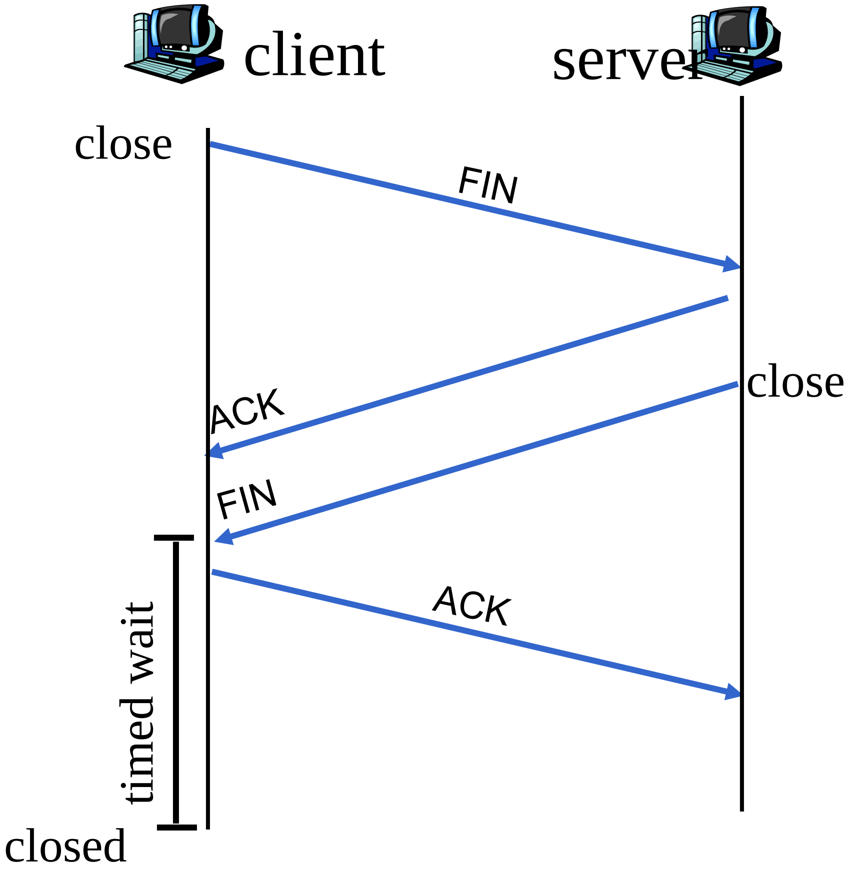
TCP Connection Management (cont.)

Closing a connection:

client closes socket:
`clientSocket.close();`

Step 1: client end system sends TCP FIN control segment to server

Step 2: server receives FIN, replies with ACK.



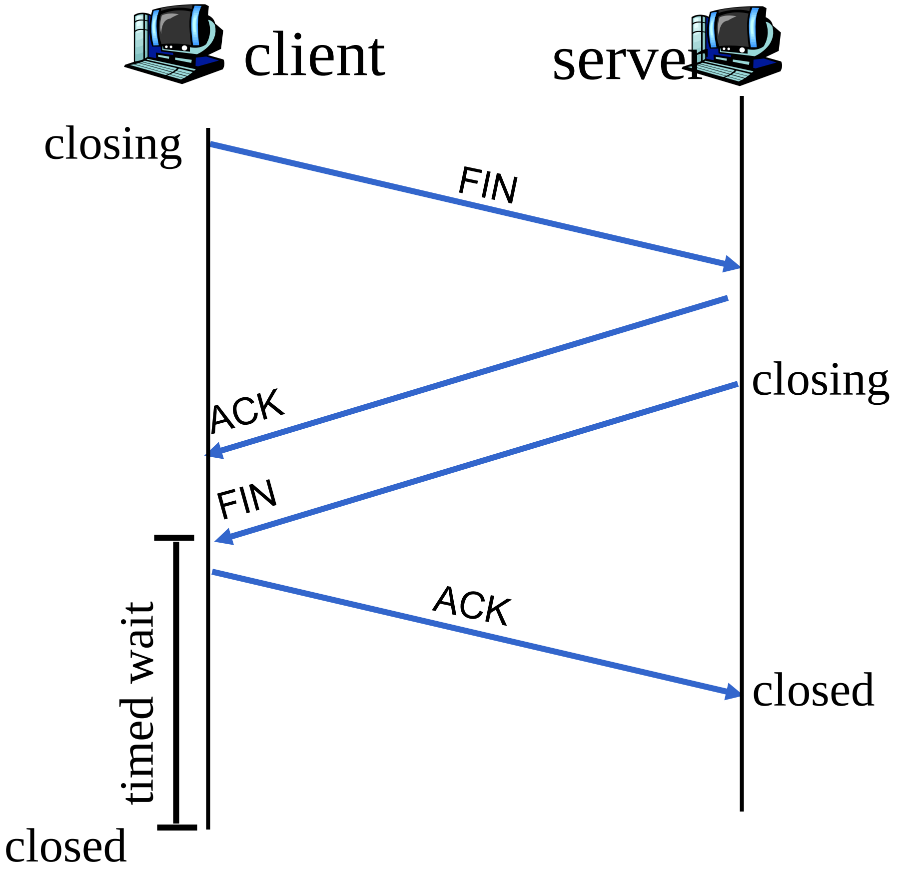
TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

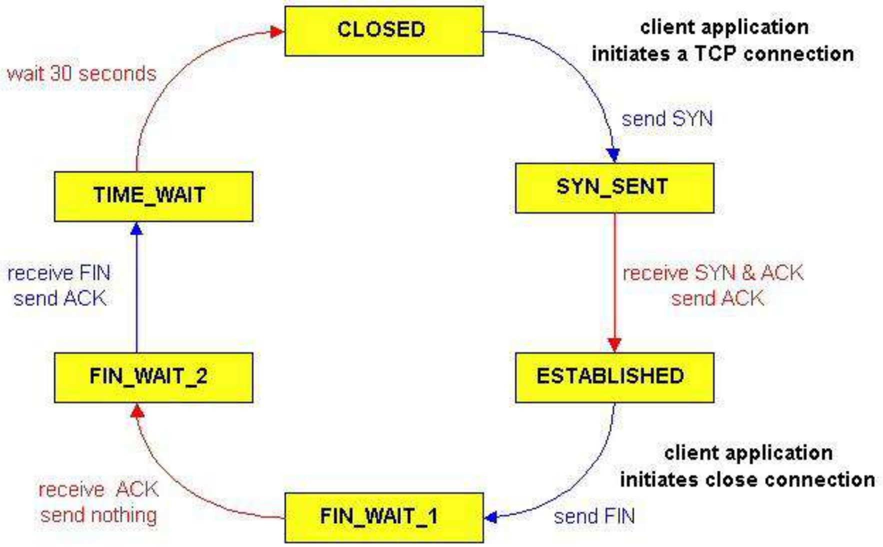
- Enters “timed wait” - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.

Step 5: after timeout, client ’s connection closed

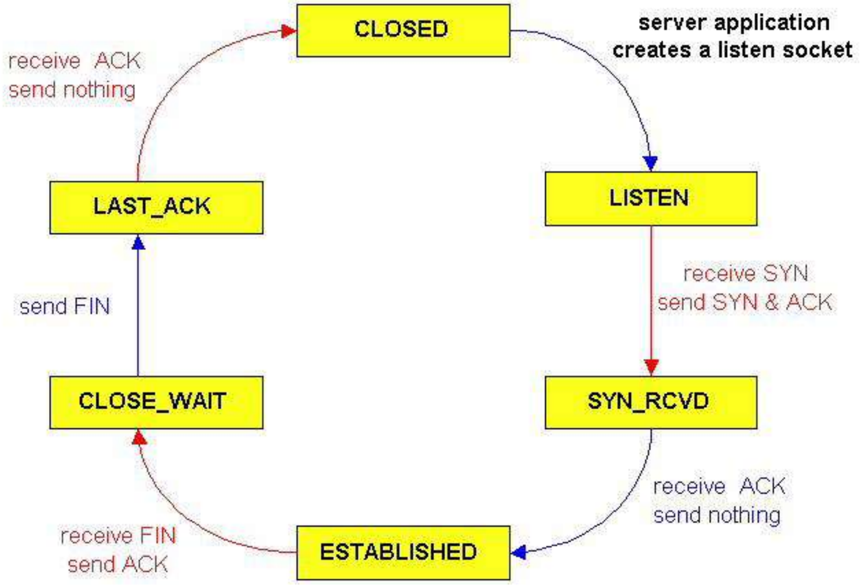


TCP Connection Management (cont)



TCP client
lifecycle

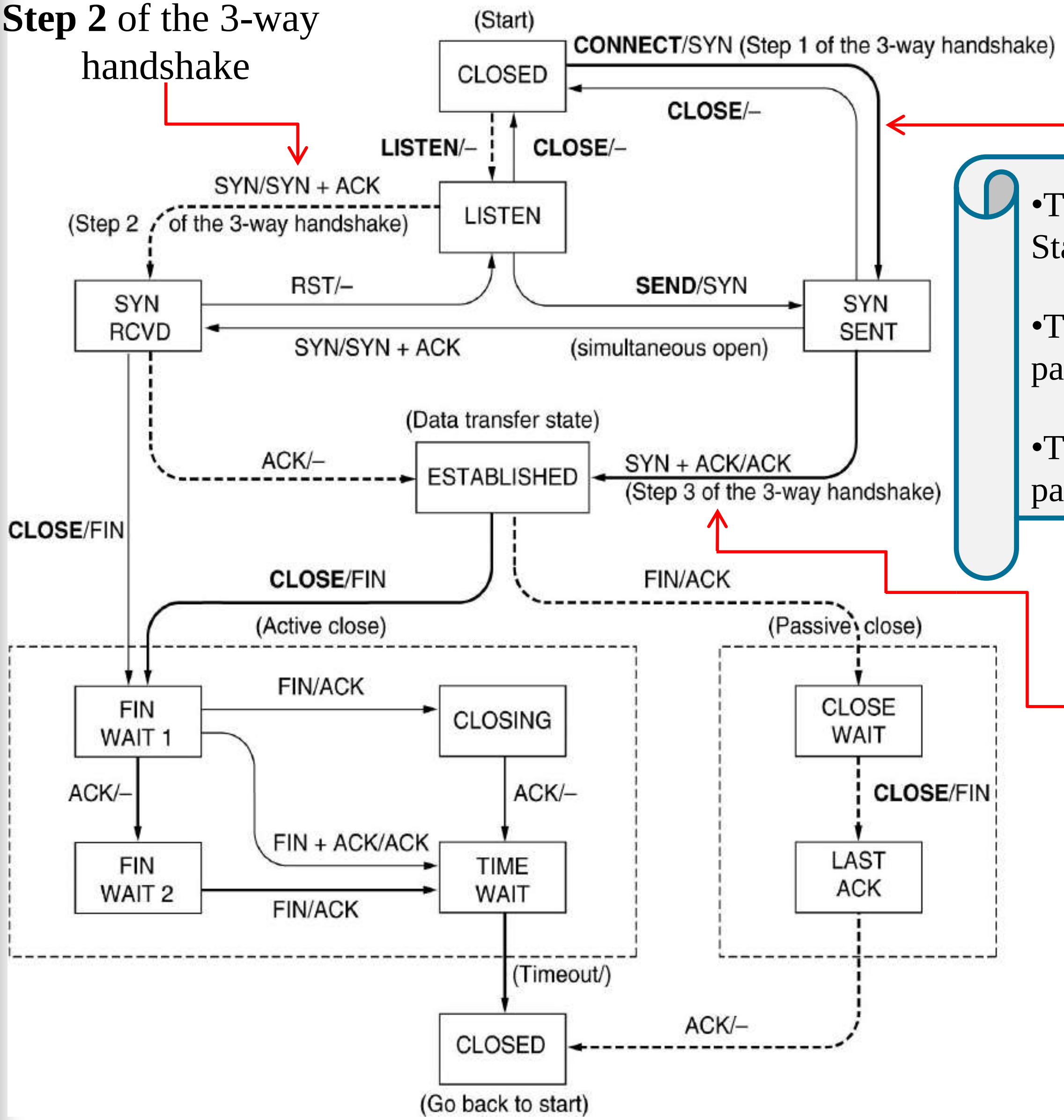
TCP server
lifecycle



TCP Finite State Machine

Step 1 of the 3-way handshake

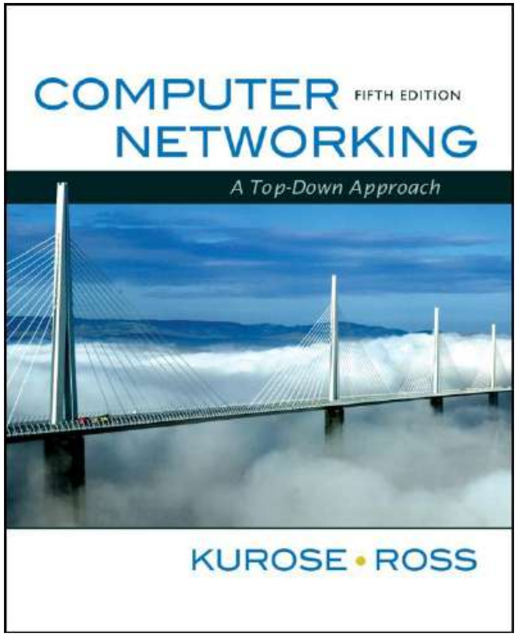
Step 2 of the 3-way handshake



- TCP connection management Finite State Machine: 3-Way Handshake
- The heavy solid line is the normal path for a client.
- The heavy dashed line is the normal path for a server.

Step 3 of the 3-way handshake

Chapter 4
Network Layer



Computer Networking: A Top Down Approach
5^{th} edition.

Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

Network Layer

Goals

- Understand principles behind network layer services
  - Network layer service models
  - Forwarding versus routing
  - How a router works
  - Routing (path selection)
  - Broadcast, multicast
- Instantiation, implementation in the Internet

Chapter 4: Network Layer

4. 1 Introduction

4.2 What's inside a router

4.4 IP: Internet Protocol

4.5 Routing algorithms

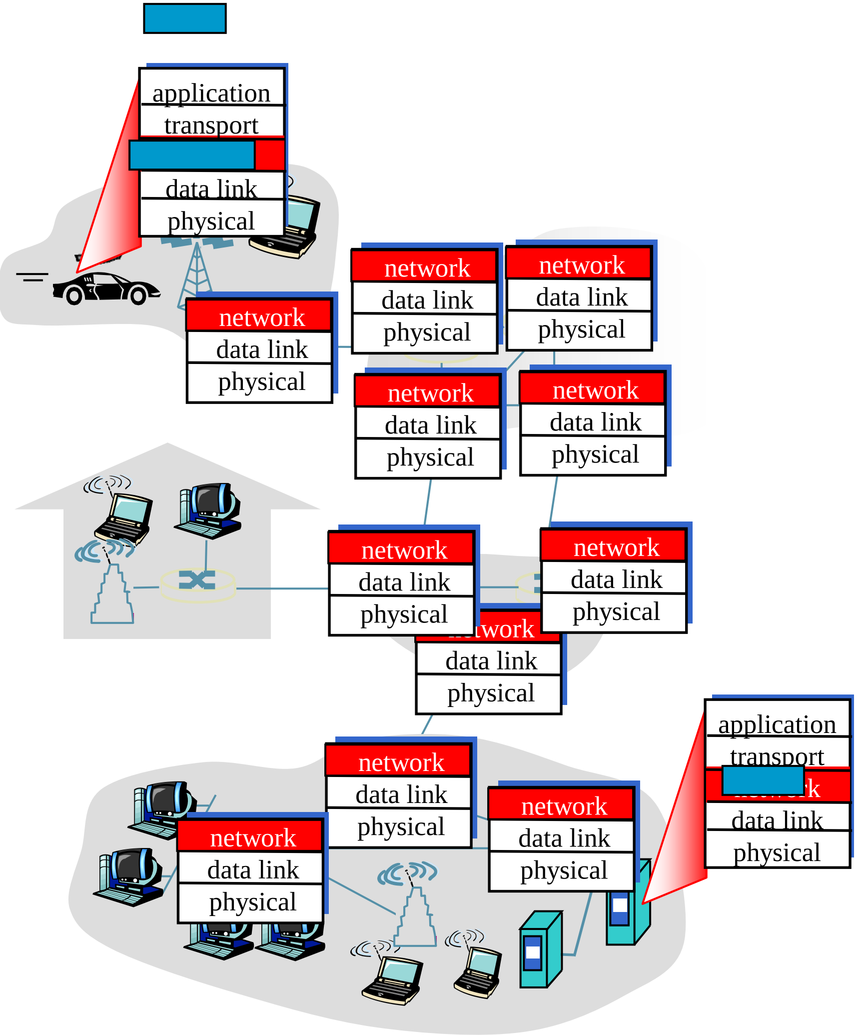
- Link state
- Distance Vector

4.6 Routing in the Internet

- RIP
- OSPF
- BGP

Network layer

- Transport segment from sending to receiving host
- On sending side encapsulates segments into datagrams
- On rcving side, delivers segments to transport layer
- Network layer protocols in *every* host, router
- Router examines header fields in all IP datagrams passing through it



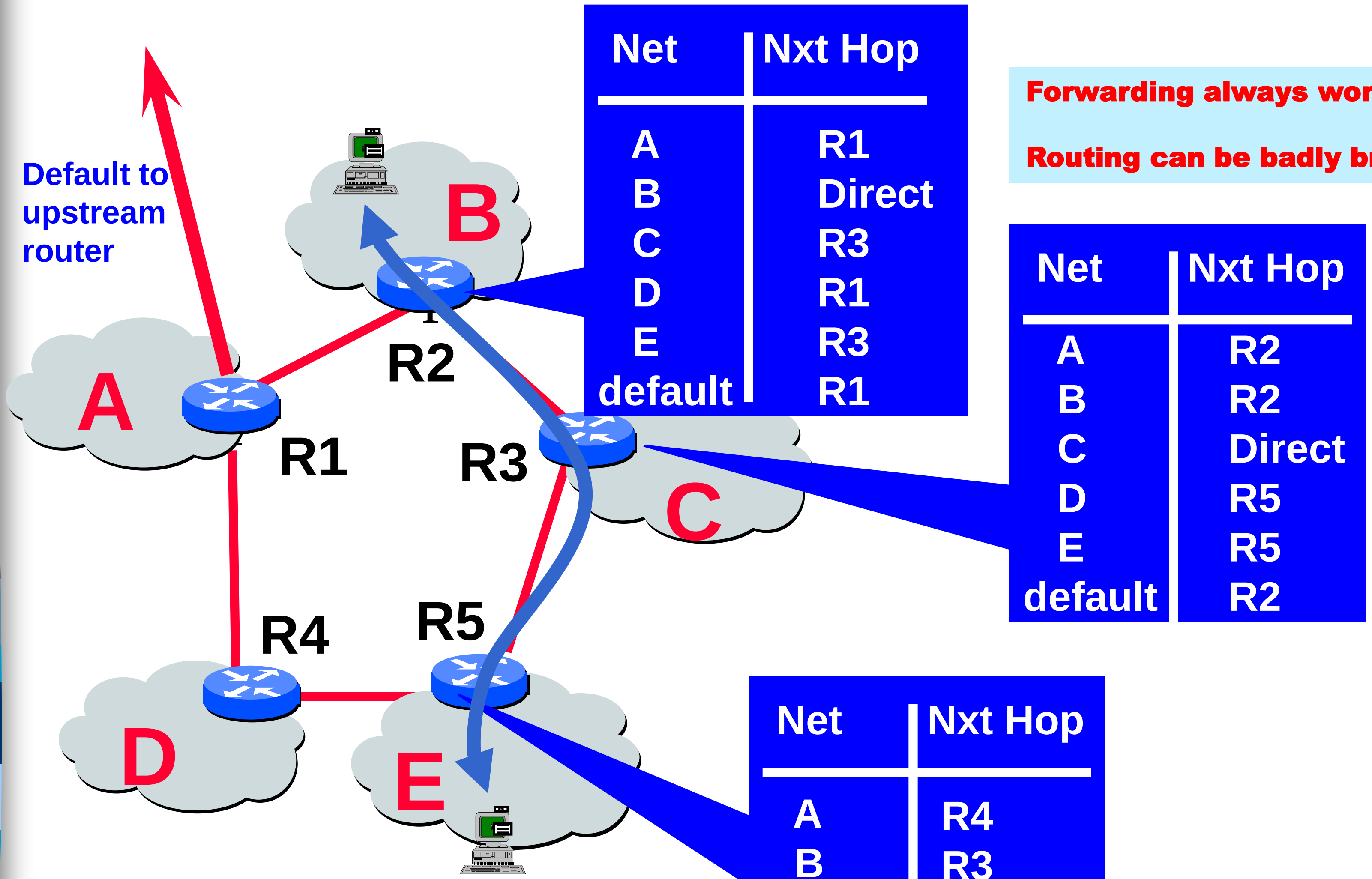
Two Key Network-Layer Functions

- ***Forwarding***
  - move packets from router's input to appropriate router output
- ***Routing***
  - determine route taken by packets from source to dest.
  - *routing algorithms*

Analogy:

- Routing
  - process of planning trip from source to destination
- Forwarding
  - process of getting through single interchange

Routing vs. Forwarding



Net	Nxt Hop
A	R1
B	Direct
C	R3
D	R1
E	R3
default	R1

Net	Nxt Hop
A	R2
B	R2
C	Direct
D	R5
E	R5
default	R2

Net	Nxt Hop
A	R4
B	R3
C	R3
D	R4
E	Direct
default	R4

Forwarding always works

Routing can be badly broken

Forwarding: determine next hop

Routing: establish end-to-end paths

Thanks to T. Griffin

How Are Forwarding Tables Populated to I Implement Routing?

Statically

Administrator manually configures forwarding table entries

+ More control
+ Not restricted to destination-based forwarding
- Doesn't scale
- Slow to adapt to network failures

Dynamically

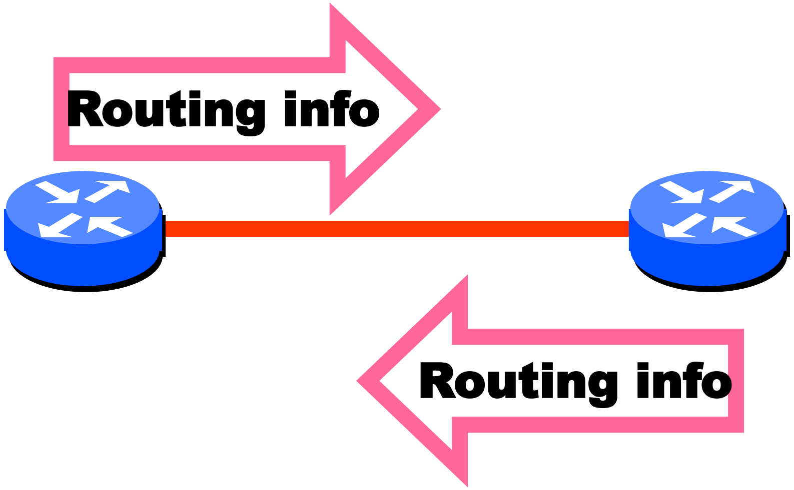
Routers exchange network reachability information using ROUTING PROTOCOLS. Routers use this to compute best routes

+ Can rapidly adapt to changes in network topology
+ Can be made to scale well
- Complex distributed algorithms
- Consume CPU, Bandwidth, Memory
- Debugging can be difficult
- Current protocols are destination-based

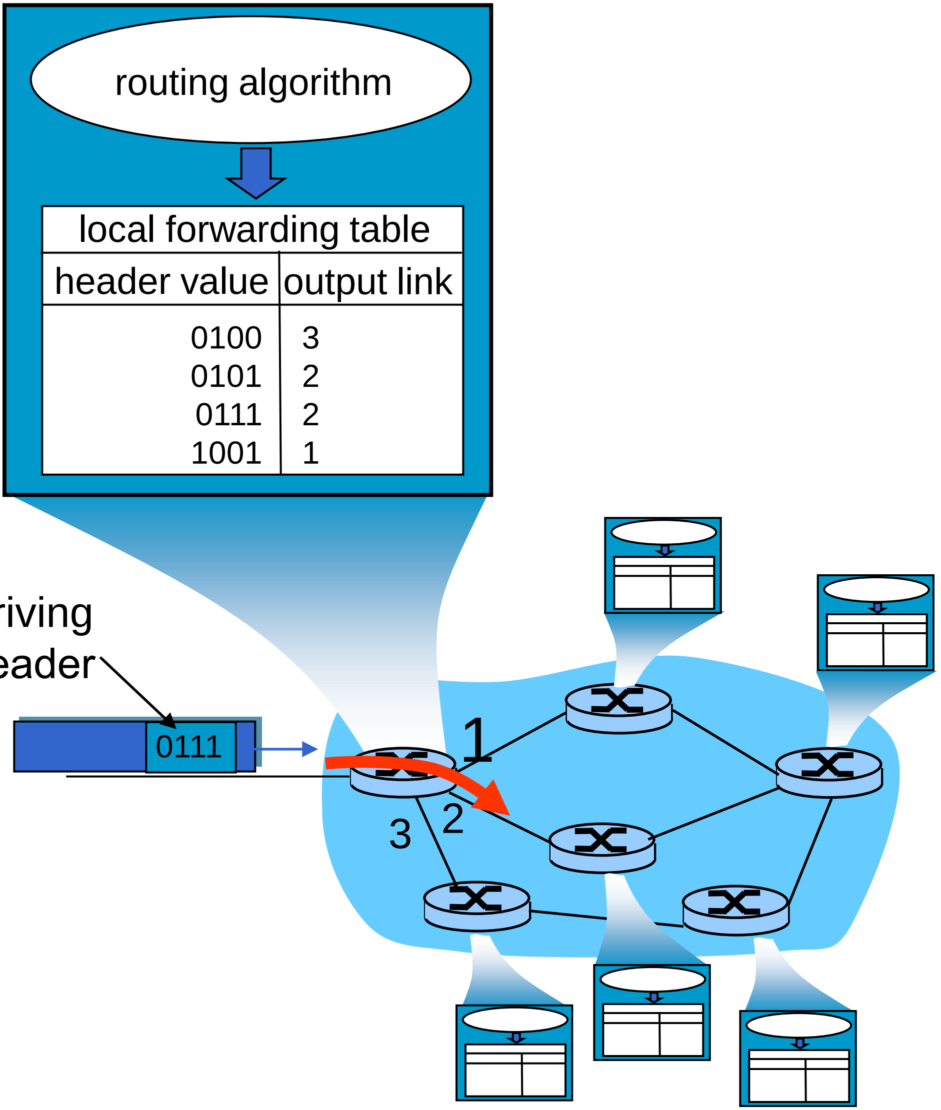
In practice : a mix of these.
Static routing mostly at the "edge"

Thanks to T. Griffin

Network reachability



Interplay between routing and forwarding



routing algorithm

local forwarding table

header value	output link
0100	3
0101	2
0111	2
1001	1

value in arriving packet's header

Chapter 4: Network Layer

4. 1 Introduction

4.2 What's inside a router

4.4 IP: Internet Protocol

4.5 Routing algorithms

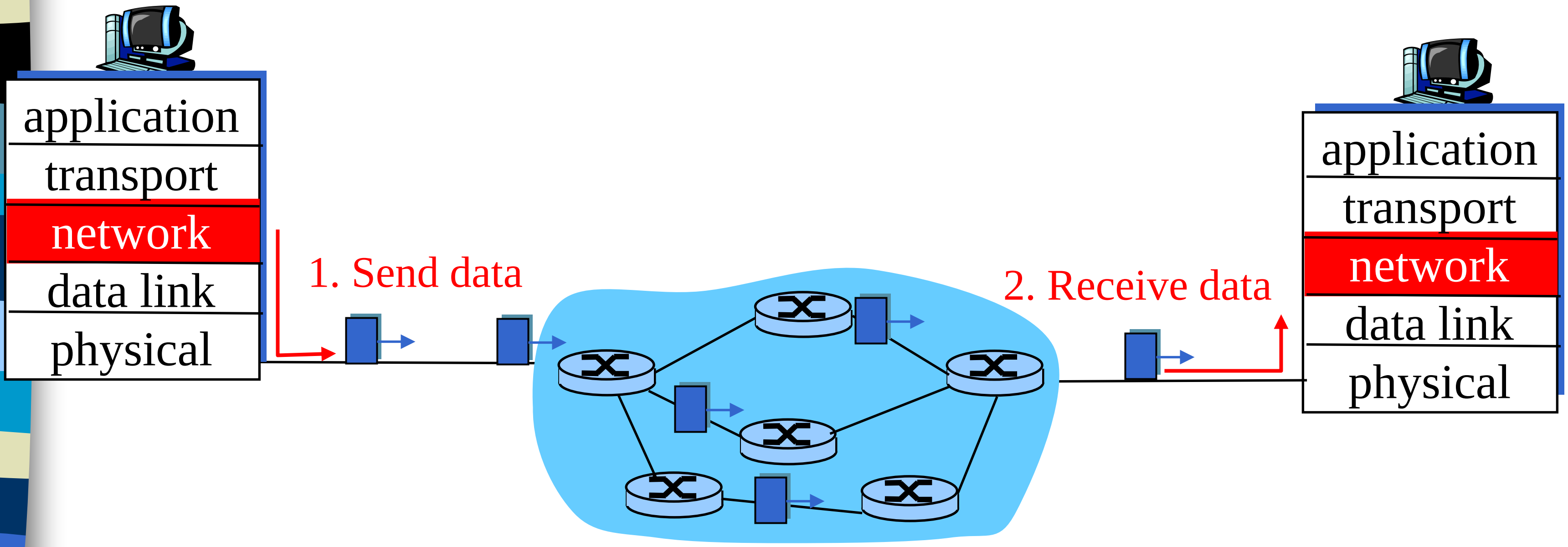
- Link state
- Distance Vector

4.6 Routing in the Internet

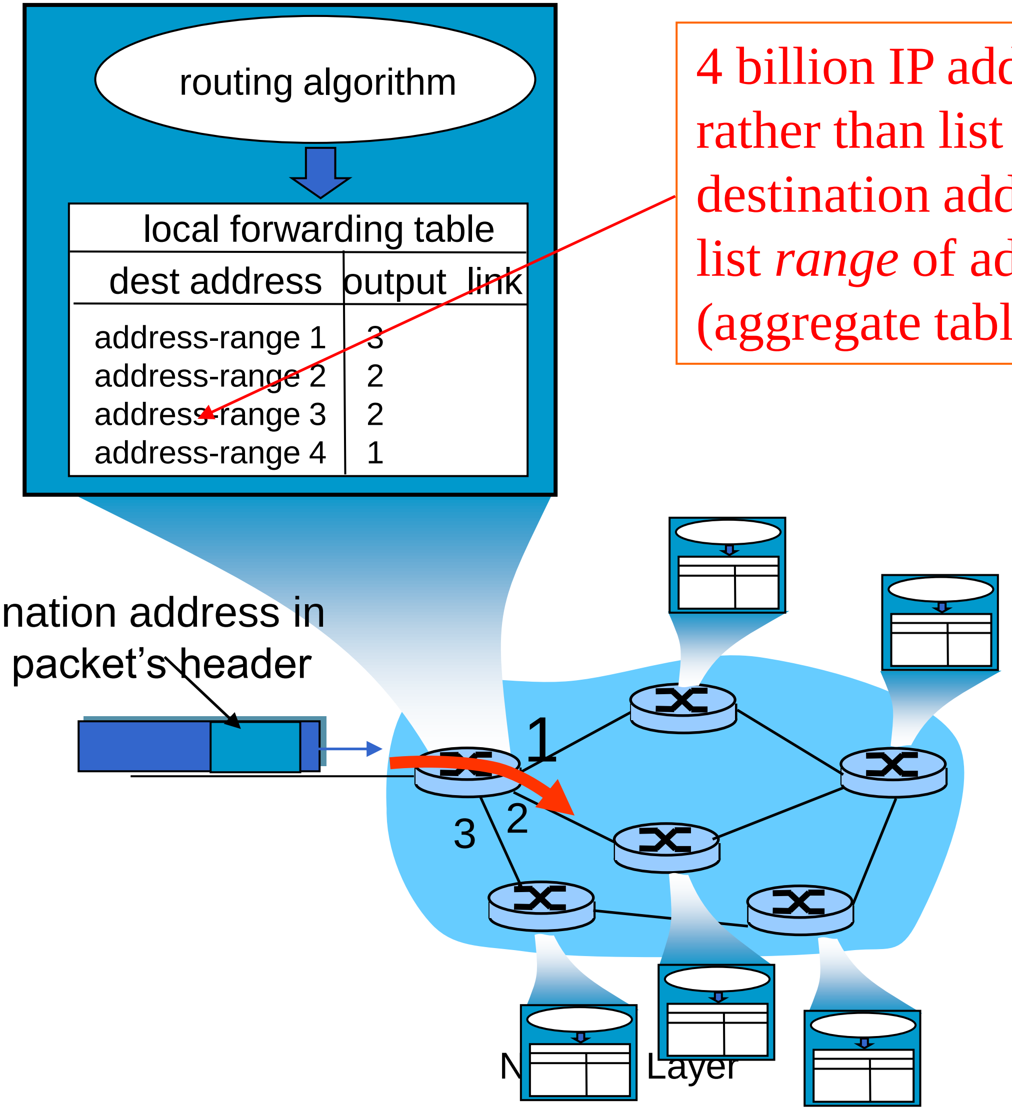
- RIP
- OSPF
- BGP

Datagram Networks

- No call setup at network layer
- Routers: no state about end-to-end connections
  - no network-level concept of “connection”
- Packets forwarded using destination host address
  - packets between same source-dest pair may take different paths



Datagram Forwarding Table



dest address	output link
address-range 1	3
address-range 2	2
address-range 3	2
address-range 4	1

4 billion IP addresses, so rather than list individual destination address list *range* of addresses (aggregate table entries)

IP destination address in arriving packet's header

Longest prefix matching

Longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
`11001000 00010111 00010*** *********`	0
`11001000 00010111 00011000 *********`	1
`11001000 00010111 00011*** *********`	2
otherwise	3

Examples:

DA: 11001000 00010111 00010110 10100001 Which interface?

DA: 11001000 00010111 00011000 10101010 Which interface?

Chapter 4: Network Layer

4. 1 Introduction

4.2 What's inside a router

4.4 IP: Internet Protocol

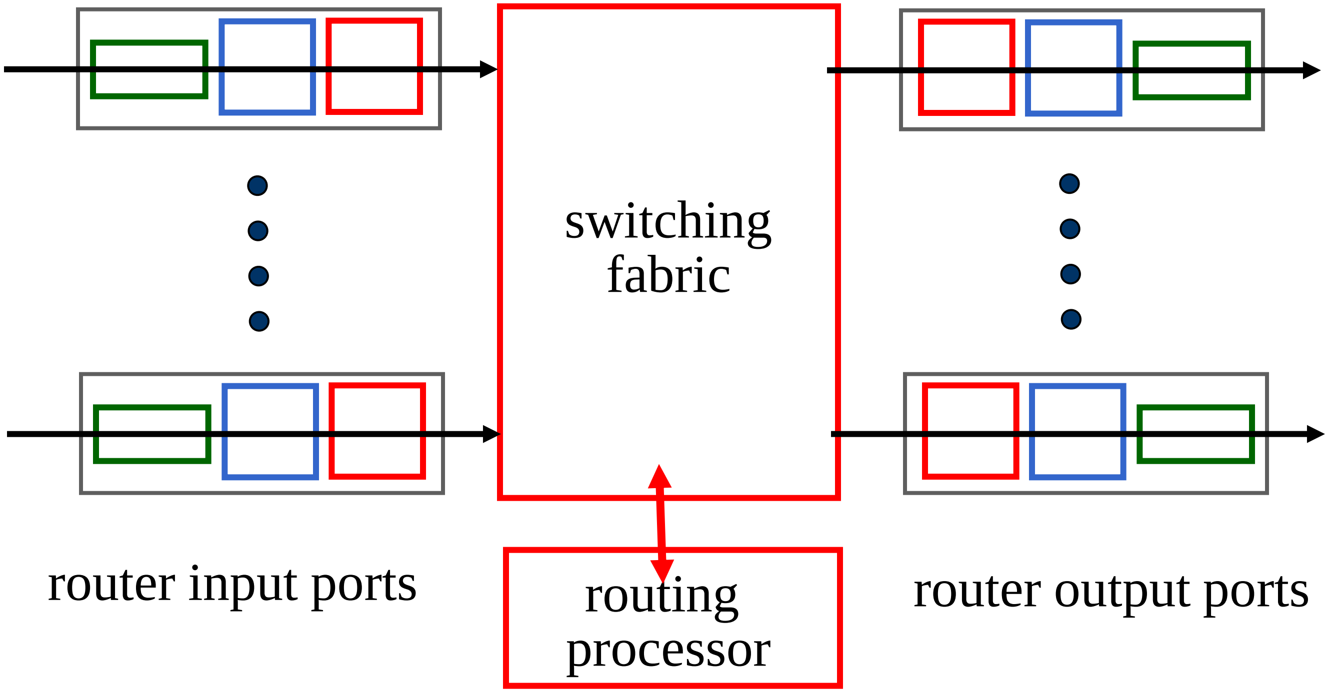
4.5 Routing algorithms
- Link state
- Distance Vector

4.6 Routing in the Internet
- RIP
- OSPF
- BGP

Router Architecture Overview

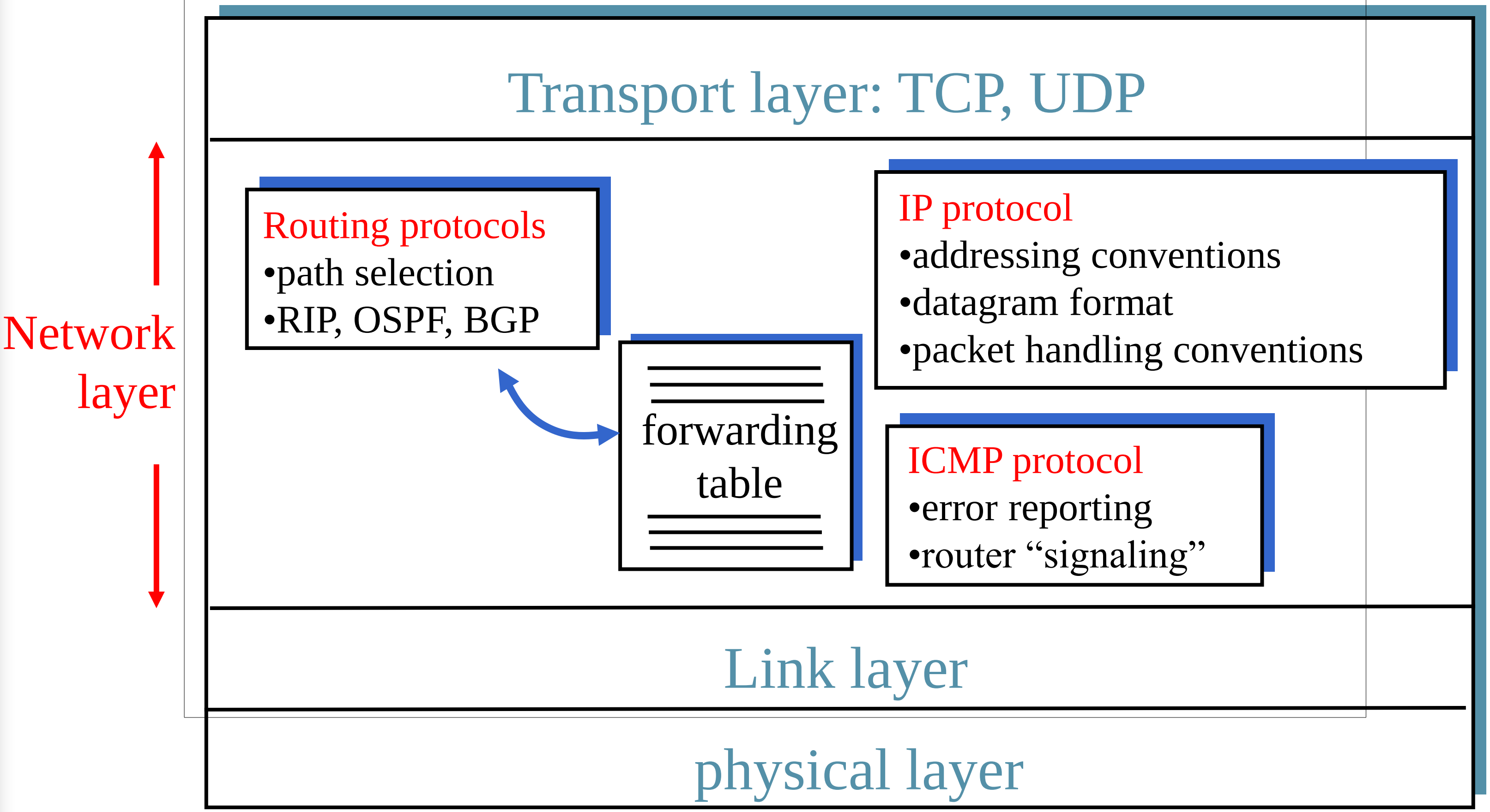
Two key router functions:

  - run routing algorithms/protocol (RIP, OSPF, BGP)
  - *forwarding* datagrams from incoming to outgoing link



The Internet Network Layer

Host, router network layer functions:



Transport layer: TCP, UDP

Network layer

Routing protocols
- path selection
- RIP, OSPF, BGP

forwarding table

IP protocol
- addressing conventions
- datagram format
- packet handling conventions

ICMP protocol
- error reporting
- router "signaling"

Link layer

physical layer

Chapter 4: Network Layer

4. 1 Introduction

4.2 Virtual circuit and datagram networks

4.3 What's inside a router

4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6

4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing

4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP

4.7 Broadcast and multicast routing

Chapter 4: Network Layer

4. 1 Introduction

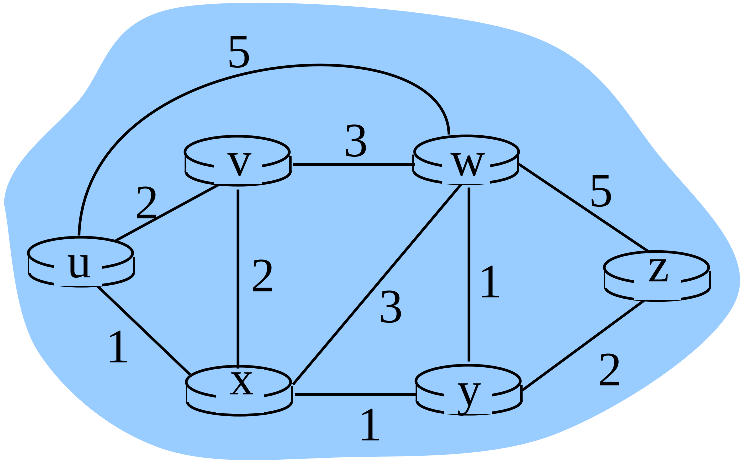
4.2 What’s inside a router

4.4 IP: Internet Protocol

4.5 Routing algorithms
- Link state
- Distance Vector

4.6 Routing in the Internet
- RIP
- OSPF
- BGP

Graph abstraction



Graph: G = (N,E)

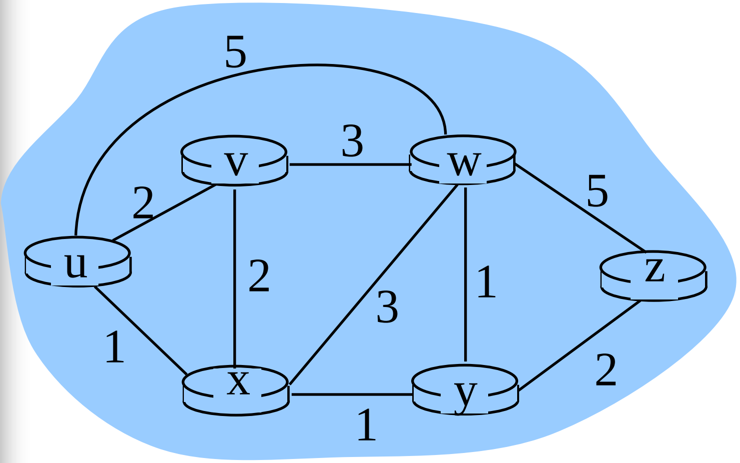
N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

Remark: Graph abstraction is useful in other network contexts

Example: P2P, where N is set of peers and E is set of TCP connections

Graph Abstraction: Costs



- c(x,x’) = cost of link (x,x’)
  - e.g., c(w,z) = 5

- Cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

Question: What’s the least-cost path between u and z ?

Routing algorithm: algorithm that finds least-cost path

Routing Algorithm Classification

Global or decentralized information?

Global:

- All routers have complete topology, link cost info
  - "Link state" algorithms

Decentralized:

- Router knows physically-connected neighbors, link costs to neighbors
- Iterative process of computation, exchange of info with neighbors
  - "Distance vector" algorithms

Static or dynamic?

Static:

- routes change slowly over time

Dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

Routing Algorithm Classification

Link State

- Topology information is flooded within the routing domain
- Best end-to-end paths are computed locally at each router
- Best end-to-end paths determine next-hops
- Based on minimizing some notion of distance
- Works only if policy is shared and uniform
- Examples: OSPF, IS-IS

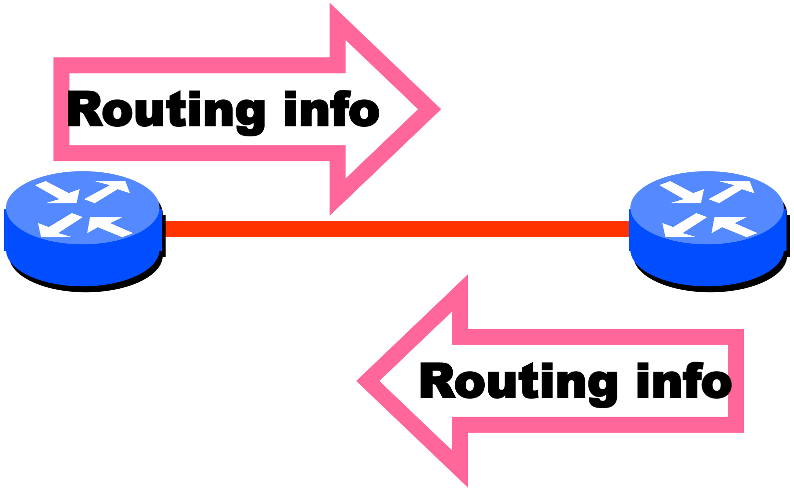
Vectoring

- Each router knows little about network topology
- Only best next-hops are chosen by each router for each destination network
- Best end-to-end paths result from composition of all next-hop choices
- Does not require any notion of distance
- Does not require uniform policies at all routers
- Examples: RIP, BGP

The Gang of Four

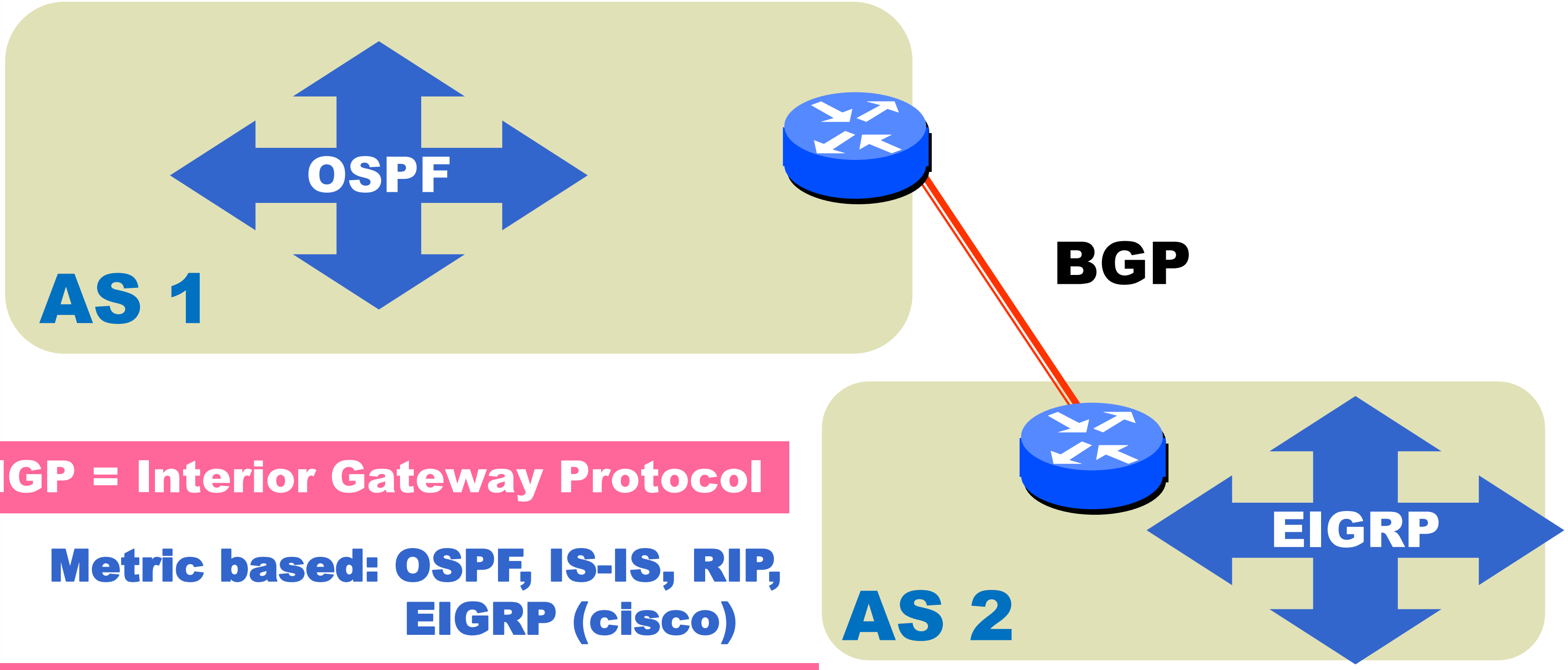
	Link State	Vectoring
IGP	OSPF IS-IS	RIP
EGP		BGP

Routers Talking to Routers



- Routing computation is distributed among routers within a routing domain
- Computation of best next hop based on routing information is the most CPU/memory intensive task on a router

Architecture of Dynamic Routing



IGP = Interior Gateway Protocol

Metric based: OSPF, IS-IS, RIP, EIGRP (cisco)

EGP = Exterior Gateway Protocol

Policy based: BGP

Chapter 4: Network Layer

4. 1 Introduction

4.2 What's inside a router

4.4 IP: Internet Protocol

4.5 Routing algorithms
  - Link state
  - Distance Vector

4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP

A Link-State Routing Algorithm

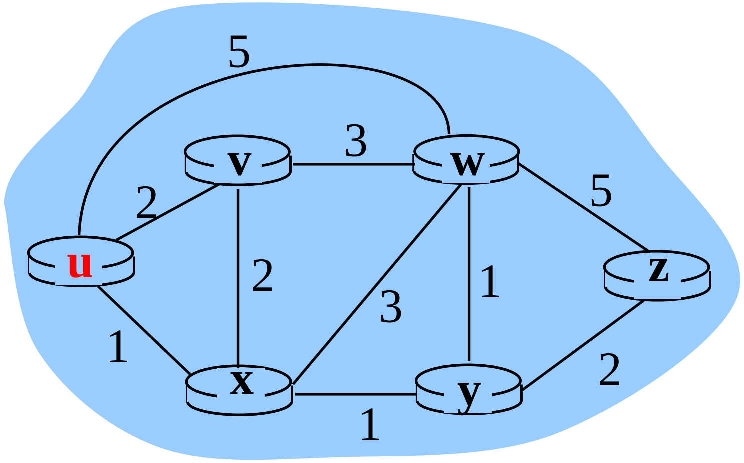
Dijkstra's Algorithm

- Net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- Computes least cost paths from one node ('source") to all other nodes
  - gives *forwarding table* for that node
- Iterative: after k iterations, know least cost path to k dest.'s

Notation

- c(x,y): link cost from node x to y = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijkstra’s Algorithm: Example

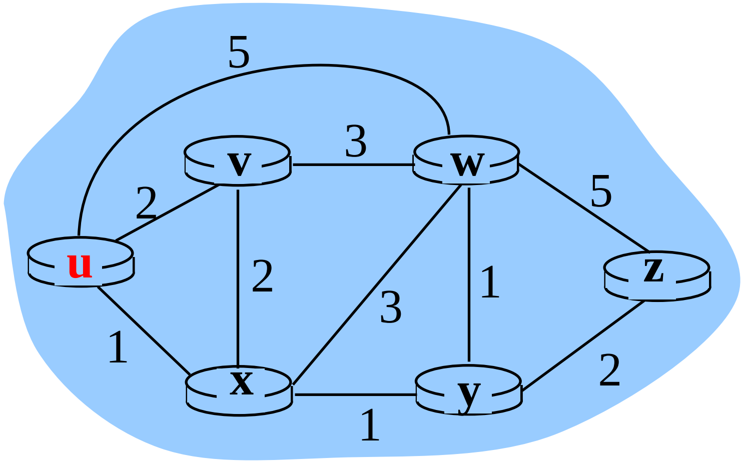


v, w and x are u’s neighbors

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞

Step 0: Add *D(v) for all v on the graph*

Dijkstra’s Algorithm: Example



Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞

Step 1:

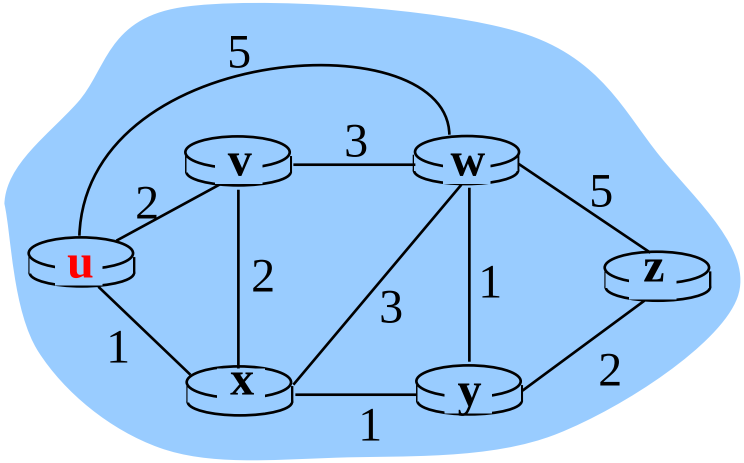
a. Find node *x not in N’ such that D(x) is a minimum*

b. Add *x in set N’*

c. Update *D(v) for all v adjacent to x and not in N’ using*

D(v) = min(D(v), D(x) + c(x,v)) //update costs to reach x’s neighbors

Dijkstra’s Algorithm: Example



Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y

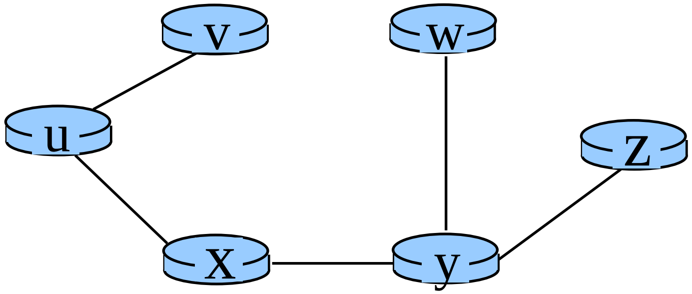
Step 2:

- Find node y not in N’ such that D(y) is a minimum; add y in N’
- Update costs to reach y’s neighbors v that are not in N’ using

D(v) = **min**(D(v), D(y) + c(y,v))

Dijkstra's algorithm: Example (2)

Resulting shortest-path tree from u:



Resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Dijsktra’s Algorithm

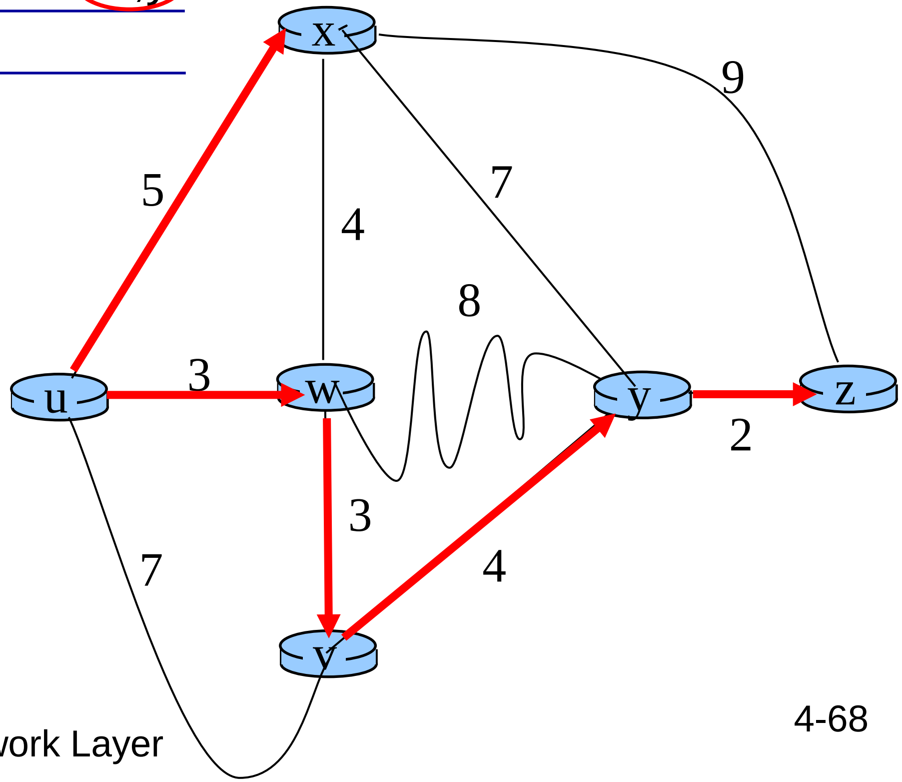
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
        then $D(v) = c(u,v)$
    else $D(v) = \infty$

Loop
  find w not in N' such that D(w) is a minimum
  add w to N'
  update D(v) for all v adjacent to w and not in N' :
      $D(v) = \min( D(v), D(w) + c(w,v) )$
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Dijkstra's algorithm: example

		D(v)	D(w)	D(x)	D(y)	D(z)
Step	N'	p(v)	p(w)	p(x)	p(y)	p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

Notes:

- Construct shortest path tree by tracing predecessor nodes
- Ties can exist (can be broken arbitrarily)



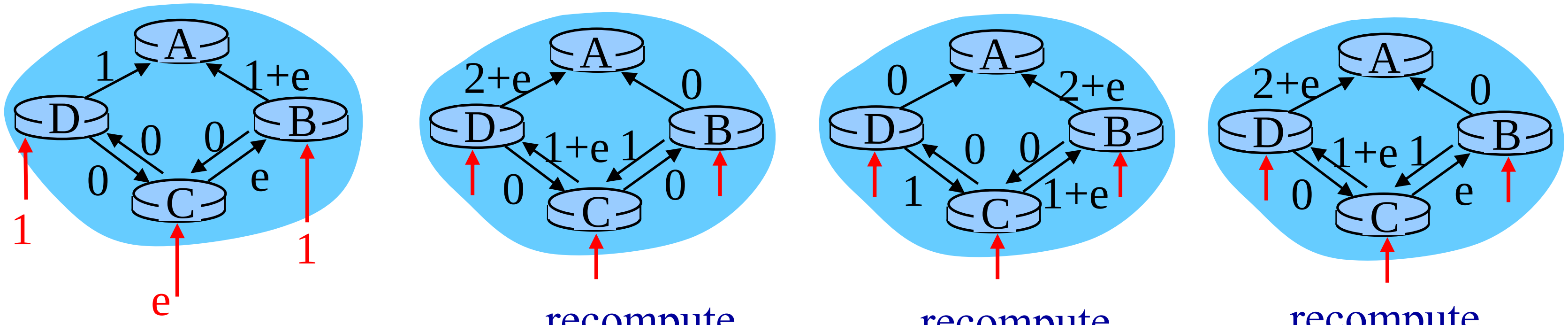
Dijkstra’s Algorithm: Discussion

Algorithm complexity: n nodes

- Each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: O(n^2)
- More efficient implementations possible: O(nlogn)

Oscillations possible:

- e.g., link cost = amount of carried traffic



initially

… recompute routing

… recompute

… recompute

D⟶A, B⟶A, C⟶A

Chapter 4: Network Layer

4. 1 Introduction

4.2 What's inside a router

4.4 IP: Internet Protocol

4.5 Routing algorithms

- Link state
- Distance Vector

4.6 Routing in the Internet

- RIP
- OSPF
- BGP

Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

- Define

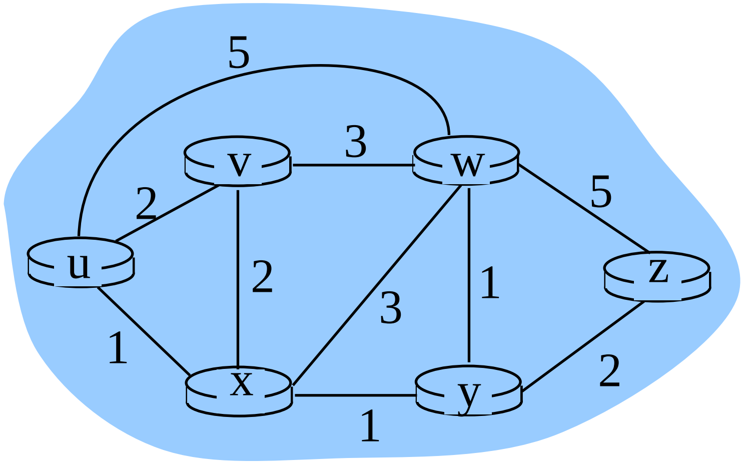
  $d_x(y)$:= cost of least-cost path from x to y

  Then

$$\boxed{d_x(y) = \min_v \{c(x,v) + d_v(y)\}}$$

where min is taken over all neighbors v of x

Bellman-Ford example



Clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$
\begin{aligned}
d_u(z) &= \min \{ c(u,v) + d_v(z), \\
&\qquad c(u,x) + d_x(z), \\
&\qquad c(u,w) + d_w(z) \} \\
&= \min \{2 + 5, \\
&\qquad 1 + 3, \\
&\qquad 5 + 3\} \;\; = 4
\end{aligned}
$$

Node that achieves minimum is next hop in shortest path ➜ forwarding table

Distance Vector Algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- Input to node x:
  - Node x knows cost to each neighbor v: c(x,v)
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains
    $\mathbf{D}_v = [D_v(y): y \in N]$

Distance vector algorithm (4)

Basic idea

- From time-to-time, each node sends its own Distance Vector (DV) estimate to neighbors
- When x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\} \quad \textit{for each node } y \in N$$

- Under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $\mathrm{d}_x(\mathrm{y})$

Distance Vector Algorithm (5)

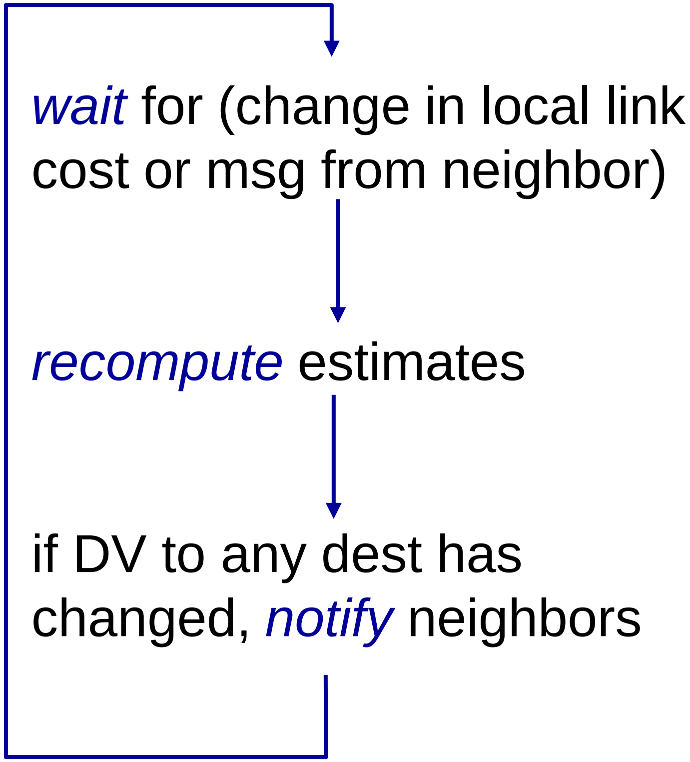
Iterative, asynchronous

- each local iteration caused by
  - local link cost change
  - DV update message from neighbor

Distributed:

- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

Each node



wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, *notify* neighbors

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

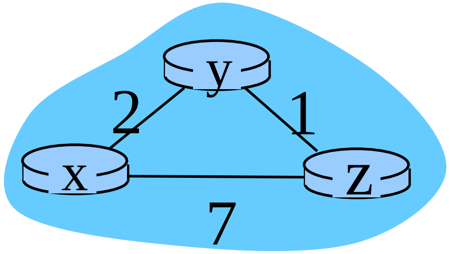
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

Each node sends its own DV to its neighbors

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

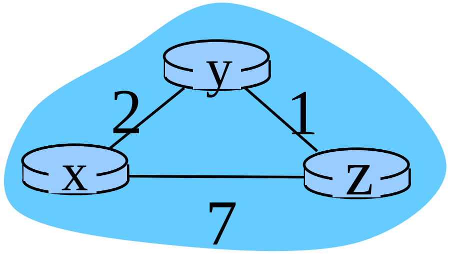
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
x	∞	∞	
y	∞	∞	∞
z	7	1	0



time

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$
$$= \min\{2+0 , 7+1\} = 2$$

→ the next-hop router to reach y is **y**

node x table

		cost to		
		x	y	z
from	x	0	2	7
	y	∞	∞	∞
	z	∞	∞	∞

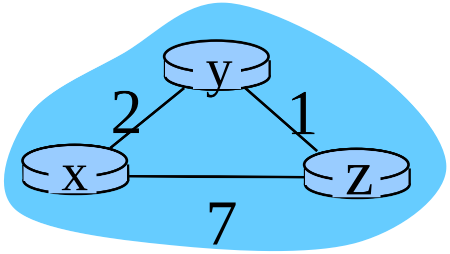
		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

node y table

		cost to		
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
	z	∞	∞	∞

node z table

		cost to		
		x	y	z
from	x	∞	∞	
	y	∞	∞	∞
	z	7	1	0



time

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$
$$= \min\{2+1 , 7+0\} = 3$$

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

→ the next-hop router to reach z is y

node y table

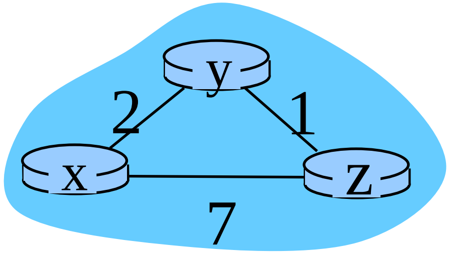
cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
x	∞	∞	
y	∞	∞	∞
z	7	1	0



time

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$
$$= \min\{2+0 , 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$
$$= \min\{2+1 , 7+0\} = 3$$

node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

cost to

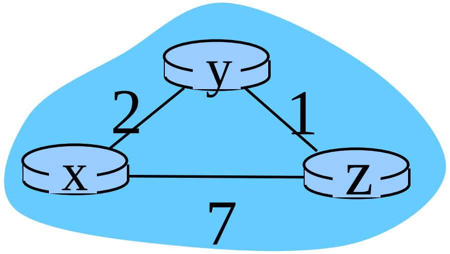
from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from	x	y	z
x	∞	∞	
y	∞	∞	∞
z	7	1	0

time



node x table

cost to

from

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

cost to

from

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

cost to

from

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

cost to

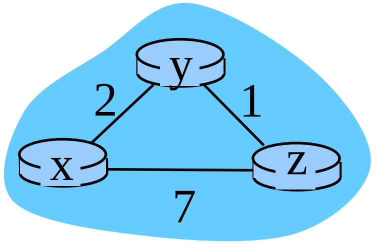
from

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0



time

node x table

cost to

from

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

cost to

from

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

cost to

from

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

node z table

cost to

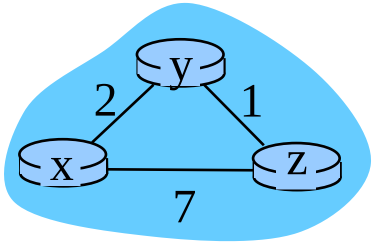
from

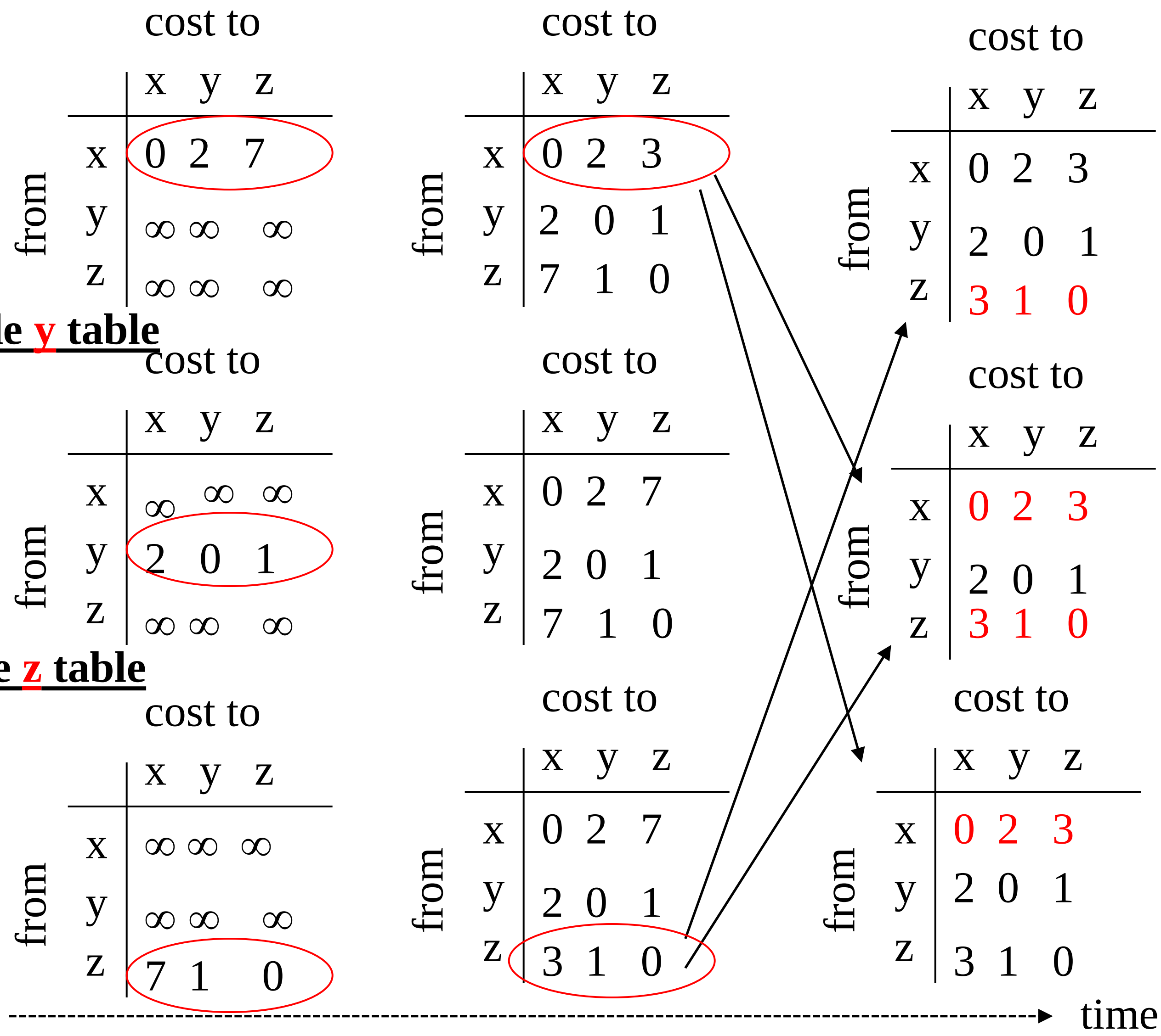
	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

cost to

from

	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0





node x table

cost to

from

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

cost to

from

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

cost to

from

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

cost to

from

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

cost to

from

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

cost to

from

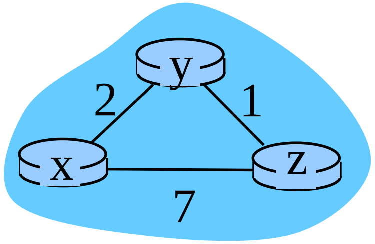
	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

cost to

from

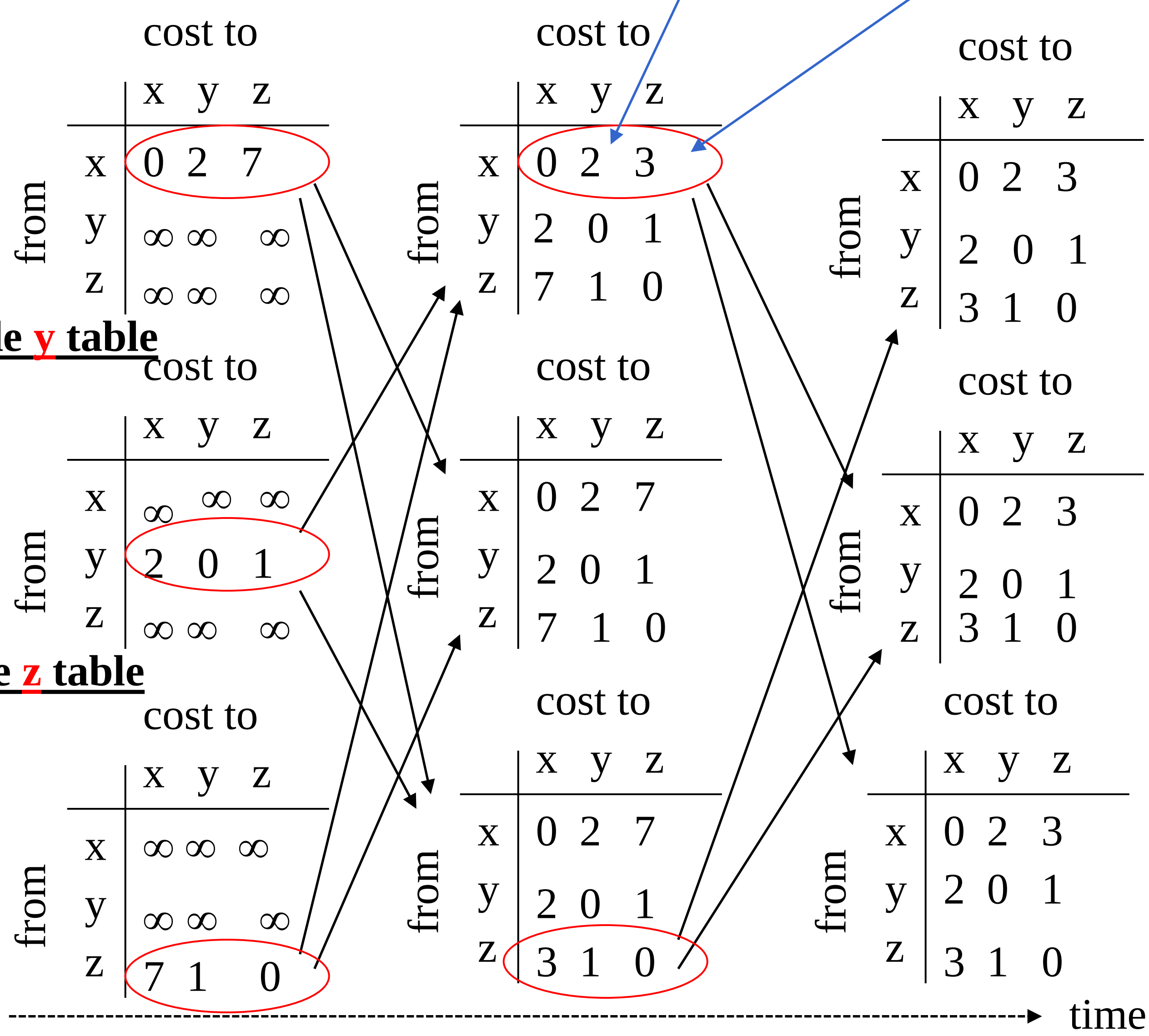
	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

time



$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$
$$= \min\{2+0 , 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$
$$= \min\{2+1 , 7+0\} = 3$$



node x table

cost to

from	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

cost to

from	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

cost to

from	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

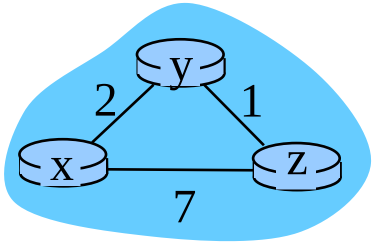
cost to

from	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

cost to

from	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

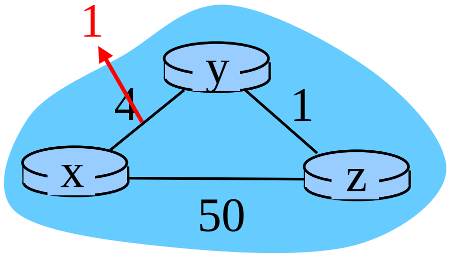
time



Distance Vector: Link cost changes

Link cost changes:

- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

Distance Vector Algorithm

- Input to x: *Node x knows cost to each neighbor v: c(x,v)*
  - Decentralized algorithm since it only needs local knowledge

- Distance vectors:
  - Node x *maintains distance (cost to y) vector* $\mathbf{D}_x = [D_x(y): y \in N]$
  - Node x *also maintains its neighbors' distance vectors:*

    $\mathbf{D}_v = [D_v(y): y \in N]$, for each neighbor *v of x*

- Iterative, asynchronous:
  - From time-to-time, each node *x sends its own DV estimate* $\mathbf{D}_x$ *to* neighbors
  - When a node *x receives new DV estimate from neighbor, it* updates its own DV using Bellman-Ford equation:

    $D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\}$ *where v* are neighbors of *x*,

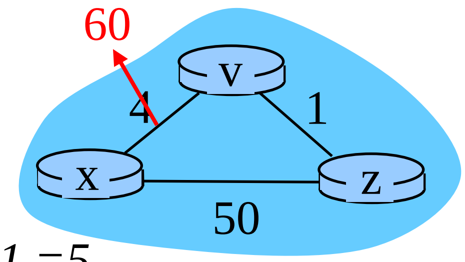
    (for each node $y \in N$*)*

- Output: least-cost paths from *x to all other nodes*

Distance Vector: link cost changes

Link cost changes:

- Good news travels fast
- Bad news travels slow - "count to infinity" problem!
- 44 iterations before algorithm stabilizes



  - Bef: $D_v(x) = 4$, $D_v(z) = 1$, $D_z(v) = 1$, $D_z(x) = 4+1 = 5$
  - Aft: $D_v(x) = \min\{60, c(v,z) + D_z(x)\} = 6$
  - Next: $D_z(x) = \ldots = c(z,v) + D_v(x) = 1+6=7$
  - Then: $D_v(x) = c(v,z) + D_z(x) = 1+7=8$

Poisoned reverse:

- If z routes through y to get to x :
  - z tells v: $D_z(x) = \infty$ *(so v won't route to x via z)*
    - $\longrightarrow D_v(x) = 60 \longrightarrow D_z(x) = 50 \longrightarrow D_v(x) = 51$
- Will this completely solve count to infinity problem?

Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) msgs sent
- DV: exchange between neighbors only
  - convergence time varies

Speed of Convergence

- LS: O(n^2) algorithm requires O(nE) msgs
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
  - error propagate thru network

Routing Algorithm Classification

Link State

- Topology information is flooded within the routing domain
- Best end-to-end paths are computed locally at each router
- Best end-to-end paths determine next-hops
- Based on minimizing some notion of distance
- Works only if policy is shared and uniform
- Examples: OSPF, IS-IS

Vectoring

- Each router knows little about network topology
- Only best next-hops are chosen by each router for each destination network
- Best end-to-end paths result from composition of all next-hop choices
- Does not require any notion of distance
- Does not require uniform policies at all routers
- Examples: RIP, BGP

The Gang of Four

	Link State	Vectoring
IGP	OSPF IS-IS	RIP
EGP		BGP

Protocoles et Interconnexions

Course Overview and Introduction

Dario Vieira

Department of Computer Science

EFREI