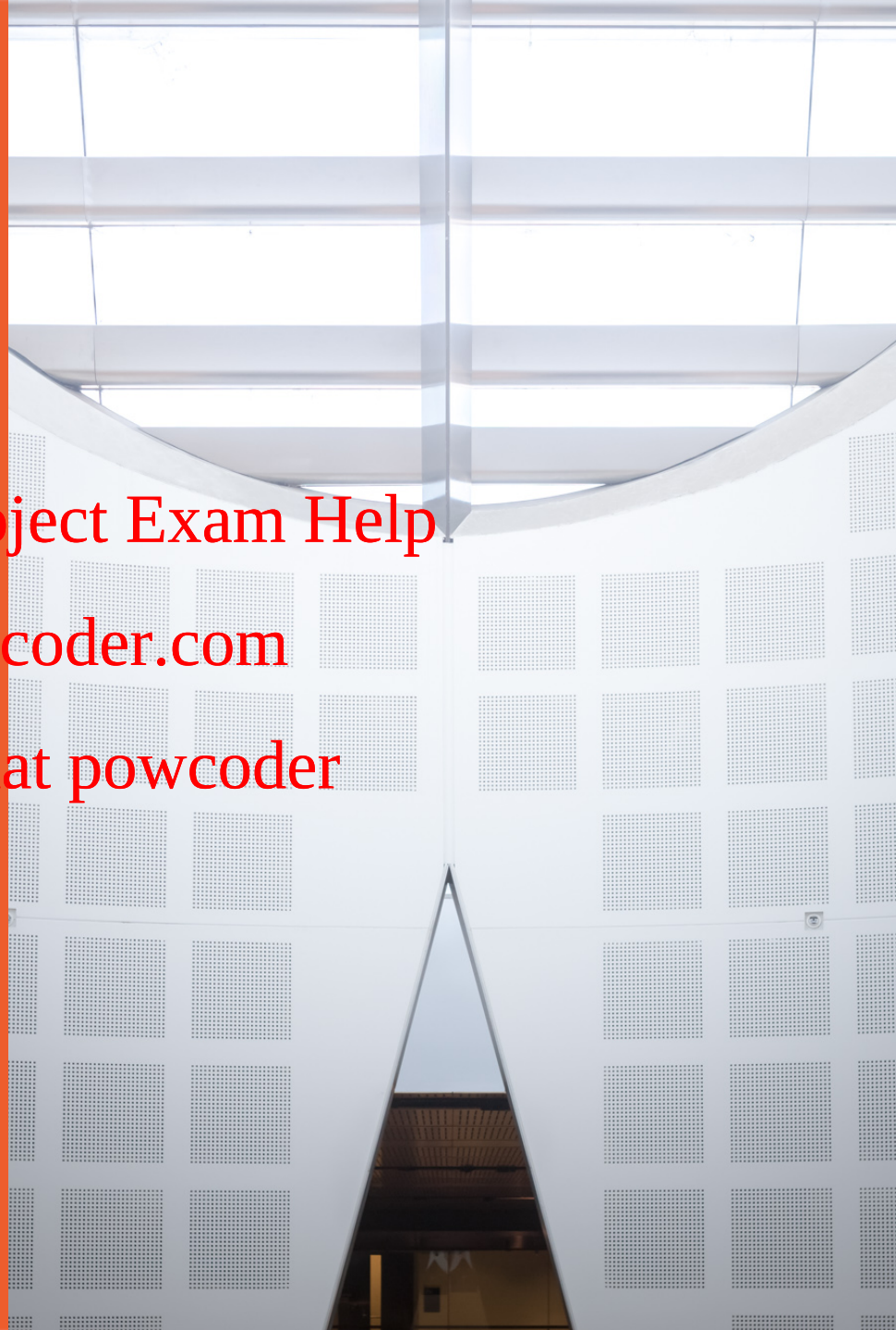
COMP3221: Distributed Systems

Communication-Routing

Dr Nguyen Tran
School of Computer Science

THE UNIVERSITY OF SYDNEY



Previously…

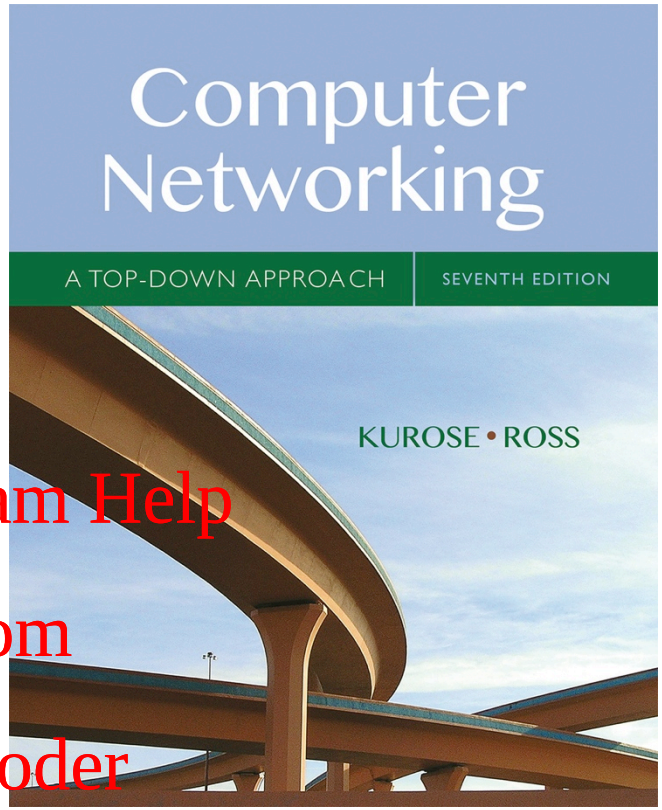
- Add questions here
- Previous lecture:
  - Shared memory allows multiple programs to communicate

- Today's lecture:

  - What if we don't have shared memory?
  - How to implement an alternative communication medium, like message channel?

Outline

- Layered Protocols
- Routing
  - Distance-vector
  - Link-state
- Message-Oriented Transient Communication
  - Socket
- Stream-Oriented Communication



Computer Networking: A Top Down Approach

7th edition
Jim Kurose, Keith Ross
Pearson/Addison Wesley
April 2016

Note: Some slides are adapted from J.F Kurose and K.W. Ross

Layered Protocols

Communication-Routing

Week 3, COMP3221

THE UNIVERSITY OF SYDNEY

Communication problem

Communicating is about transmitting encoded information

- Problem: two computers must agree on the code (language) they use
- Open standards give rules for everyone to communicate with each other

  - International Standards Organization (ISO)

The Open Systems Interconnection (OSI) reference model names communication levels, and assigns roles to each level

  - Internet Engineering Task Force (IETF)

The Request For Comments (RFC) are a public description of internet communication protocols (e.g., TCP->RFC793, UDP->RFC768, SMTP->RFC2821, ICMP->RFC792)

- Private "closed" protocols exist
  - Skype: people did reverse engineering to discover the protocol, find security issues, or to implement IM clients

Two modes of communication

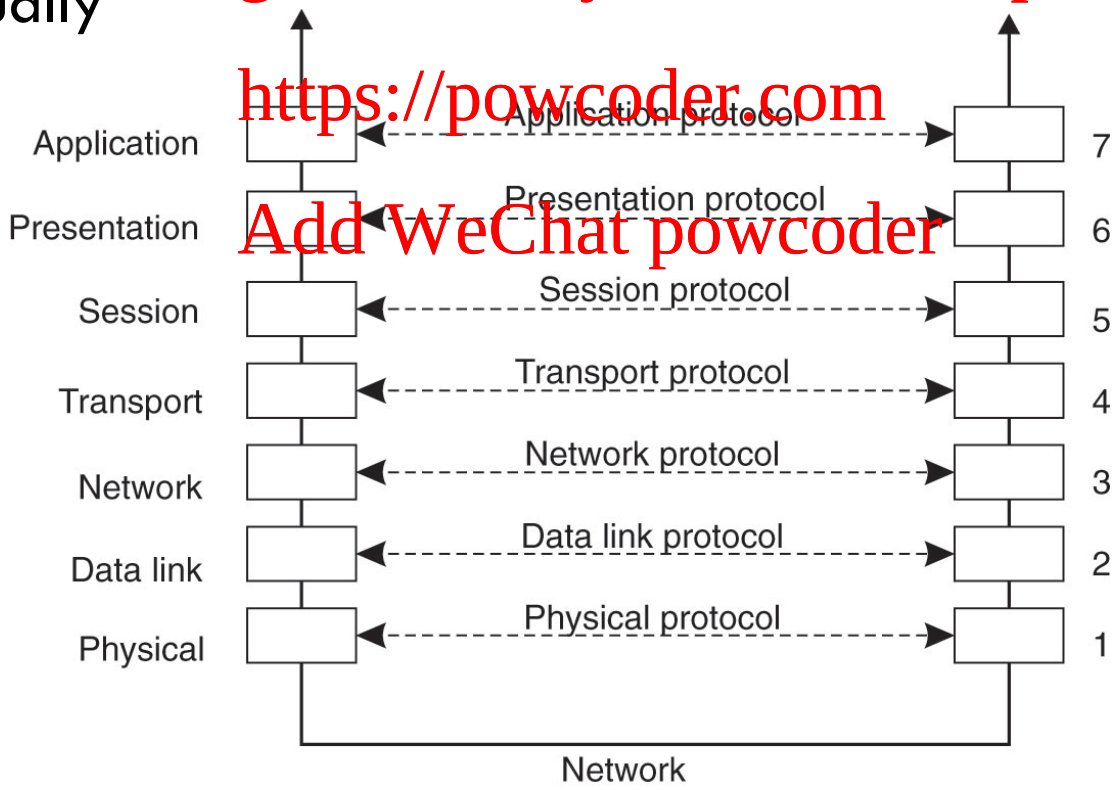
Connection oriented vs. Connectionless

- Connection oriented
  - Establish explicitly a connection with a partner before exchanging data
  - Protocol example: Transmission Control Protocol (TCP)
  - Application usage: File transfer, web browsing, email
- Connectionless
  - No setup in advance is needed
  - Protocol example: User Datagram Packet (UDP, IP)
  - Application usage: VoIP (Skype), IPTV

Layers

OSI *layers*

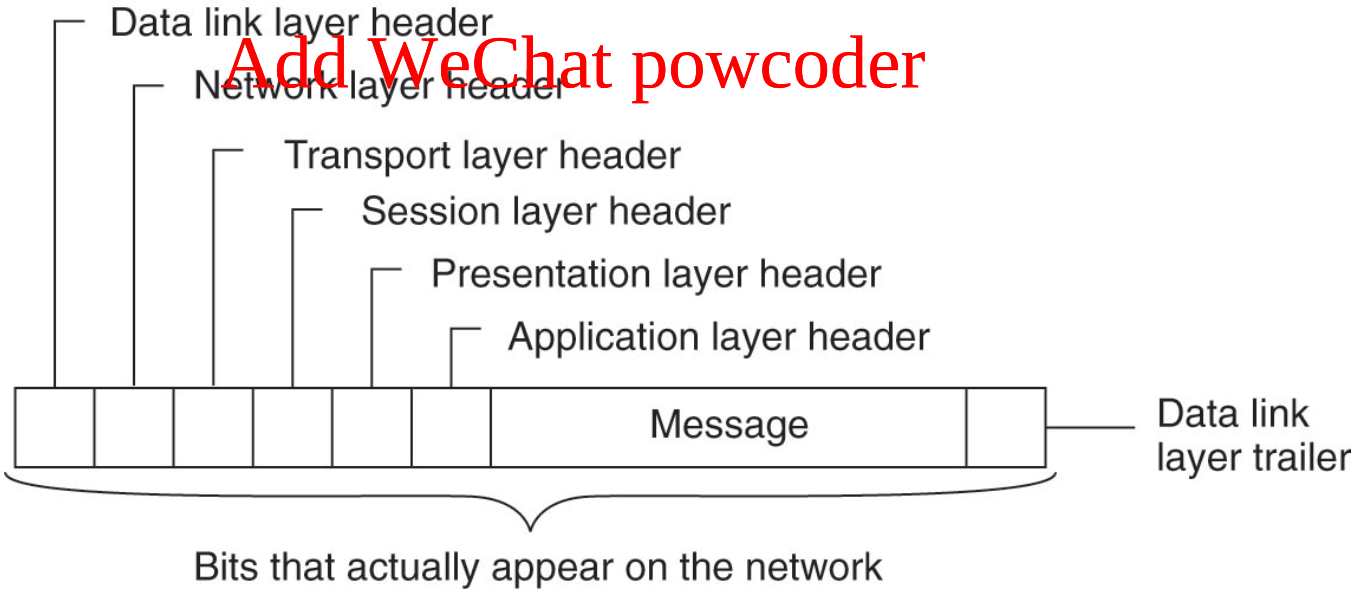
- Each layer deals with one specific aspect of the communication
- The problem is divided into sub-parts that can be implemented individually



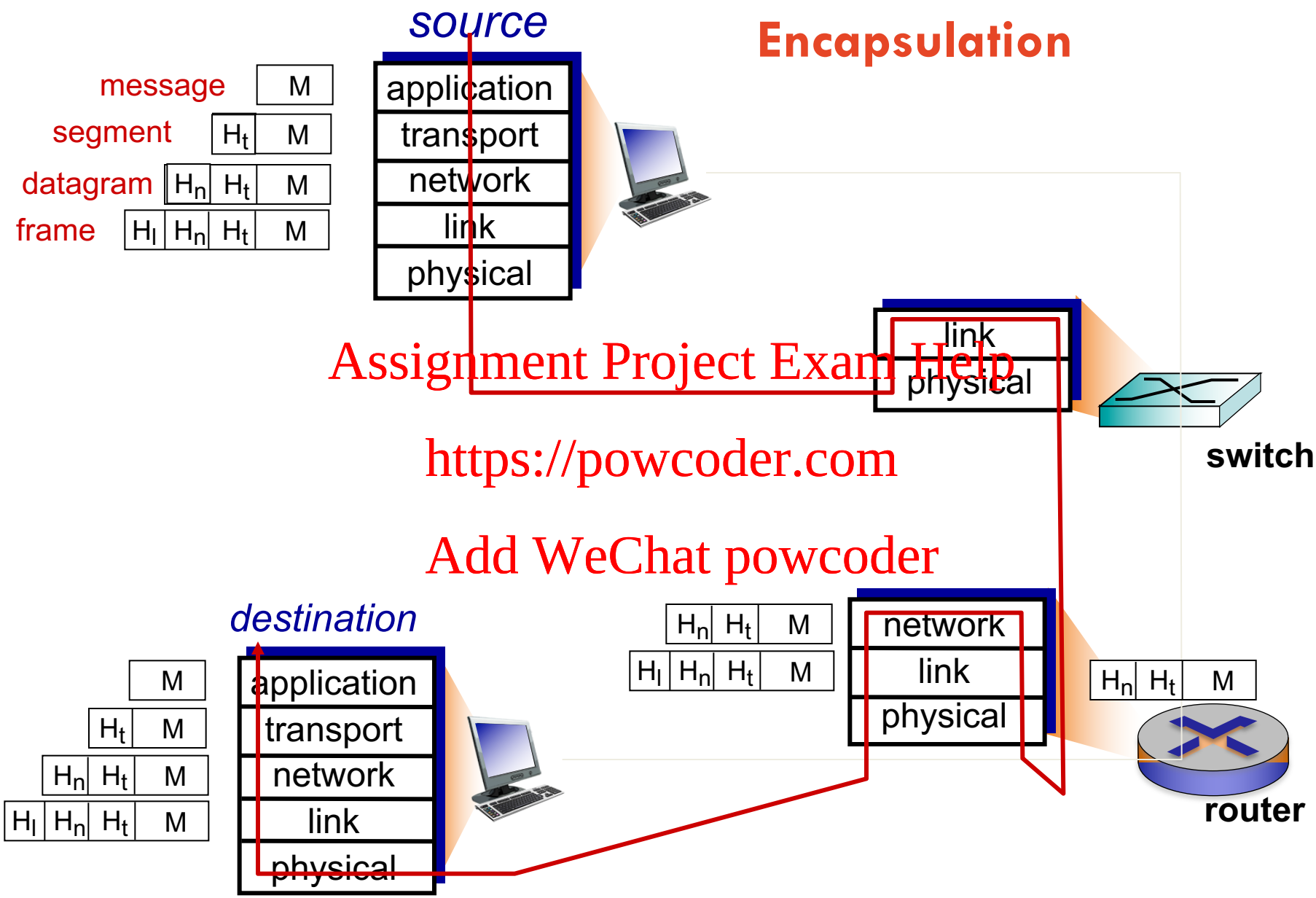
Layers

Message structure

- As the message go through lower layers before being sent, each layer adds its header to the message.
- Upon reception, the message is unmarshalled by the successive layers from bottom to top.



Encapsulation



Network communication

Low-level layers

- **Physical layer**
  - send bits (0 and 1)

- **Data link layer**
  - groups bits into frames
  - assigns sequence numbers to frames and adds special bits at the beginning and end
  - adds a checksum (the result of some operation on the frame content)
  - If receiver disagree about the checksum, then it asks the sender to resend
  - Example: Ethernet for *Local Area Network (LAN)* or PPTP in *Virtual private network (VPN)*

- **Network layer**
  - Routing of datagrams from source to destination
  - Example: *Internet Protocol* (IP) for *Wide Area Network* (WAN)

Network communication

Transport Layer

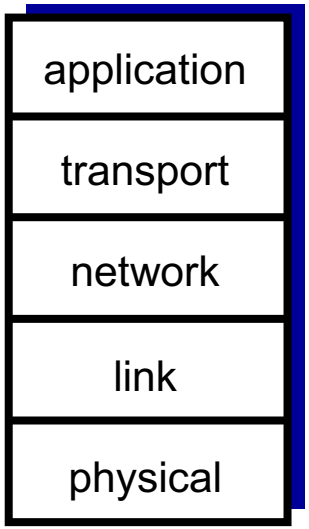
- Transport layer:
  - split the message from the application layer
  - number them
  - add the amounts of sent and remaining packets
- Reliable transport layer can be built on top of:
  - Connection-oriented protocol: packet would be ordered
  - Connectionless protocol: packet could be reordered
- Example of transport layer protocol: Transmission Control Protocol (TCP)
- Example of transport layer and network layer combination: TCP/IP

Network communication

High level layers

- OSI specifies three higher level layers:
- **Session**
  - synchronization, checkpointing, recovery of data exchange, dialog control
  - Example:
    - Domain Name Service (DNS)
    - Lightweight Directory Protocol (LDAP)
- **Presentation layer**
  - allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
  - Sometimes the session and presentations layers are omitted: Internet protocol suite.
- **Application**
  - Hypertext Transfer Protocol (HTTP)
  - File Transfer Protocol (FTP)
  - TCP/IP Terminal Emulation Protocol (Telnet)
  - X-Window

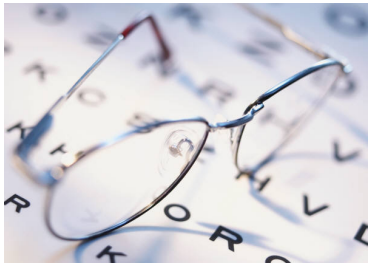
Internet protocol stack



Network communication

Analogy: Mail service vs. Web service

- Application (no clue of intermediary steps):
  - You post a letter with some address, the receiver reads it
  - HTTP: A user types a URL in a browser, a server sends back the web page
- Transport (error control):
  - Upon writing a wrong address on a letter, the letter will be sent back to you
  - TCP initialized a connection, checks for potential errors and may retransmit
- Internet (recipient is unknown):
  - An airplane moves letters between cities without knowing the recipients
  - IP brings packets over the WAN potentially from one LAN to another
- Data link:
  - Trucks move letters within a city
  - Ethernet handles transmission within the LAN
- Physical layer:
  - Use pen to write and glasses to read letters
  - Specifying fiber, wire, radio to transmit the one and zero encoding the message



Routing

Communication
Week 3, COMP3221

THE UNIVERSITY OF SYDNEY

Routing Problem

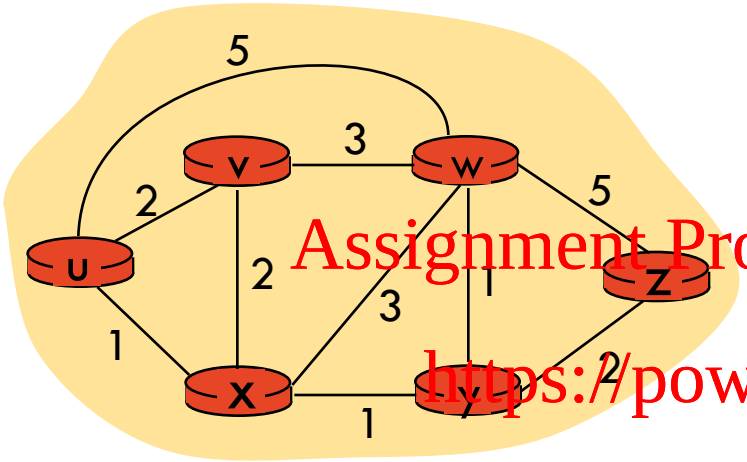
What are routing protocols used for?

- Dijkstra thought about the shortest path algorithm in Amsterdam when trying to find his way [Frana&Misa, CACM'10]
- Related to path finding problem in graphs
- Graph nodes represent internet routers, edges are communication links

GPS clients use shortest path algorithms

- Routing is necessary in all networks except Local Area Networks (LANs) where Ethernet provides direct communication between all pairs of attached hosts
- The routing protocol is implemented in the network layer of each router to determine the route for the transmission of packets to their destination

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

key question: what is the least-cost path between u and z ?
routing algorithm: algorithm that finds that least cost path

Routing algorithm classification

Q: 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

Q: static or dynamic?

Static:

- routes change slowly over time

Dynamic:

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

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

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

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

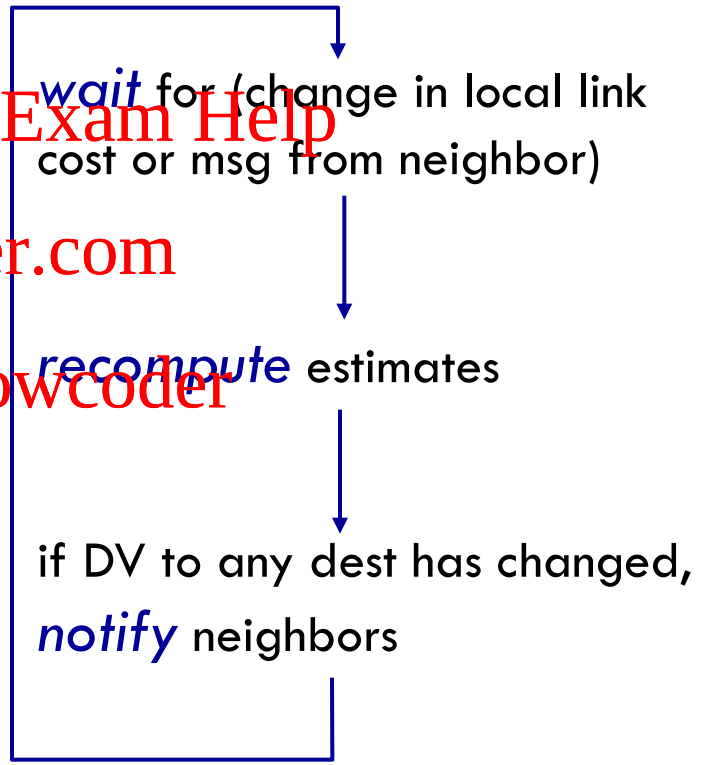
Distance vector algorithm

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



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

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

node x table

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

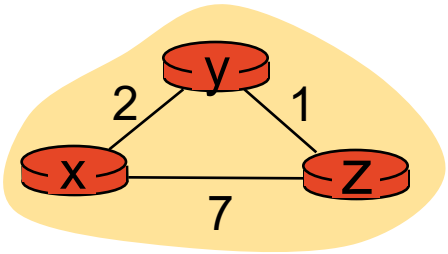
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

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

node z table

from \ cost to	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

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

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

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

node y table

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

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

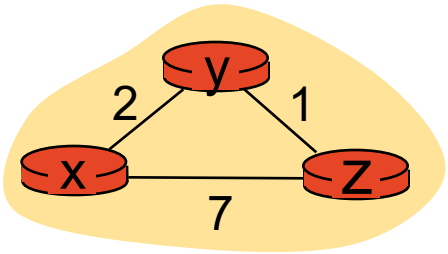
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

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

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

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

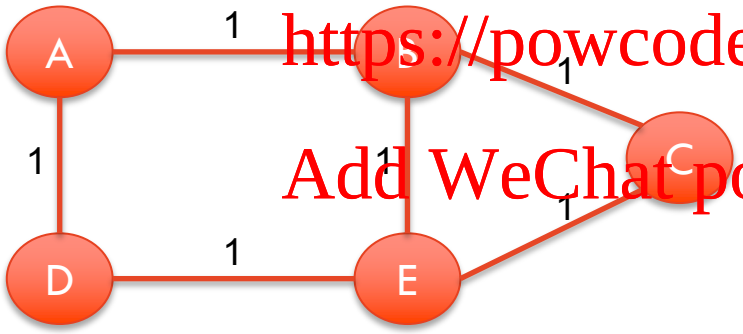


time

Distance-vector routing algorithm

Router Information Protocol (RIP)

- Mainly used in internet up to 1979
- Relies on Bellman-Ford algorithm: Bellman algorithm [1957] distributed by Ford and Fulkerson [1972]
- Link cost = 1



Routing table of router A		
Dest.	**Dir**	**Cost**
A	local	0
B	B	1
C	B	2
D	D	1
E	B	2

- If a node detects a link failure, it sets ∞ as the associated cost of such a link and sends its local table to neighbours
- *Eventually (when failures stop) each router gets for each destination the direction leading to the minimal cost*

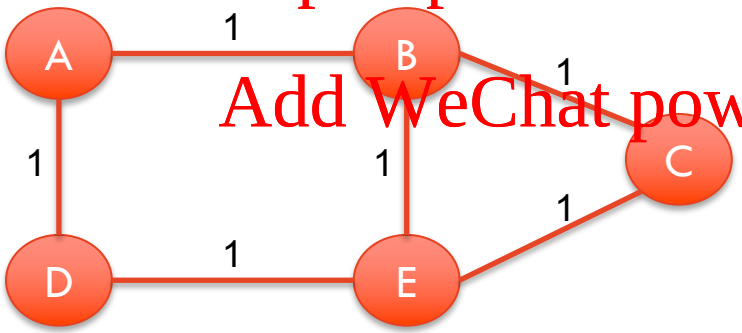
Distance-vector routing algorithm

RIP (con't)

Let's re-start the algorithm from scratch: all routing tables are empty

Routing table of router A		
Dest.	**Dir**	**Cost**
A	local	0

Routing table of router B		
Dest.	**Dir**	**Cost**
B	local	0



Routing table of router C		
Dest.	**Dir**	**Cost**
C	local	0

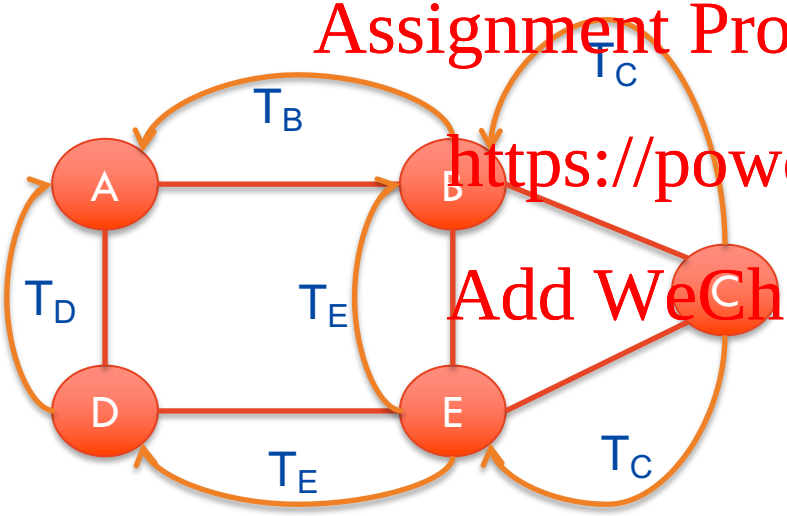
Routing table of router D		
Dest.	**Dir**	**Cost**
D	local	0

Routing table of router E		
Dest.	**Dir**	**Cost**
E	local	0

Distance-vector routing algorithm

RIP (con't)

- Example: How is routing table A built from scratch?

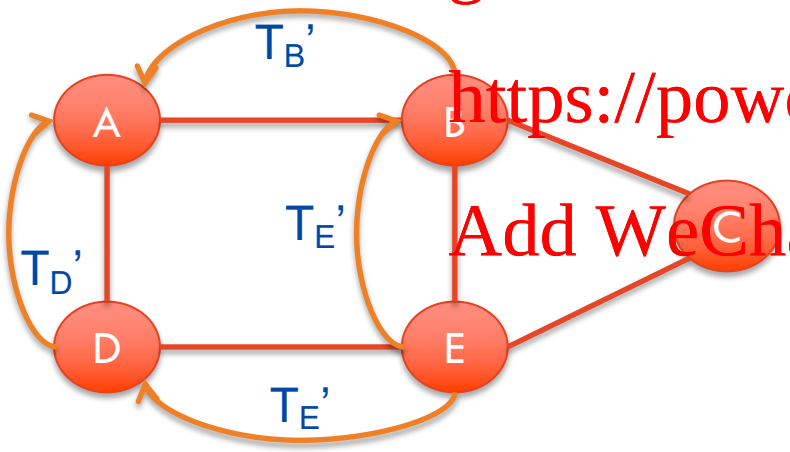


Routing table of router A		
Dest.	Dir	Cost
A	local	0
B	B	1
D	D	1

Distance-vector routing algorithm

RIP (con't)

- Example: How is routing table A built from scratch?

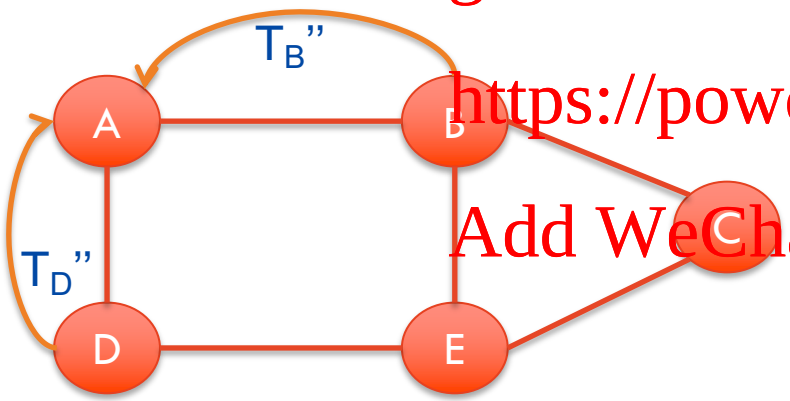


Routing table of router A		
Dest.	Dir	Cost
A	local	0
B	B	1
C	B	2
D	D	1
E	B	2

Distance-vector routing algorithm

RIP (con't)

- Example: How is routing table A built from scratch?



Routing table of router A		
Dest.	Dir	Cost
A	local	0
B	B	1
C	B	2
D	D	1
E	B	2

RIP algorithm

- Advantages:
  - Simple
  - Efficient in small networks
- Limitations:
  - Costs based on the number of hops (bandwidth of links counts)
  - No big deal: this is just a matter of defining link weights (support negative values)
  - Inefficient in large networks as loops may occur before the convergence state is reached

A link-state routing algorithm

Dijkstra's algorithm *(1956):* find the shortest path from a node i to other nodes

- 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
- Used in IS-IS (IP) and OSPF protocols

Dijsktra's algorithm

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5          then D(v) = c(u,v)
6      else D(v) = ∞
7
8  Loop
9     find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N' :
12       D(v) = min( D(v), D(w) + c(w,v) )
13    /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15 until all nodes in N'
```

Notation:

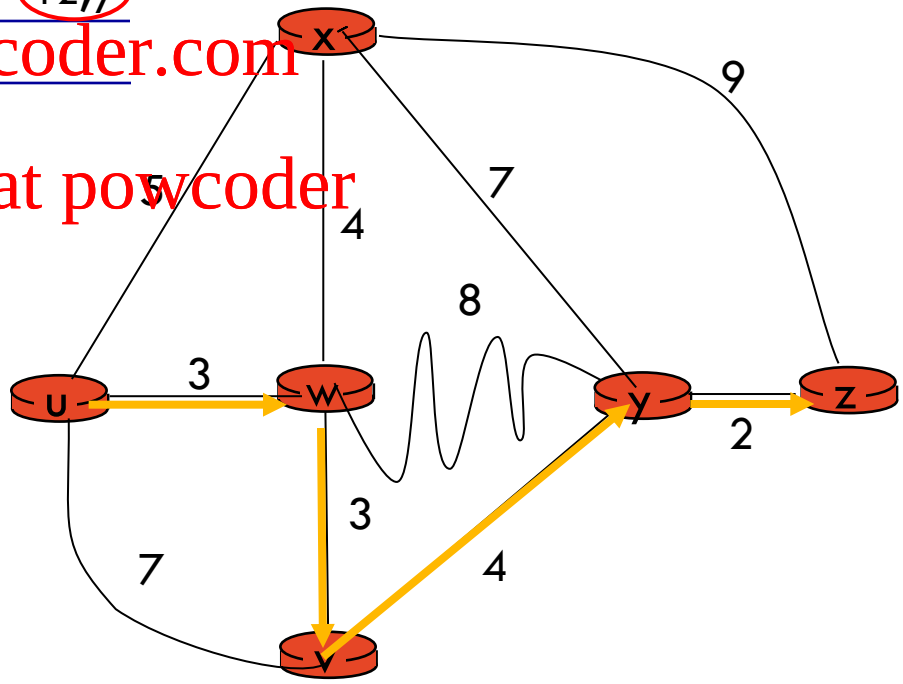
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijkstra's algorithm: example

Step	N'	D(v), p(v)	D(w), p(w)	D(x), p(x)	D(y), p(y)	D(z), 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					

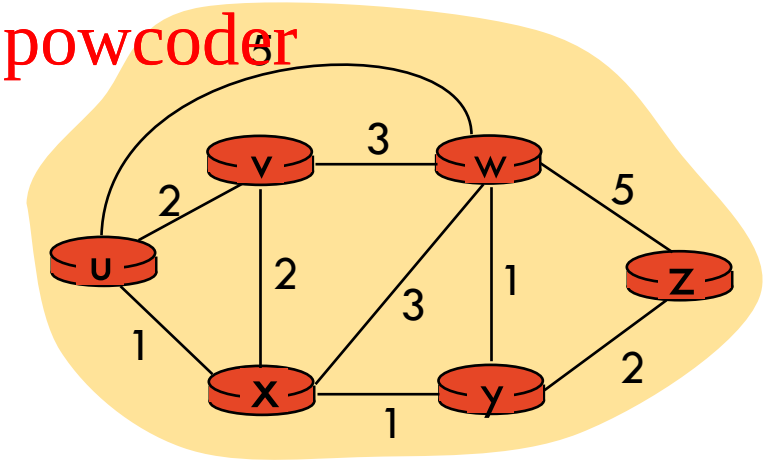
Notes:

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



Dijkstra's algorithm: another 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
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

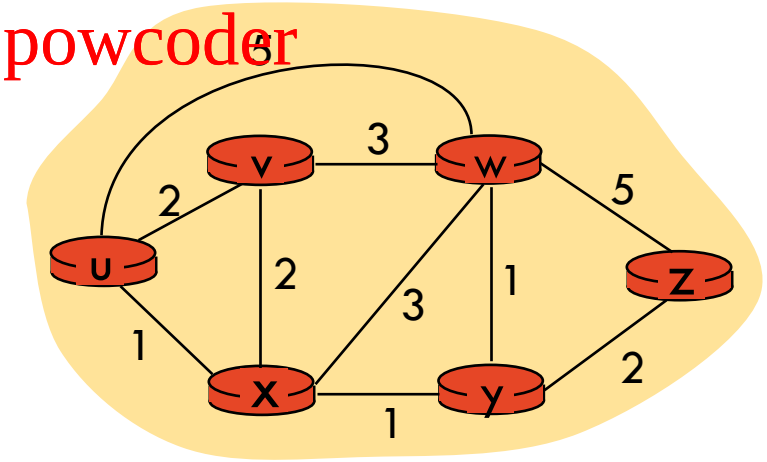


Dijkstra's algorithm: another example

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
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Resulting shortest-path to w:

- w ← y ← x ←u



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)

Link-state routing algorithm

OSPF (Open Shortest Path First)

- "open": publicly available
- Uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - **route computation using Dijkstra's algorithm**
- router floods OSPF link-state advertisements to all other routers in *entire* AS
  - carried in OSPF messages directly over IP (rather than TCP or UDP
  - link state: for each attached link
- multiple same-cost paths allowed (only one path in RIP)

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

Path finding in large scale networks

- Many graph libraries are available
  - Python: NetworkX - https://networkx.github.io

- Efficient implementations for many commonly known graph algorithms are available

- Example: Dijkstra's shortest path


```
>>> G = nx.path_graph(5)
>>> print(nx.shortest_path(G, source=0, target=4))
[0, 1, 2, 3, 4]
>>> p = nx.shortest_path(G, source=0) # target not specified
>>> p[4]
[0, 1, 2, 3, 4]
>>> p = nx.shortest_path(G, target=4) # source not specified
>>> p[0]
[0, 1, 2, 3, 4]
>>> p = nx.shortest_path(G) # source, target not specified
>>> p[0][4]
[0, 1, 2, 3, 4]
```

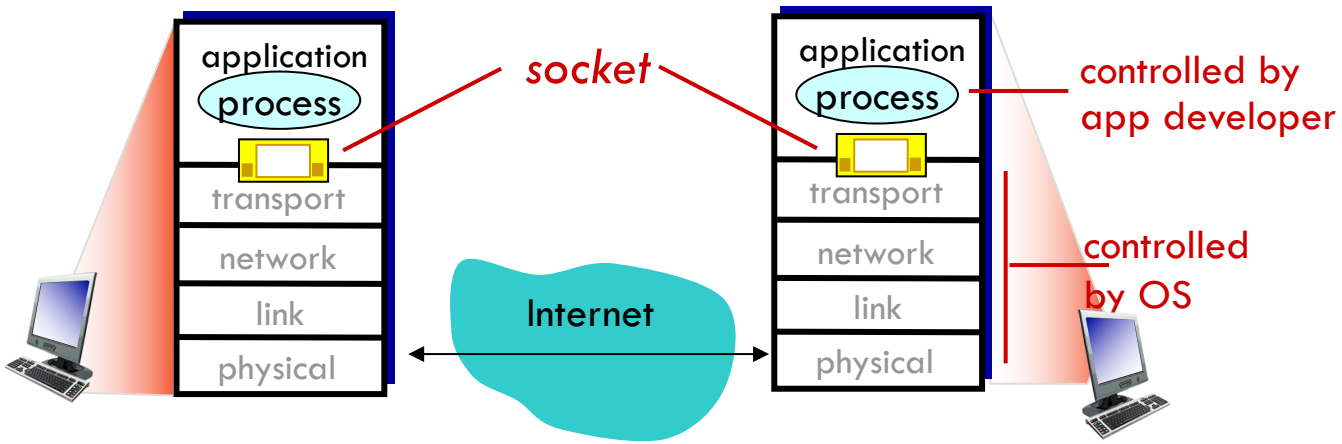
Message-Oriented Transient Communication

Socket

Communication
Week 3, COMP3221

THE UNIVERSITY OF SYDNEY

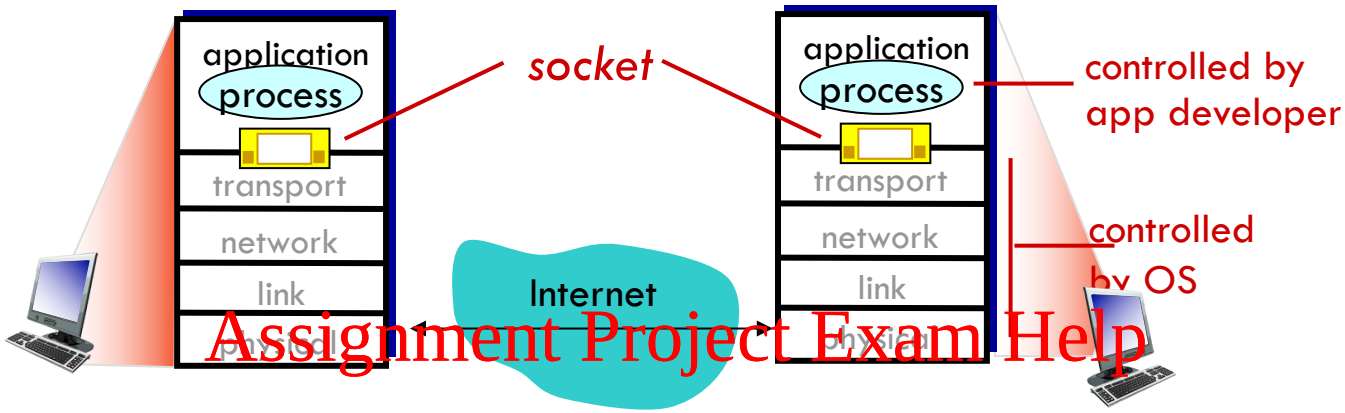
Socket



- *Socket*: interface between application and network: communication end-points write data sent out over the network, and incoming data can be read

  - Application creates a socket
  - Socket type dictates the style of communications: connection less/oriented
- A socket is identified by an IP address concatenated with a port number
- In general, sockets use the client-server model:
  - The server waits for incoming client requests by listening to a specified port
  - Once a request is received, the server accepts a connection from the client

Address, Port, and Socket



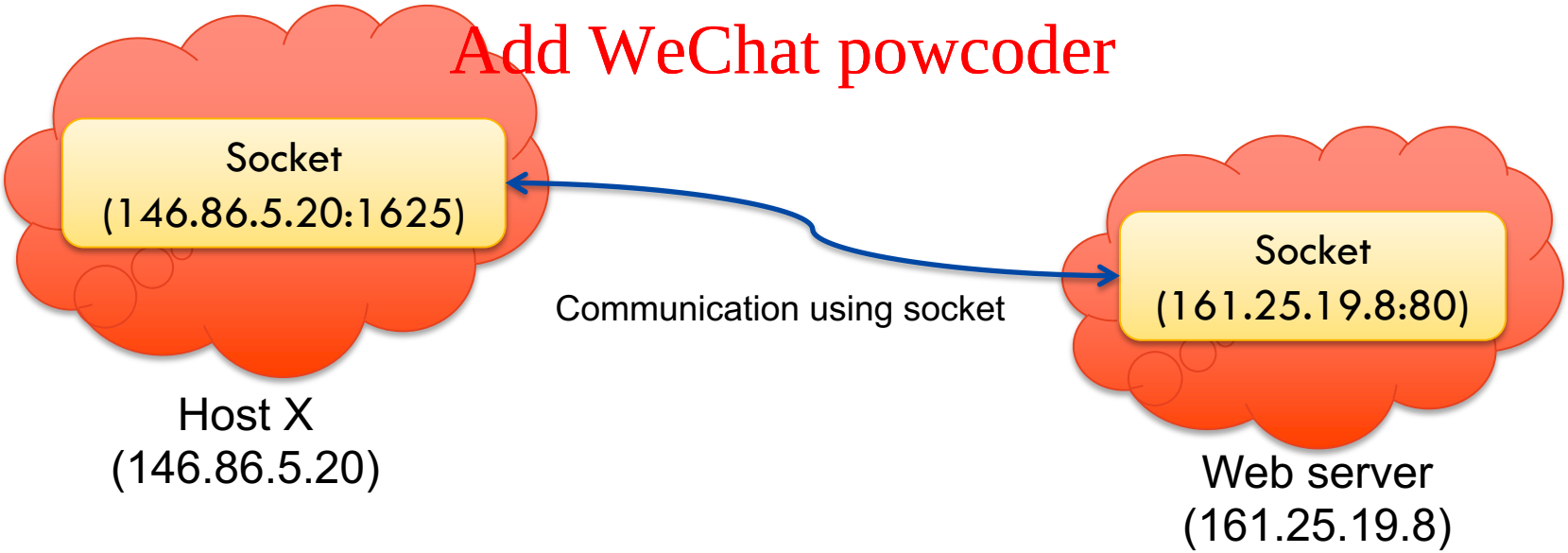
- *Like Apartment and Mailboxes*
  - You are the application
  - Your apartment building address is address
  - Your mailbox is the port
  - Socket is the key giving you access to the right mailbox

- Q: How to choose which port a socket connects to ?

Socket

Port

- Port numbers <1024 are for specific service protocols
  e.g., 80:HTTP, SSH:22, FTP:21, SMTP:25
- A client initiating a connection is assigned a free port number (>1024) by its host

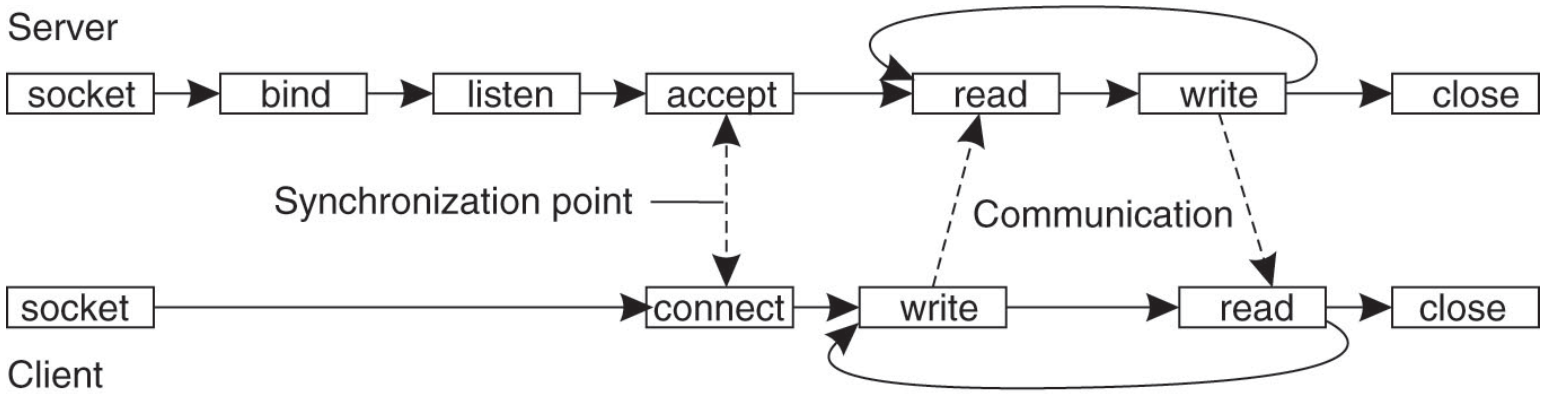


Socket

Berkley Sockets: Socket interface as proposed in Berkley UNIX in the 70's

1. Servers generally execute the first 4 primitives on accept
2. Bind associates the newly created socket to a local address and a port; the client binds implicitly to any available port
3. The client connects to a specified address
4. Once the connection is accepted by the server, the client and server can communicate with send and receive.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



Stream-Oriented Communication

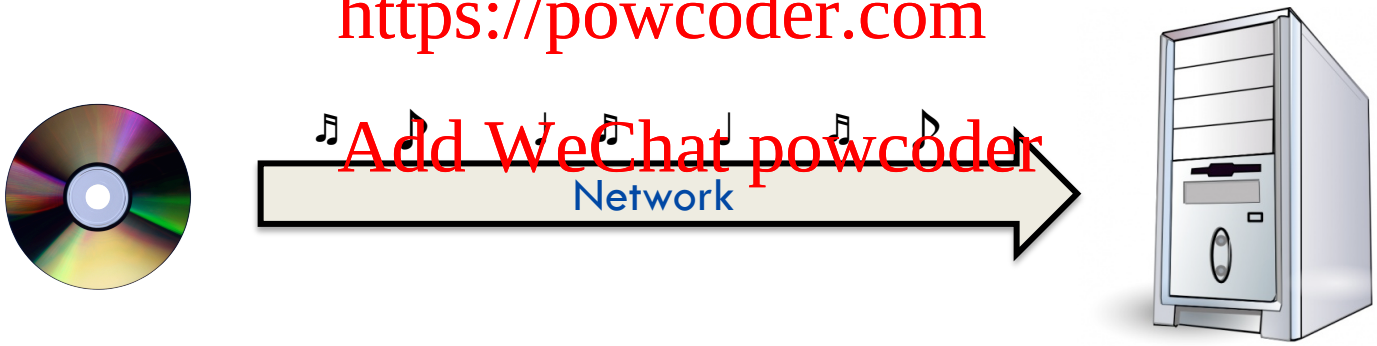
Communication
Week 3, COMP3221

THE UNIVERSITY OF SYDNEY

Stream-oriented communication

When timing is crucial

- Previous communication modes do not guarantee transmission rate
- In stream-oriented communication, the stream should not be interrupted, i.e., messages should keep being received at regular (typically small) time intervals



- Example: an *audio stream* in CD quality w/ sound wave at 44,100Hz
  - Destination starts reading before the entire information has been transmitted
  - Each piece of data should be read in order at fixed rate: every 1/44,100 seconds.

Stream-oriented communication

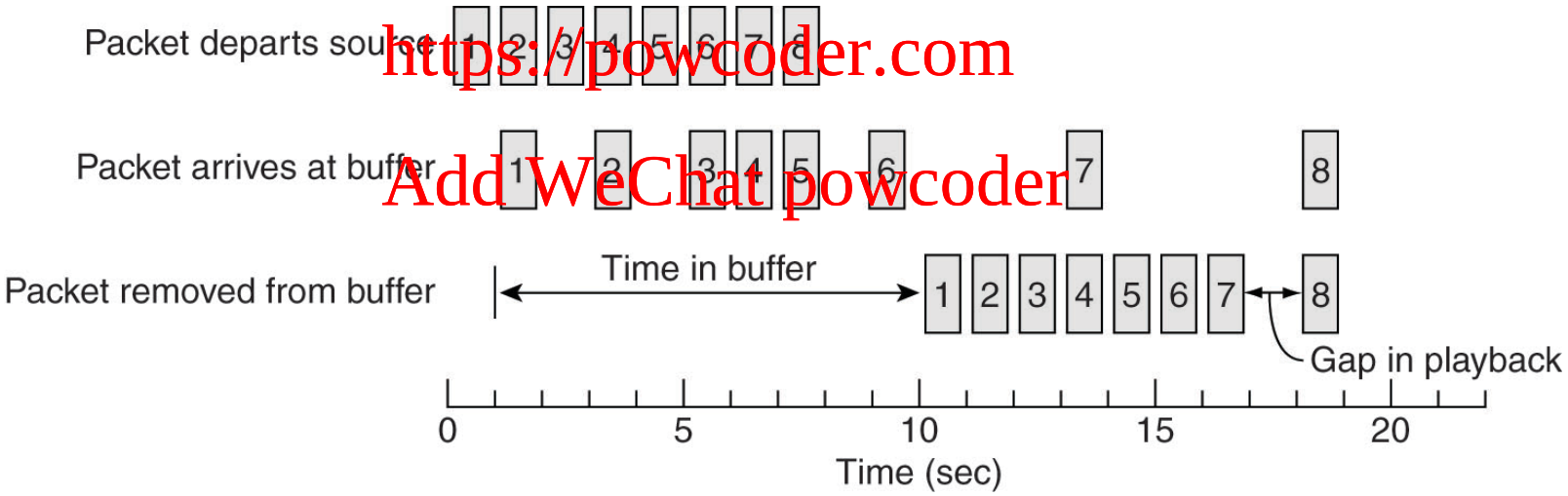
Quality of Service (QoS)

- Requirements to ensure that the temporal relationships in a stream can be preserved:
  - The required bit rate at which data should be transported
  - The maximum delay until a session has been set up (i.e., when an application can start sending data)
  - The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient)
  - The maximum delay variance, or jitter
  - The maximum round-trip delay
- Problem: how to stream over internet?
  - Internet Protocol (IP) is best effort, it drops packet
  - IP rarely implements QoS

Stream-oriented communication

Buffer to cope with variable delays

- Use a buffer to store several data in advance at the receiver
- If packets are delayed with a certain variance but the average rate is sustainable
- The receiver can pass packets to the application at regular time intervals

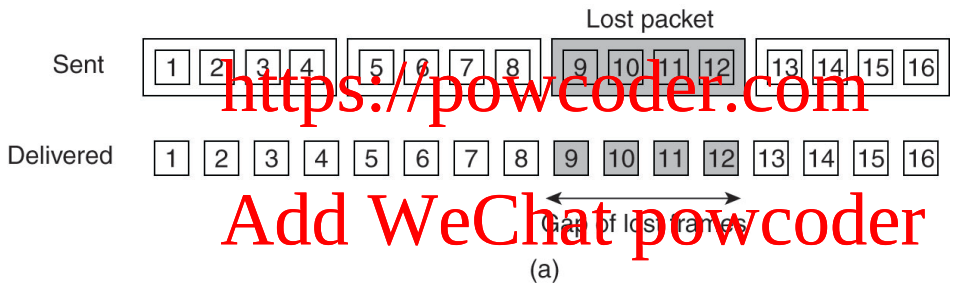


- The size of the receiver buffer is 9 seconds of packets to pass, unfortunately packet #8 took 11 seconds to reach the receiver at which time the buffer was empty.

Stream-oriented communication

Error correction and interleaving to cope with message losses

- With best effort protocols, packets can be dropped
- TCP/IP retransmit drops packets, yet this is too heavy for streaming application
- Forward error correction: k out of n packets are enough to reconstruct k packets
- Interleaving circumvents dropped packets for certain multiple consecutive audio and video frame



- The effect of packet loss in (a) non-interleaved transmission and (b) interleaved transmission

Streaming multimedia: DASH

- *DASH: D*ynamic, *A*daptive *S*treaming over *H*TTP
- *server:*
  - divides video file into multiple chunks
  - each chunk stored, encoded at different rates
  - *manifest file:* provides URLs for different chunks
- *client:*
  - periodically measures server-to-client bandwidth
  - consulting manifest, requests one chunk at a time
    - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)

Streaming multimedia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *"intelligence" at client:* client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is "close" to client or has high available bandwidth)

Video Streaming and CDNs: context

- video traffic: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users
- challenge: scale - how to reach ~1B users?
  - single mega-video server won't work (why?)
- challenge: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)

Content distribution networks

- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of *simultaneous* users?

- *option 1:* single, large "mega-server"
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

….quite simply: this solution *doesn't scale*

Content distribution networks

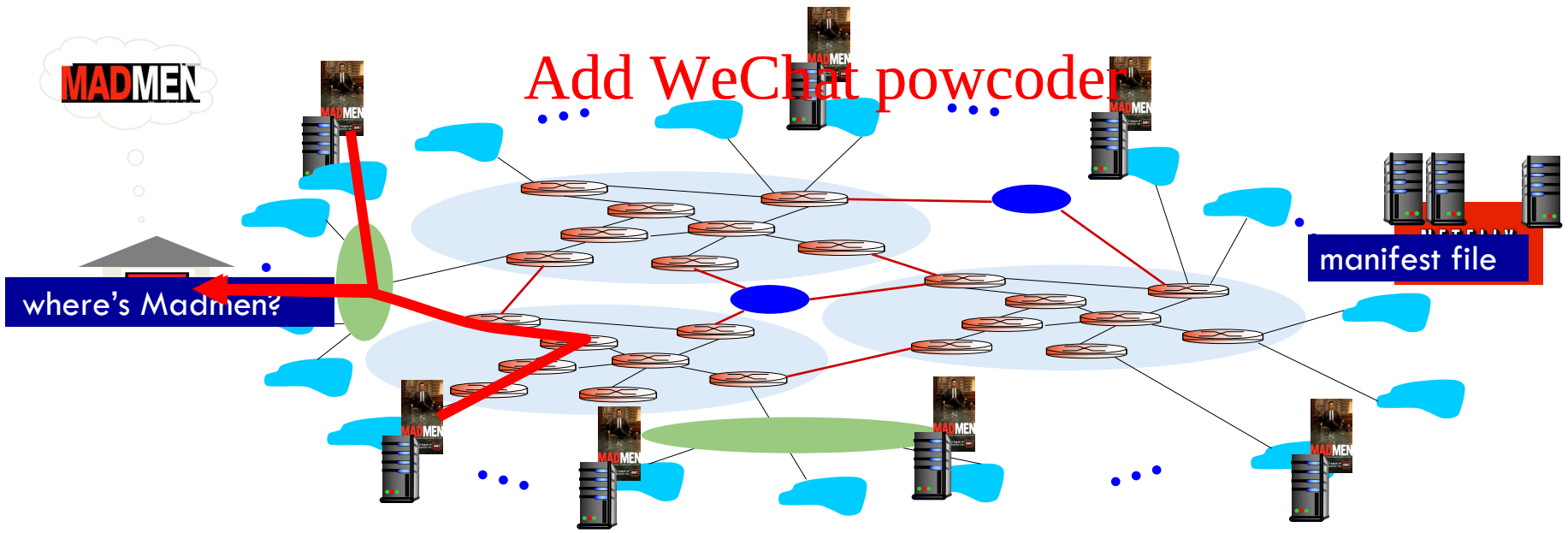
- *challenge:* how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- *option 2:* store/serve multiple copies of videos at multiple geographically distributed sites *(CDN)*
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10's) of larger clusters in POPs near (but not within) access networks
    - used by Limelight

- *solution:* distributed, application-level infrastructure

Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested



Conclusion

- Network protocols are divided into layers
- Routing protocols are
  - simple but not scalable (distance-vector) or
  - more complex but scalable (link-state)
- There are various ways of communicating depending on the needs:
  - We covered: General-purpose communication (Socket)
- Streaming-oriented Communication

What's Next ?

- **Tutorial on Wednesday.**

- Read Chapter 4 of the textbook

- Next week: more about communications in distributed systems

- See you all next week !