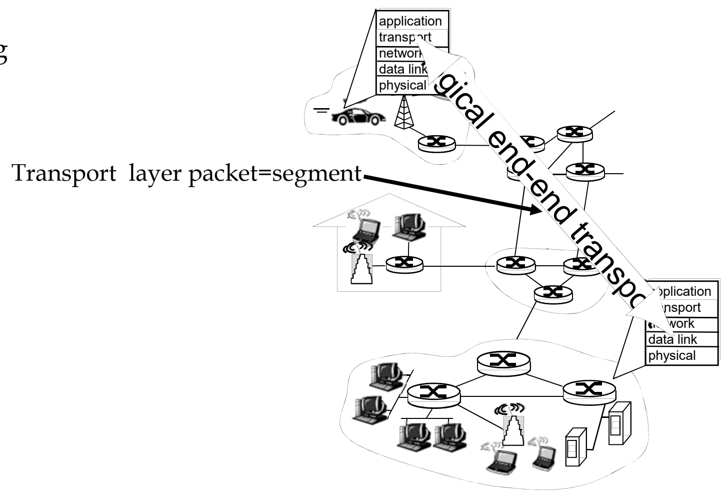
Transport Protocols and Network Control #8

Outline

- Goals:
  - Understand principles behind transport layer services
  - Multiplexing/demultiplexing (Ports/Sockets)
- Examples of Transport Protocols
  - UDP
  - TCP
  - Note there are other transport layer protocols
- Network control
  - Active Queue Management
  - MPLS
  - SDN

Transport services and protocols

- Provide *logical communication* between app processes running on different hosts
- 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, RTP, UDP and others



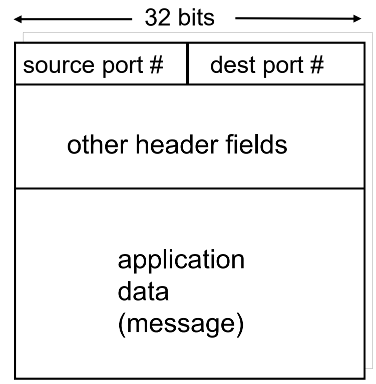
Modified from: Computer Networking: A Top Down Approach
4th edition. Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Sockets

- A socket is programming interface that allows communication between applications running on different computers or devices.
- A socket is identified by an IP address and a port number (which identifies a specific application or service running on that device). The combination of an IP address and port number is used to establish a unique endpoint for communication.

How multiplexing works

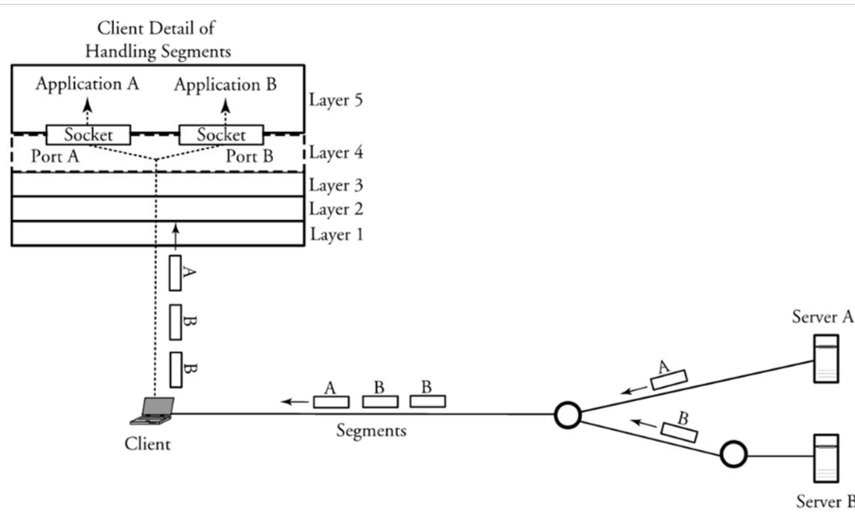
- IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has a:
    - source port number
    - destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

From: *Computer Networking: A Top Down Approach*
4th edition. Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Ports & Sockets: Multiplexing



Modified from: *Computer and Communication* Networks.
Nader F. Mir Prentice Hall.

Ports

- Port address
- Ports are 16 bits
- The port address is internal to the host (indicates application)
- A socket address is unique in the Internet
- Once an application creates a socket and TCP connection then a ***write*** is used to send to the network and a ***read*** used to receive from the network.

Ports

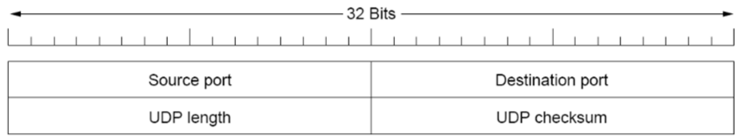
- Ports are 16 bits
- Well Known Ports are those from 0 through 1023.
- The Registered Ports are those from 1024 through 49151
- The Dynamic and/or Private Ports are those from 49152 through 65535

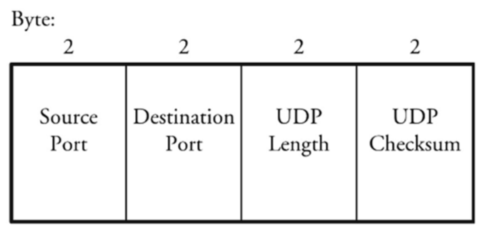
- There are some common port numbers
  - Example:
    - File data transfer (21)
    - TELNET (23)
    - Simple Mail Transfer Protocol (SMTP)(25)
    - Remote Procedure Call [RPC] (111)
    - Web servers listens on port 80

http://www.iana.org/assignments/port-numbers

Transport Layer: UDP

- UDP
  - Connectionless
  - No congestion control
  - No acknowledgments
  - Packets may be
    - lost
    - delivered out of order to app
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others
  - UDP checksum covers header and data → optional, but commonly used





Ports and UDP

- UDP socket, must specify
  - destination IP address
  - destination port #
- When receiving host receives *UDP* segment:
  - Checks destination port # in segment
  - Directs UDP segment to socket with that port #
  - IP/UDP datagrams with *same dest. port* #, but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

UDP Use Cases

- Streaming multimedia apps
  - loss tolerant
  - rate sensitive
- DNS
- Simple Network Management Protocol(SNMP)
- Reliable transfer over UDP: add reliability at application layer → application-specific error recovery!

UDP

- "no frills" protocol:
  - segments may be lost, delivered out of order
  - best effort service: "send and hope for the best"
- UDP has its plusses:
  - no setup/handshaking needed (no RTT incurred)
  - can function when network service is compromised
  - checksum on covers the header and data
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)

Transport Layer: TCP

- TCP provides for assured delivery of PDU's
- TCP Services
  - Connection oriented (end-to-end)
    - Need call processing (in end points) (not inside the network)
    - Information on the status of each connection is available
  - Reliable data transfer
    - Uses acknowledgments
    - Uses sequence numbers

TCP: overview RFCs: 793,1122, 2018, 5681, 7323

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Connection-oriented demultiplexing

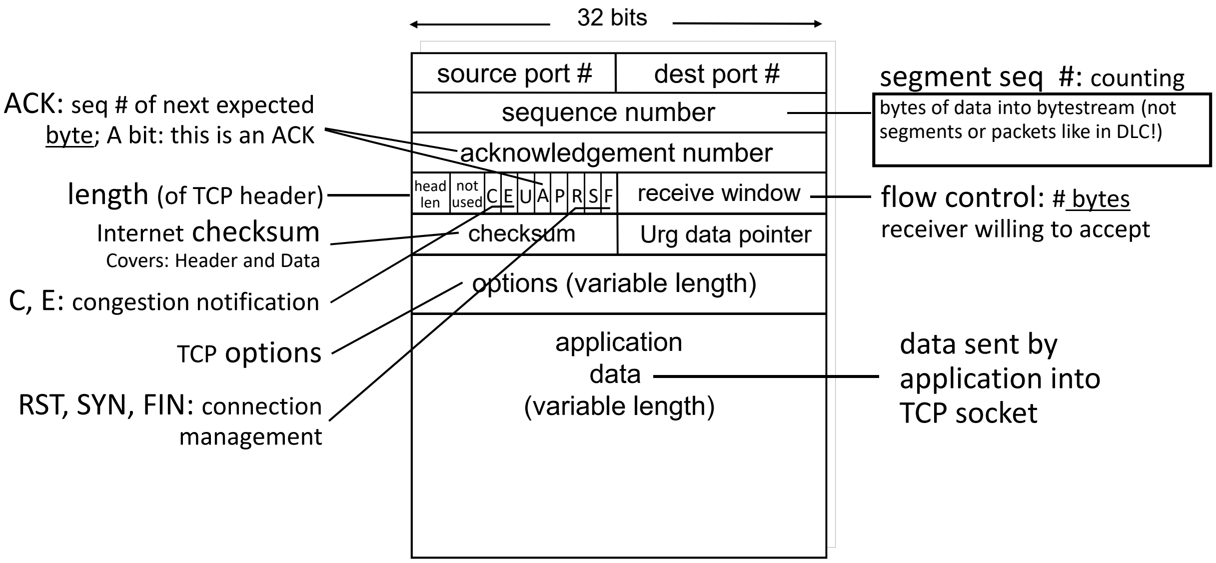
- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Connection management

- Connection management allocates, synchronizes, and deallocates states while allowing the communicating parties to negotiate their operation modes and resources needed for their association.

TCP header



Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

TCP sequence numbers, ACKs

Sequence numbers:

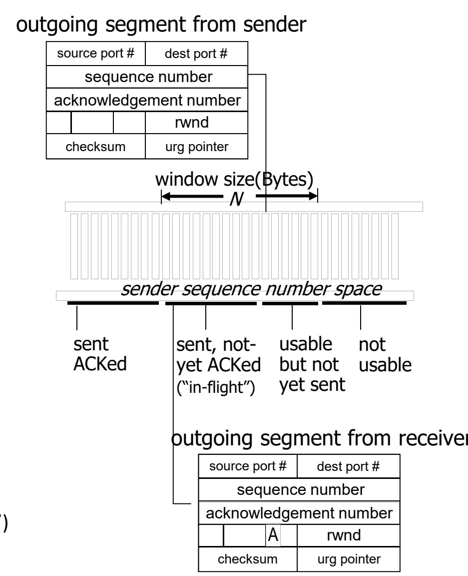
- byte stream "number" of first byte in segment's data

Acknowledgements:

- 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 (common to use "Selective Repeat")



Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

TCP Header

- Source/Destination identify local end points
- Window size (in Bytes) used to dynamically control source rate into the network
- Checksum, checks the header and data

TCP

- Stream-oriented
  - TCP collect user bytes and forms segments to be passed on to the IP layer
  - Sequence number based on byte counts
- Push
  - Upper layer protocol send Push message to TCP to force it to send all the bytes collected in a segment
- Resequencing
  - IP may deliver information out of order, TCP must put it back together

TCP Header

- Urgent data pointer is used when a sender wants to send data that requires immediate attention at the receiver's end, it can mark that data as "urgent."
- The urgent data could be used, for example, to send time-critical control messages or notifications.
- The urgent data mechanism in TCP is optional and not commonly used.

From: "Computer Networks, 3rd Edition, A.S. Tanenbaum. Prentice Hall, 1996

Modified from: *Computer and Communication* Networks. Nader F. Mir Prentice Hall.

TCP

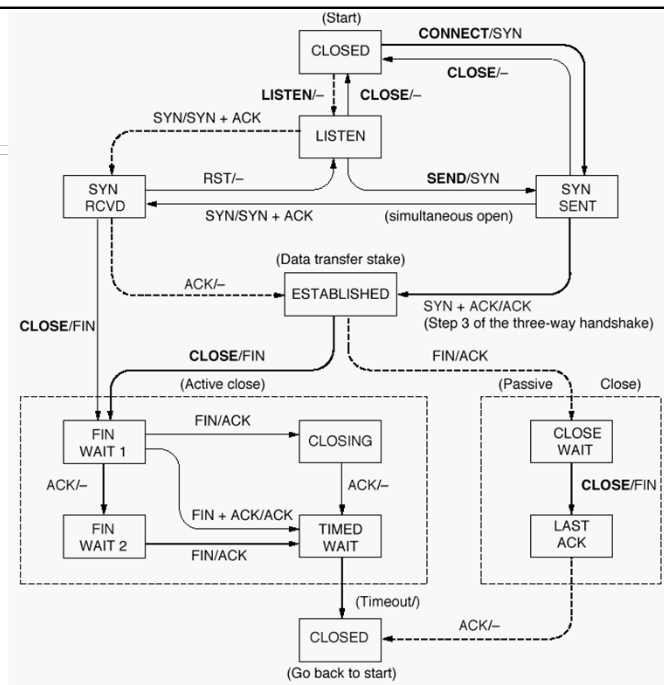
- Inclusive Acknowledgment
  - Acknowledgment number, acknowledges all received bytes prior to the one specified
- Flow control
  - Window size is in bytes
  - Transmit N-bytes and the must wait for acknowledgment
  - Window size is dynamic, i.e., it changes based on "knowledge" of availability of buffer space in receiver and network congestion

TCP

- Multiplexing
  - Allows multiple sessions within one host to be transmitted over an IP path (ports/sockets)
- Full duplex
- Graceful close
  - All traffic in flow is acknowledged before the session is ended.

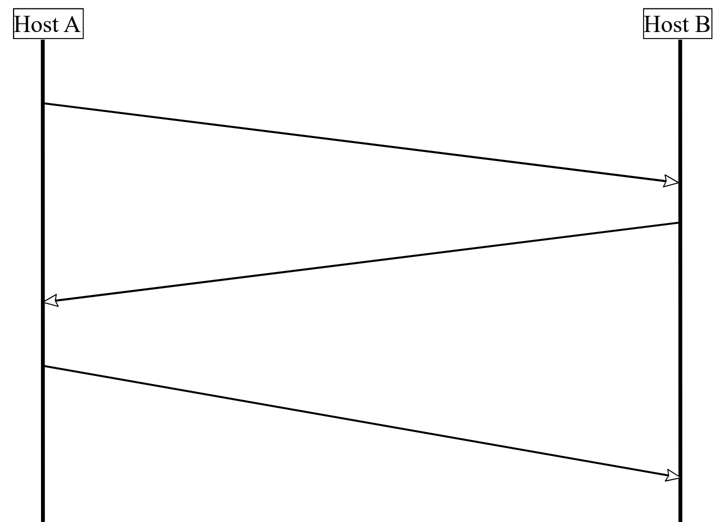
TCP Session Processing

TCP Connection Management: Finite State Machine



From: "Computer Networks, 3rd Edition, A.S. Tanenbaum. Prentice Hall, 1996

TCP Connection Setup: Three-way Handshake



From: Communications Networks, Garcia and Widaja, McGraw Hill, 2000

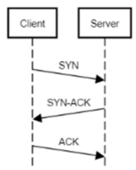
tcpdump
http connection set up

- Output columns are → Time SourceIP.SourcePort > DestIP.DestPort Flags ...
- 11:13:38.524046 x.x.x.x.3600 > 64.233.167.104.80: S 2021815674:2021815674(0) win 64240 <mss 1460,nop,nop,sackOK> (DF)
  - First packet is from client host x.x.x.x.
  - Client host is using 3600 as a source port.
  - Destination host is 64.233.167.104 on port 80 (that's Google's webserver).
  - The packet with flag S is a TCP SYN packet, means in words "i'd like to open a TCP connection with you"
  - Client host will have a temporarily opened port (3600) in order to receive data back from the server.
- 11:13:38.558668 64.233.167.104.80 > x.x.x.x.3600: S 3132749891:3132749891(0) ack 2021815675 win 8190 <mss 1460>
  - Second packet is sent from Google webserver. This packet comes from 64.233.167.104 source port 80, and contains SYN/ACK
  - TCP flags sent to client port 3600, means "ok you may open a connection with me".
- 11:13:38.559105 x.x.x.x.3600 > 64.233.167.104.80: . ack 1 win 64240 (DF)
  - Third packet is the client host sending a last ACK packet, which means "ok we are now connected". Source and dest ports must stay the same here.

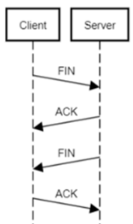
Example

Session Establishment

3-way handshake



Tear down



Wireshark Trace-Session Establishment (Set-up)

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.16.128	212.58.226.142	TCP	66	2826 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
2	0.132627	212.58.226.142	172.16.16.128	TCP	66	80 → 2826 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1406 SACK_PERM=1 WS=128
3	0.132768	172.16.16.128	212.58.226.142	TCP	54	2826 → 80 [ACK] Seq=1 Ack=1 Win=16872 Len=0

Wireshark Trace-Session Close (Tear-down)

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	67.228.110.120	172.16.16.128	TCP	60	80 → 3363 [FIN, ACK] Seq=1 Ack=1 Win=71 Len=0
2	0.000056	172.16.16.128	67.228.110.120	TCP	54	3363 → 80 [ACK] Seq=1 Ack=2 Win=4218 Len=0
3	0.460318	172.16.16.128	67.228.110.120	TCP	54	3363 → 80 [FIN, ACK] Seq=1 Ack=2 Win=4218 Len=0
4	0.541853	67.228.110.120	172.16.16.128	TCP	60	80 → 3363 [ACK] Seq=2 Ack=2 Win=71 Len=0

There are other Session Close (or Connection Termination) sequences

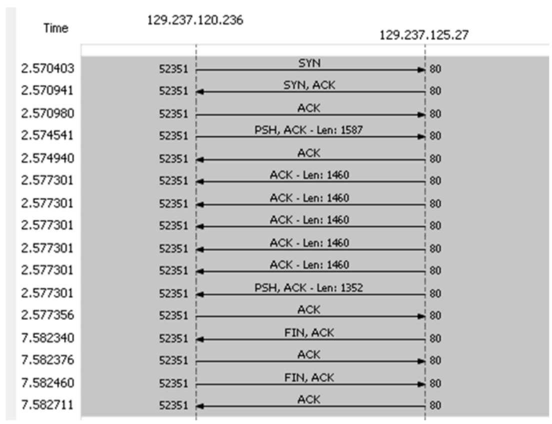
Modified from: Basic TCP analysis with Wireshark, by Waleed Tageldeen: https://codeburst.io/basic-tcp-analysis-with-wireshark-b99ed54fa499

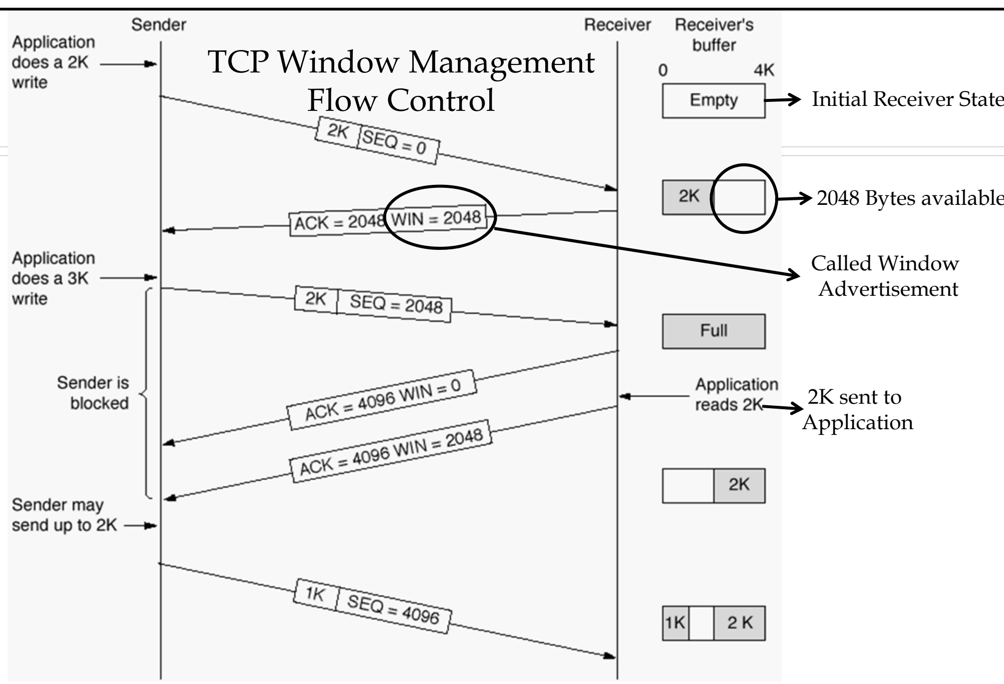
Wireshark traces: Open/Close

- From Basic TCP analysis with Wireshark, by Waleed Tageldeen: https://codeburst.io/basic-tcp-analysis-with-wireshark-b99ed54fa499
- Connection Establishment:
  - https://github.com/chrissanders/packets/blob/master/tcp_handshake.pcapng
- Connection Close
  - https://github.com/chrissanders/packets/blob/master/tcp_teardown.pcapng

Example:
A http TCP Session: Start-to-Finish

Wireshark Trace





From: "Computer Networks, 3rd Edition, A.S. Tanenbaum. Prentice Hall, 1996

TCP Animation: TCP Window Management

- http://www.ccs-labs.org/teaching/rn/animations/flow/

Silly Window Syndrome

- Situation:
  - Transmitter sends large amount of data
  - Receiver buffer depleted slowly, so buffer fills
  - Every time a few bytes read from buffer, a new advertisement to transmitter is generated
  - Sender immediately sends data & fills buffer
  - Many small, inefficient segments are transmitted
- Solution:
  - Receiver does not advertize window until window is at least ½ of receiver buffer or maximum segment size
  - Transmitter refrains from sending small segments

From: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

Delay-BW Product & Advertised Window Size

- Suppose RTT=100 ms, R=2.4 Gbps
  - # bits in pipe = 30 Mbytes
- If single TCP process occupies pipe, then required advertised window size is
  - RTT x Bit rate = 30 Mbytes
  - Normal maximum window size is 65535 bytes
  - With normal max window efficiency ~ 0.2%
- Solution: Window Scale Option
  - Window size up to 65535 x 2^{14} = 1 Gbyte allowed
  - Requested in SYN segment
  - Uses options Fields

From: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

TCP: Retransmission Procedures

- TCP uses a positive acknowledgment
- Selecting timeout timer value
  - Delay unknown a-priori
  - Segments may be lost making measurements of the round-trip time (RTT) difficult, i.e., measurement of RTT can have a large variance

TCP round trip time, timeout

Q: how to set TCP timeout value?

- longer than RTT, but RTT varies!
- *too short:* premature timeout, unnecessary retransmissions
- *too long:* slow reaction to segment loss

Q: how to estimate RTT?

- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current `SampleRTT`

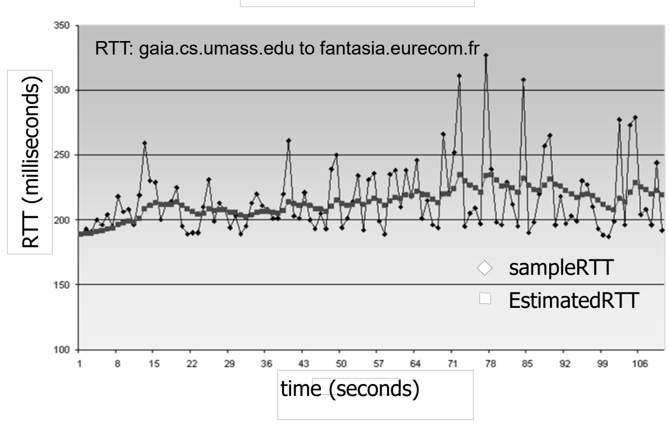
Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

TCP round trip time, timeout

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- exponential weighted moving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: α = 0.125

Note: first order infinite impulse response (recursive) filter $y[n]=(1-\alpha)y[n-1]+\alpha x[n]$



Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

TCP round trip time, timeout

- timeout interval: **`EstimatedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT:`** want a larger safety margin

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT "safety margin"

- **`DevRTT`**: EWMA of **`SampleRTT`** deviation from **`EstimatedRTT`**:

```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
```

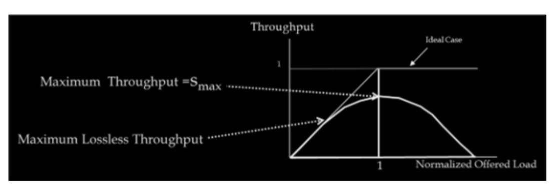
(typically, β = 0.25)

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Principles of congestion control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!



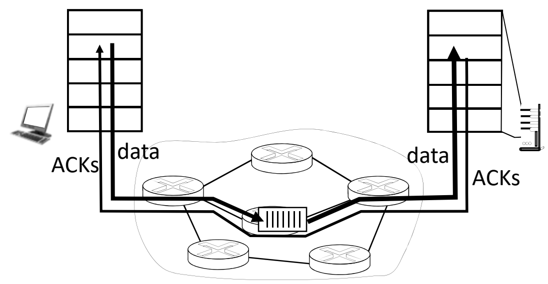
flow control (local):
one sender too fast for one receiver

congestion control (global):
too many senders, sending too fast for network

Approaches towards congestion control

End-end congestion control:

- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



TCP: Adaptive Congestion Control

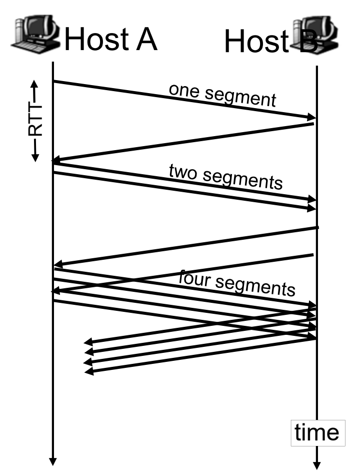
- If time out TCP **assumes** congestion caused loss
- If the network is congested then want to slow the source down to reduce congestion
- When the network congestion disappears then want to allow the source to send faster

TCP: Adaptive Congestion Control

- Turn efficiency calculation of data link control algorithms around
- Use window size to control the flow of traffic into the network

TCP: Adaptive Congestion Control

- Increase algorithm
  - If acknowledgement received then increase the window size by one segment, i.e.,
  - new_window = old_window + 1 segment
  - This is called the slow start phase
  - Initial rate is slow, but ramps up exponentially fast



Modified from: Computer Networking: A Top Down Approach
4th edition. Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

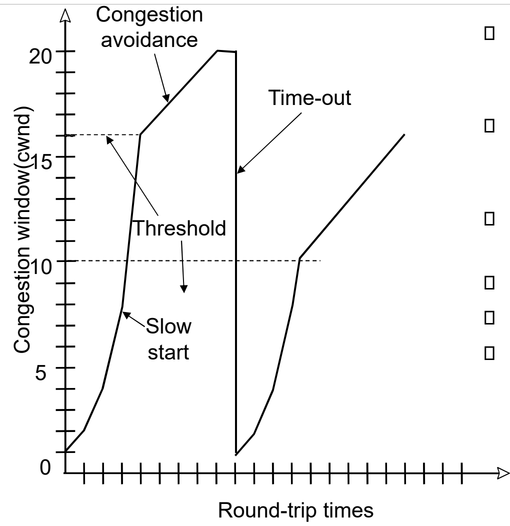
TCP: Adaptive Congestion Control

- If every packet is acknowledged in slow start then the window (and rate) doubles every RTT, Exponential increase.
- After the window reaches a threshold, it enters the congestion avoidance phase.
- In the congestion avoidance phase, upon receipt of an Ack it is increased by 1 segment every RTT, Linear increase

TCP: Adaptive Congestion Control

- Decrease Algorithm
  - If loss then set
    - – new_threshold = (1/2)current window
    - – Redo Slow Start from CWND = 1 Segment
- Congestion Window (CWND): CWND is a parameter that dynamically adjusts the amount of unacknowledged data a sender can have in flight at any given time. It acts as a throttle to prevent sending data faster than the network can handle.
- This is a distributed, asynchronous algorithm – has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties

TCP Congestion Control: Congestion



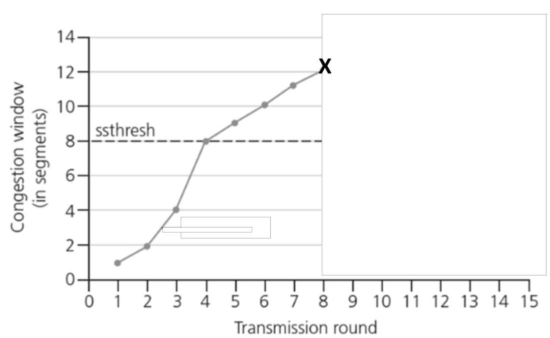
- Congestion is detected upon timeout or receipt of duplicate ACKs
- Assume current cwnd corresponds to available bottleneck link capacity
- Adjust congestion threshold = ½ x current cwnd
- Reset cwnd to 1 (TCP Tahoe)
- Go back to slow-start
- Over several cycles expect to converge to congestion threshold equal to about ½ the available bottleneck link capacity

Modified from: Communication Networks: Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.



Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Variation of TCP Algorithms: Intertwined algorithms used commonly in TCP implementations

- TCP can use Go-Back-N or Selective Acknowledgements (SACK); SACK is most common
- **Slow Start** - Every *ack* increases the sender's window (*cwnd)* size by 1
- **Congestion Avoidance** - Reducing sender's window size by half at experience of loss, and increase the sender's window at the rate of about *one packet per RTT (NOTE: not per ack)*
- **Fast Retransmit** - Don't wait for retransmit timer to go off, loss event is when *3 duplicate acks* received
- **Fast Recovery** - Since duplicate *ack* came through, one packet *has left the wire*. Perform *congestion avoidance*, don't jump down to *slow start*

Modified from : Paul D. Amer, University of Delaware
*www.cis.udel.edu/~amer/856/**tcpvariations**-Amer.ppt*

TCP Animation:
TCP Congestion Control

- https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/tcp-congestion/index.html

Approximate TCP (Reno) performance

$$R_{TCP} \approx \text{Average throughput of a connection (bytes/sec)} = \frac{1.22 * MSS(bytes)}{RTT(\text{Sec}) * \sqrt{P_L}}$$

MSS=Maximum Segment Size=largest amount of data can be received in a single TCP segment
P_L=packet loss probability

With P_L = (#bits in segment)*BER
$P_L = 10^{-6}$ and 1500B segment
BER= ~$8*10^{-10}$

Example 1: MSS = 1500 bytes, RTT=10ms, segment loss rate = 10^{-6}
R_{TCP}=1.8x 10^8 Bytes/sec = 1.4 Gb/s

Example 2: Large DBP network: MSS = 1500 bytes, RTT=100ms, segment loss rate = 10^{-10}
R_{TCP}=1.8 10^9 Bytes/sec = 14.64 Gb/s

Example 3: Large DBP network: MSS = 1500 bytes, RTT=100ms, BER= 10^{-7} & segment loss rate = $1.2*10^{-3}$
R_{TCP}=520kBytes/sec = ~4.64 Mb/s

To achieve very high throughputs requires a very small segment loss probability, spurring on development of new TCPs for high speed environment

See Average Throughput of TCP Connection for TCP Reno

Approximate TCP performance

- For losses due to transmission (bit) errors an approach to reducing the packet loss rate at the transport layer
  - Recover errored packets at the link layer
    - Error control at the link layer increases the delay
    - Error control at the link layer "hides" loss from the transport layer
    - Trade-off delay for loss
    - Cannot hide all losses

Flavors of TCP

- TCP is end-to-end so many variations can co-exist in the Internet.
  - TCP-Tahoe
  - TCP-Reno (most commonly deployed variant)
  - TCP-Vegas
  - TCP-NewReno
  - Fast TCP (FastTCP)
  - BIC TCP (Binary Increase Congestion control)
  - CUBIC TCP
  - HighSpeed TCP (HSTCP)
  - Compound TCP (CTCP)
    - Microsoft algorithm that was introduced as part of the Windows Vista and Window Server 2008 TCP stack.

Congestion Control

- Global Issue
- Demand for network resources must be controlled.

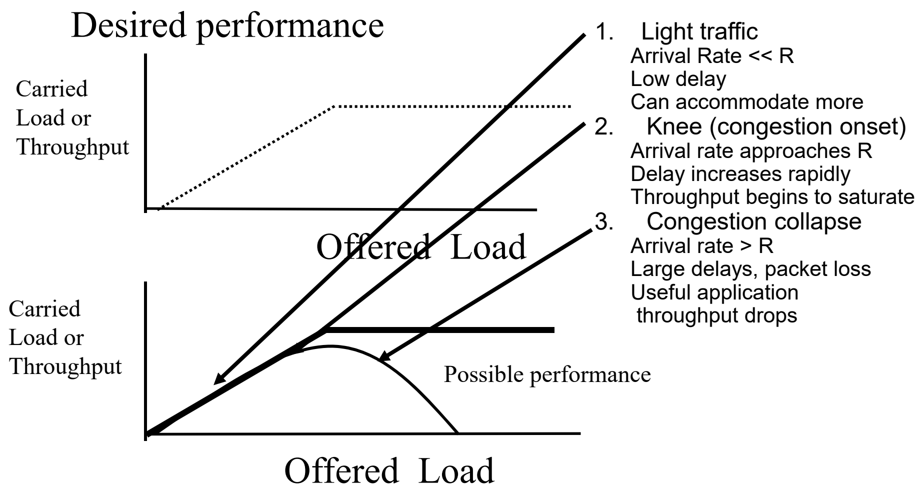
Traffic and network engineering

- Network Engineering: Network engineering involves the design, implementation, and maintenance of computer networks. It focuses on the overall architecture, topology, and infrastructure of a network, including hardware, protocols, and connectivity. Network engineering involves network planning, device configuration, network optimization, security, and scalability.
- Network engineering involves long-term planning and design activities that are typically performed during the initial network design, deployment, or major upgrades. Network engineering builda foundation for the network infrastructure, considering factors like scalability, redundancy, and future growth.
- The goal of Network Engineering is to design and maintain a robust and scalable network infrastructure that meets the organization's requirements.

Traffic and network engineering

- Traffic Engineering is a subset of network engineering that specifically deals with the management and optimization of network traffic flows. It focuses on controlling and directing network traffic to improve performance, efficiency, and resource utilization. Traffic engineering involves modifying traffic patterns, implementings traffic management techniques, optimizings routing protocols.
- Traffic engineering mechanisms are implemented at the packet level and are dynamic and responsive, addressing real-time or near-real-time conditions within the network.
- The goal of traffic engineering is to **prevent congestion** and optimize the flow of network traffic to achieve specific performance objectives.
- TCP uses Adaptive Congestion Control
- Other mechanisms will be discusses next.

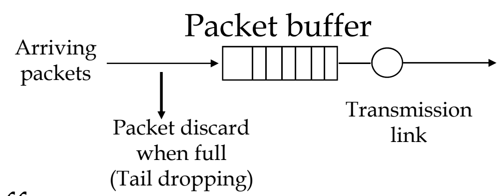
Congestion Control-Objective



Congestion Control

- Preventative
  - Call Admission Control (CAC)
  - VC switching
- Reactive
  - Packet Dropping
  - TCP is reactive→ End-to-End

Queue Management: FIFO Queueing



- All packet flows share the same buffer
- Transmission Discipline: First-In, First-Out
- Buffering Discipline: Discard arriving packets if buffer is full
- Called → Tail dropping
- Alternatives:
  - Random discard;
  - Pushout head-of-line, i.e. oldest, packet

FIFO Queueing

- Cannot provide differential CoS to different packet flows
  - Different packet flows interact strongly
- Difficult to determine performance delivered
- Finite buffer determines a maximum possible delay
- Buffer size determines loss probability
  - But depends on arrival & packet length statistics
- Variation: packet enqueueing based on queue thresholds
  - some packet flows encounter blocking before others
  - higher loss, lower delay

Bufferbloat

- Buffers are good, they are needed to queue packets.
- Too much of a good thing can be bad.
- Very large high speed buffers are now economically feasible.
- TCP has a congestion control function,
  - TCP packets from the source to the destination can be excessively delayed in a large buffer at congested (bottleneck) interface.
  - Then TCP then does not "learn" about the congestion in time and continues to transmit at the same rate.
- TCP Acks can be delayed by large buffers in the reverse path,
  - The source rate maybe reduced for lack of an ACK.
  - If delay is too long, TCP may see that as a loss.
  - But TCP congestion control, i.e., slowing down, does not help relieve congestion in the reverse path.

Congestion Management: Possible solutions (AQM)

- Routers set the Explicit Congestion Notification (E) bit in the TCP header
- Random Early Detection (RED) – more later
- ColDel (Controlled Delay)
  - Packet arrives at buffer, timer started for this packet.
  - When packet timer exceeds threshold, the packet is dropped. (The time in the buffer is called sojourn time)
  - A dropped packet tells TCP to slow down, mitigating congestion at the bottlenecked link.

Congestion Management: Possible solutions (AQM)

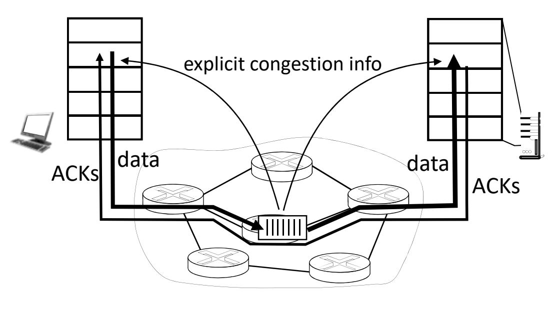
- Proportional Integral controller Enhanced (PIE)
  - When a packet arrives, the packet maybe droped.
  - The drop probability is updated periodically based on how far the current latency is away from the target value and whether the queuing latency is currently trending up or down.
  - Implemented in DOCSIS 3.1

Modified from: Proportional Integral Controller Enhanced (PIE):
A Lightweight Control Scheme to Address the Bufferbloat Problem, Internet Engineering Task Force, RFC 8033

Approaches towards congestion control

Network-assisted congestion control:

- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate

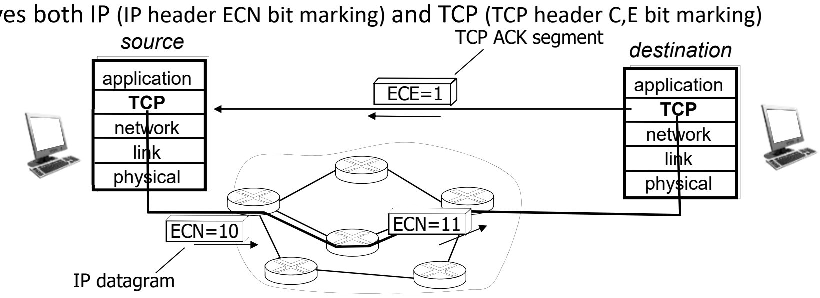


Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Explicit congestion notification (ECN)

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets E bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



- TCP ECN, ATM, DECbit protocols→ not commonly deployed

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Active Queue Management:
Random Early Detection

- A Congestion Control Method for the Internet
- Implemented in routers
- Random Early Detection (RED)
  - RED is an example of Active Queue Management (AQM)
  - Monitor average **ROUTER** queue length
  - If average **ROUTER** queue length > threshold then Drop arriving packet with some probability p, (p=drop probability)
- This *implicitly* notifies the TCP source that there is congestion and the source then backs off
- In the Internet *"Random Early Detection"* (RED) gateways use this basic concept with some added complexity

Early or Overloaded Drop

Random early detection:

- Drop packets if short-term average of queue length exceeds threshold
- Packet drop probability increases linearly with queue length
- Option to just mark offending packets (DE)
- Improves performance of cooperating TCP sources
- Increases loss probability of misbehaving sources

Random Early Detection (RED)

- Packets produced by TCP will reduce input rate in response to network congestion
- Early drop: discard packets before buffers are full
- Random drop causes some sources to reduce rate before others, causing gradual reduction in aggregate input rate

Packet Drop Profile in RED

Algorithm:

- Maintain running average of queue length, Q_{avg}
- If Q_{avg} < minthreshold, do nothing
- If Q_{avg} > maxthreshold, drop packet
- If in between, drop packet according to probability
- Flows that send more packets are more likely to have packets dropped

Q_{avg}

Traffic Engineering – Directing Traffic

- MultiProtocol Label Switching (MPLS)
- Software-Defined Networking (SDN)

MPLS Why?

- Provide a form a virtual circuit switching in the Internet for aggregates of flows not for individual hosts
- Label switching enables routing flexibility
- Virtual circuit switching enables QoS on aggregates of flows
- Enables traffic engineering
  - Moving the traffic to where the bandwidth is
  - Establish separate paths to meet different performance requirements of aggregated traffic flows
  - Uses explicit routes for better load balancing.

MPLS: Why?

- Improve IP forwarding performance - faster look-up using a fixed length identifier
- Decouple routing and forwarding components of IP
  - Routing - to build and maintain forwarding tables
  - Forwarding - directs packet from input interface to output interface, based on forwarding table look-up
  - MPLS can use different routing protocols for flow aggerates.
- Keeps IP addressing

MPLS: Why?

- Circuits are good (sometimes)
  - Conventional IP routing selects one path, does not provide choice of route
  - Label switching enables routing flexibility
  - Survivability
- Virtual Private Networks: establish tunnels between user nodes

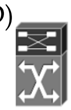
MPLS: Why

- MPLS provides a tunneling mechanism to interconnect VPN sites
- MPLS can be generalized to provide
  - Control plane for optical cross-connects
  - Automatic protection switching, without SONET overheads
  - Generalized MPLS (GMPLS)
    - Time Slot→ Label
    - Wavelength → Label
    - MPLS (IP) → Label
    - All can use the same infrastructure

MPLS concepts- How?

- Just like Virtual Circuit Switching (but with different terms)
- Forwarding Equivalence Class **(FEC)** - group (***Aggregate***) of IP packets (range of IP addresses) that are forwarded in the same manner
- Label - assigned per FEC (like Virtual circuit ID)
- Label Switch Router **(LSR)** –

  Here a routers acts *Like a VC switch*

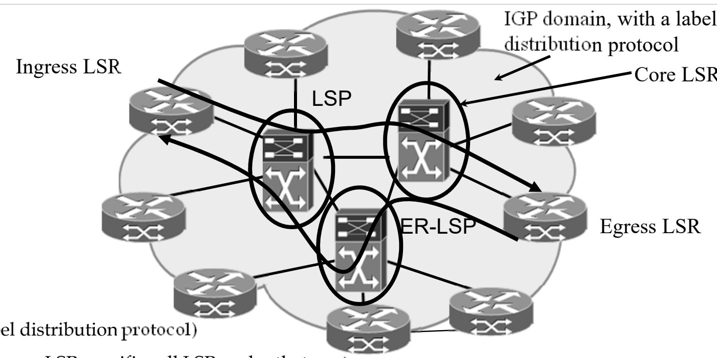


- Edge (Ingress/Egress) LSRs assign/remove labels, can perform packet classification
- Core LSRs switch packets based on label value
- Existing IP routing protocols used to exchange routing info
- All LSRs use some kind of label distribution protocol **(LDP)**

  a ***signaling protocol***
- Label Switched Path **(LSP)** - sequence of LSRs through which labeled packets go through to reach the egress LSR

Section 10.3

MPLS Concepts-How?

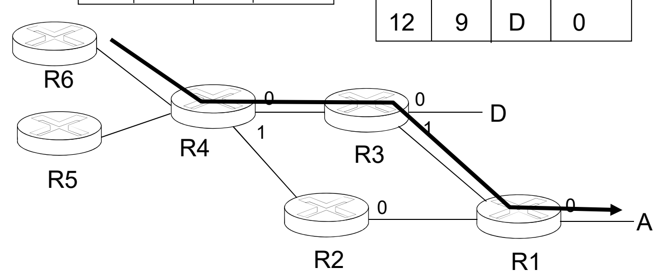


- LSP's are unidirectional
- Route selection can use:
  - Hop-by-hop routing (using IGP and a label distribution protocol)
  - Explicit routing (ER) - ingress LSR specifies all LSR nodes that are in the path: statically (like source routing), or using link-state topology information)
    - May be signaled using RSVP-TE or CR-LDP
    - May be different from IGP-shortest path
    - Explicit routing useful for traffic engineering

MPLS forwarding tables

in label	out label	dest	out interface
	10	A	0
	12	D	0
	8	A	1

in label	out label	dest	out interface
10	6	A	1
12	9	D	0

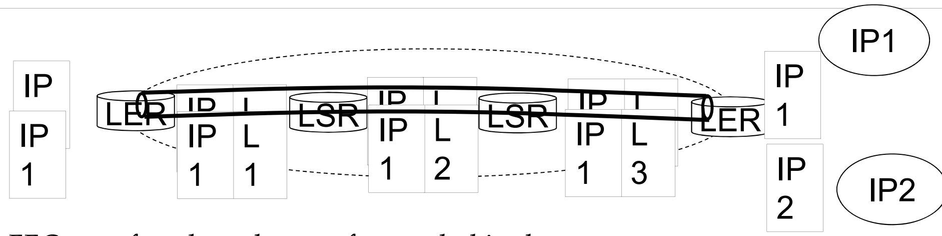


in label	out label	dest	out interface
8	6	A	0

in label	out label	dest	out interface
6	-	A	0

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Forwarding Equivalence Class (FEC)



- *FEC:* set of packets that are forwarded in the same manner
  - Over the same path, with the same forwarding treatment
  - Packets in an FEC have same next-hop router
  - Packets in same FEC may have different network layer header
  - Each FEC requires a *single entry* in the forwarding table
    - Coarse Granularity FEC: packets for all networks whose destination address matches a given address prefix
    - Fine Granularity FEC: packets that belong to a particular application running between a pair of computers

Modified from: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

MPLS Concepts- How?: Packet Header

PPP or Ethernet header	**MPLS header**	IP header	Remainder of Packet

label	Exp	S	TTL
20	3	1	5 -- Bits

Label
- Value to determine next hop of the packet

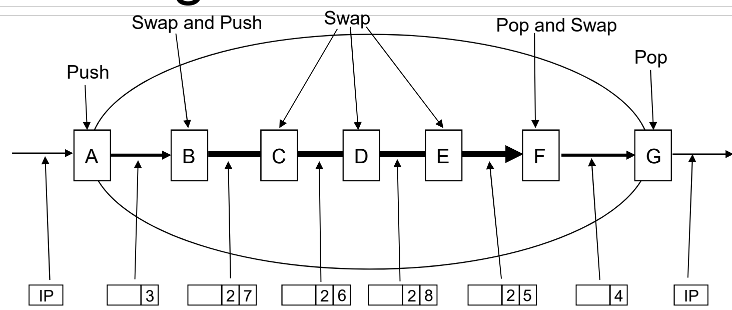
Experimental (EXP)
- Used as CoS field - Limited QoS parameters, derived from IP header, diffserv, etc.

Bottom of Stack (S)
- Set to 1 if bottom of label stack, otherwise 0

Time to Live (TTL)
- Eliminates loops and prevents packets from remaining in the network indefinitely

Modified from: Computer Networking: A Top Down Approach
4th edition. Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Label Stacking

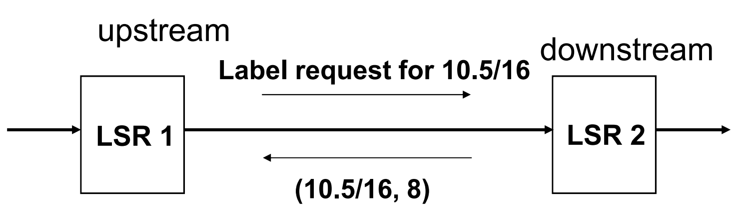


- MPLS allows multiple labels to be stacked
  - Ingress LSR performs *label push* (S=1 in label)
  - Egress LSR performs *label pop*
  - Intermediate LSRs can perform additional pushes & pops (S=0 in label) to create tunnels
  - Above figure has tunnel between A & G; tunnel between B&F
  - All flows in a tunnel share the same outer MPLS label

Modified from: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

Label Distribution (Example)

- Label Distribution Protocols distribute label bindings between LSRs



Downstream-on-Demand Mode

- LSR1 becomes aware LSR2 is next-hop in an FEC
- LSR1 requests a label from LSR2 for given FEC
- LSR2 checks that it has next-hop for FEC, responds with label

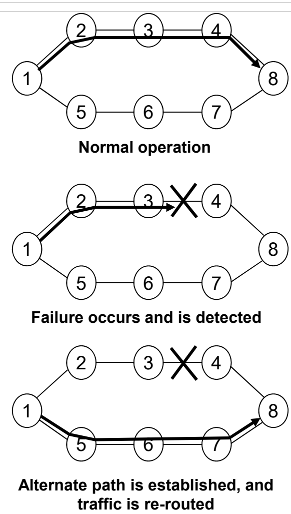
Modified from: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

MPLS Survivability

- IP routing recovers from faults in seconds to minutes
- Synchronous Optical Network (SONET) recovers in 50 ms
- MPLS targets in-between path recovery times
- Basic approaches:
  - Restoration: slower, but less capacity overhead
  - Protection: faster, but more protection capacity
- Repair methods:
  - Global repair: node that performs recovery (usually ingress node) may be far from fault, depends on failure notification message
  - Local repair: local node performs recovery (usually upstream from fault); does not require failure notification

Modified from: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

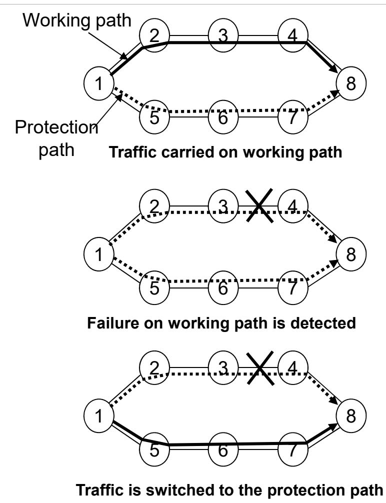
MPLS Restoration



- No protection link capacity allocated prior to fault
- New paths are established after a failure occurs
- Traffic is rerouted onto the new paths

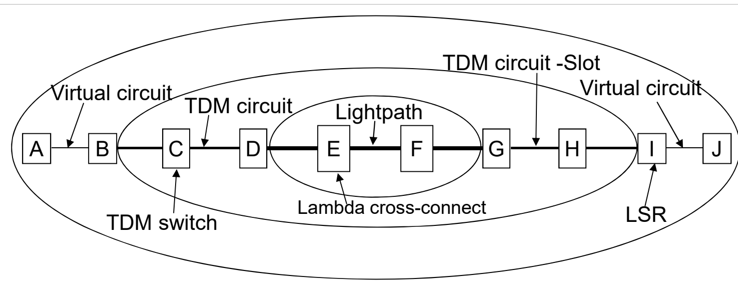
Modified from: Communication Networks:
Fundamentals Concepts and Key Architectures
Authors: A. Leon-Garcia and I. Widjaja

MPLS Protection



- Protection paths are setup as backups for working paths
  - 1+1: working path has dedicated protection path
  - 1:1: working path shares protection path
- Protection paths selected so that they are disjoint from working path
- Faster recovery than restoration

GMPLS & Hierarchical LSPs



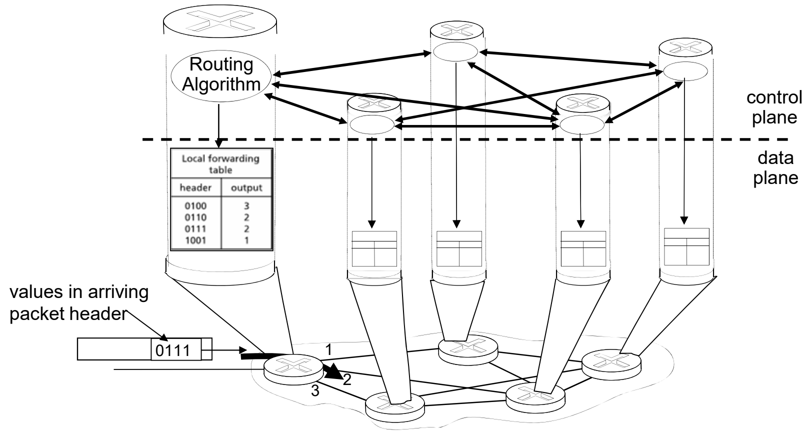
- GMPLS allows node with multiple switching technologies to be controlled by one control component
- Notion of "label" generalized:
  - TDM slot, WDM wavelength, optical fiber port
- LSP Hierarchy extended to generalized labels"
  - MPLS LSP over SONET circuit over wavelength path over fiber

Software defined networking (SDN)

- Internet network layer: historically implemented via distributed, per-router control approach:
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

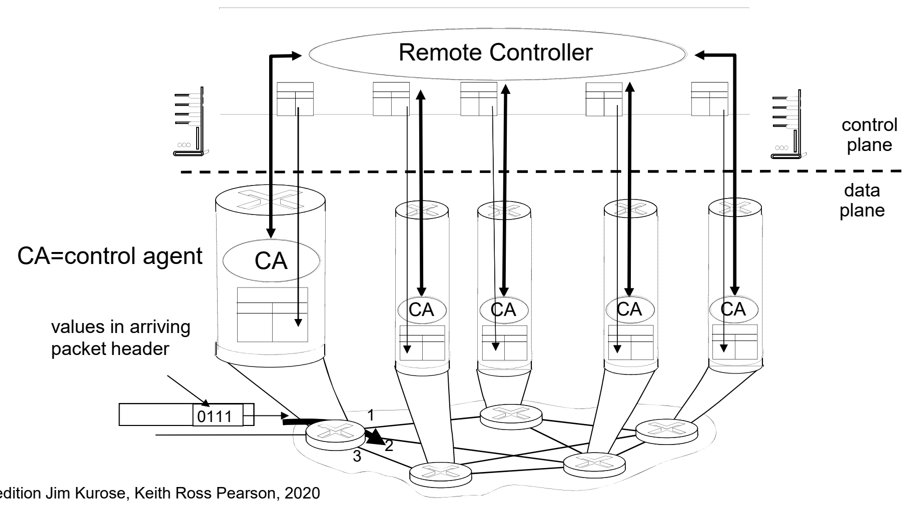
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane to computer forwarding tables



Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

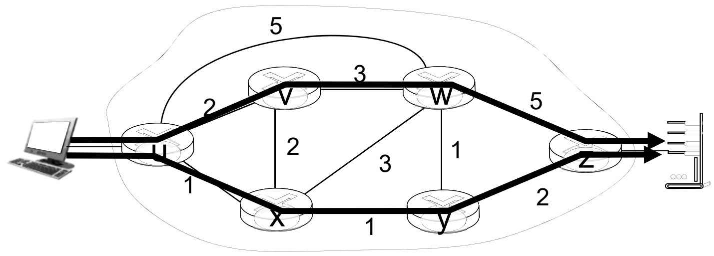
Software defined networking (SDN)

Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding allows “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Traffic engineering: difficult with traditional routing

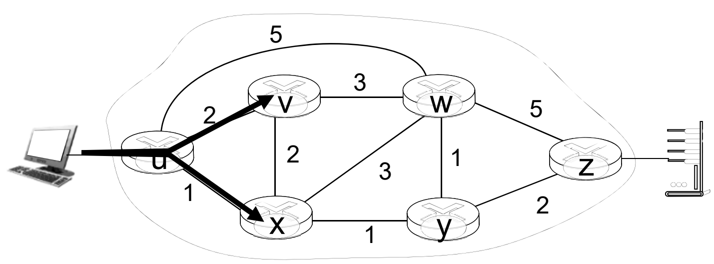


Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

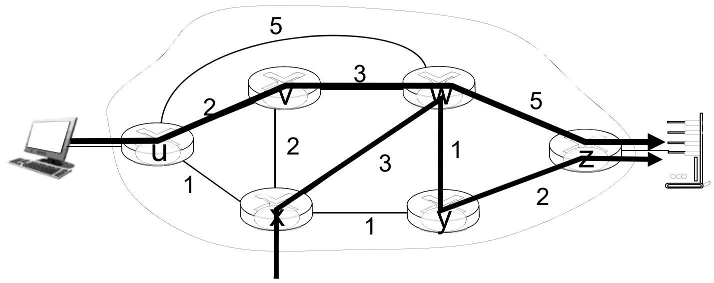
link weights are only control "knobs": not much control!

Traffic engineering: difficult with traditional routing



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?
A: can't do it (or need a new routing algorithm)

Traffic engineering: difficult with traditional routing



Q: what if w wants to route blue and red traffic differently from w to z?

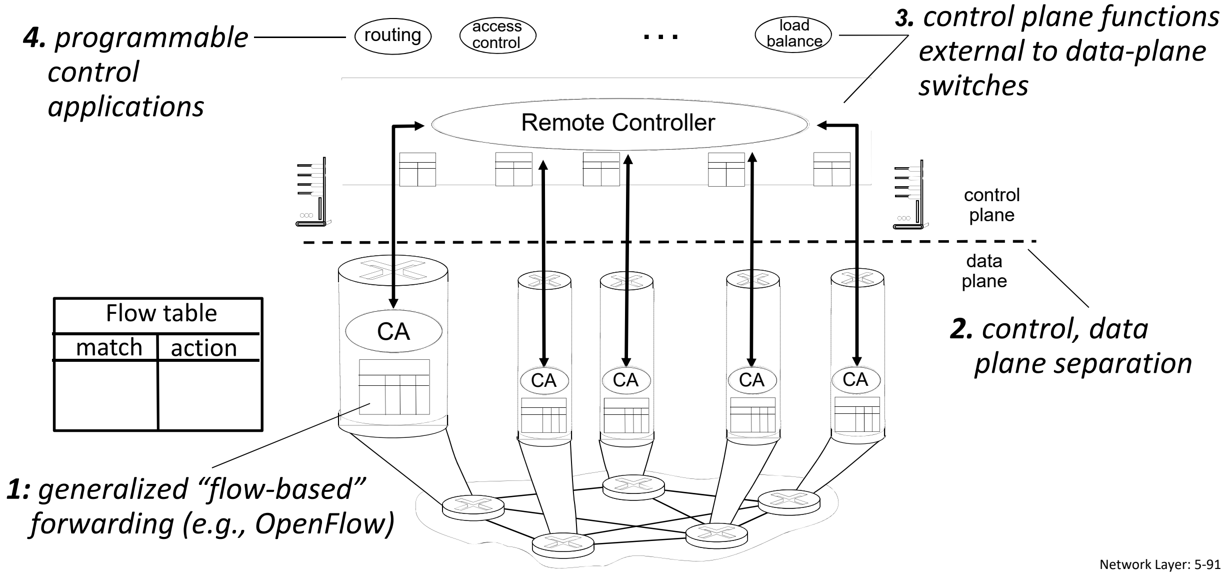
A: can't do it (with destination-based forwarding, and LS, DV routing)

Generalized forwarding can be used to achieve *any* routing desired

OpenFlow

- OpenFlow is a communication protocol that enables the centralized control of network switches and routers by external software known as a controller. It decouples the control plane (decision-making) from the data plane (forwarding of traffic) in network devices.
- Key components of OpenFlow include:
  - Flow Table: generalized forwarding: |Match|Action|Counters|
  - Controller: The controller is responsible for managing the flow tables in network devices. It communicates with these devices using the OpenFlow protocol to install, update, and remove flow entries based on network policies and conditions.
- Enables software-defined networking (SDN)

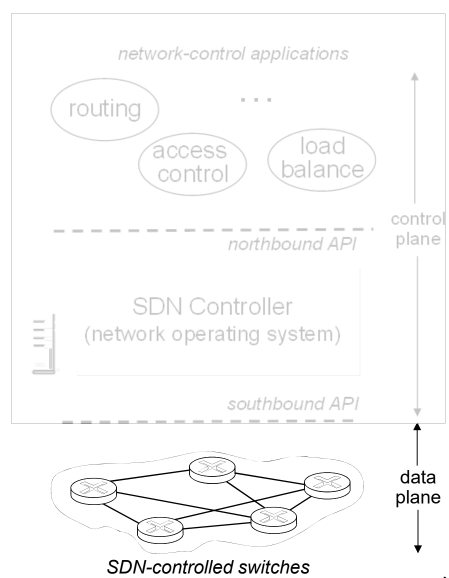
Software defined networking (SDN)



Software defined networking (SDN)

Data-plane switches:

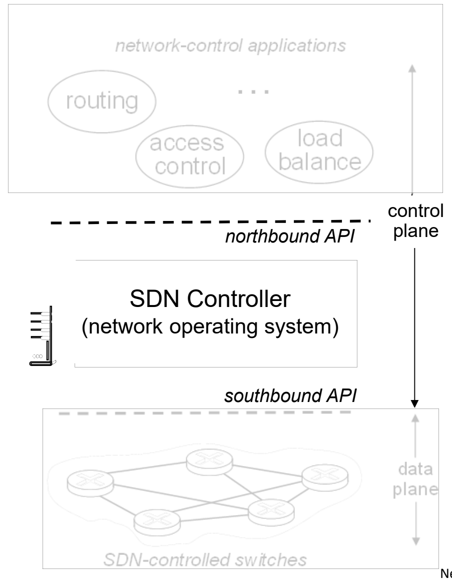
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control defines what is controllable, what is not
- protocol for communicating with controller



Software defined networking (SDN)

SDN controller (network OS):

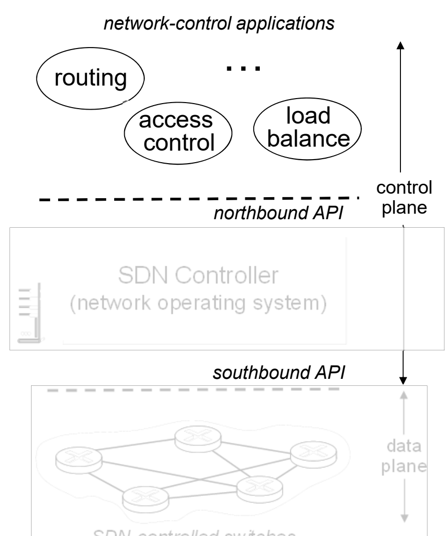
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



Software defined networking (SDN)

network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller



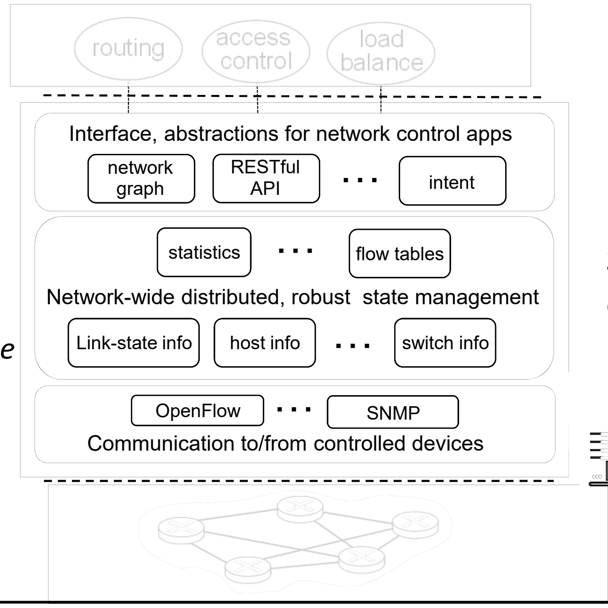
Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Components of SDN controller

interface layer to network control apps: abstractions API

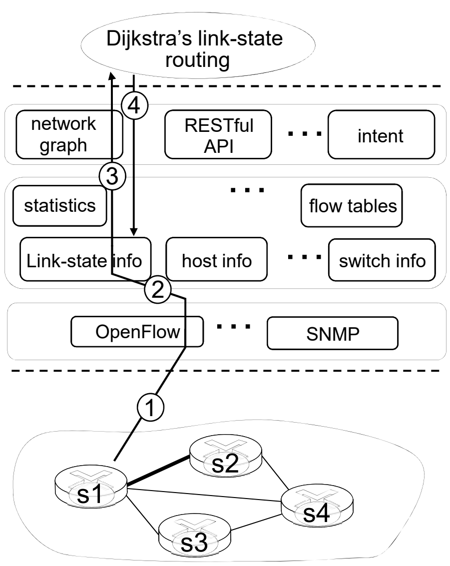
network-wide state management : state of networks links, switches, services: a *distributed database*

communication: communicate between SDN controller and controlled switches



SDN controller

SDN: control/data plane interaction example



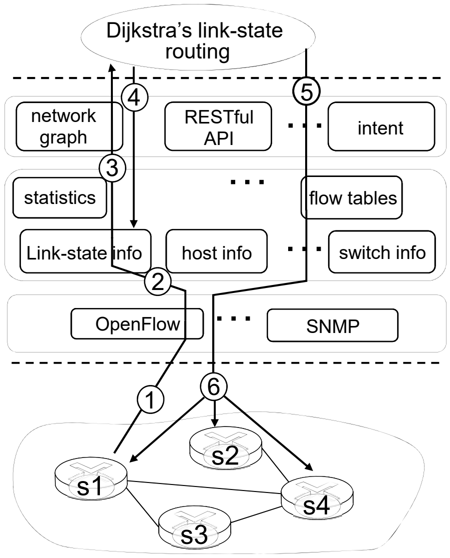
① S1, experiencing link failure uses OpenFlow port status message to notify controller

② SDN controller receives OpenFlow message, updates link status info

③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.

④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

SDN: control/data plane interaction example



⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed

⑥ controller uses OpenFlow to install new tables in switches that need updating

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, **security**: "baked in" from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., **real-time**, **ultra-reliable**, **ultra-secure**
- Internet-scaling: beyond a single AS
- SDN maybe critical in 5G and beyond cellular networks by providing necessary flexibility, programmability, and control for network operators to efficiently deploy and manage the complex and diverse requirements of 5G and beyond networks.

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

SDN and the future of traditional network protocols

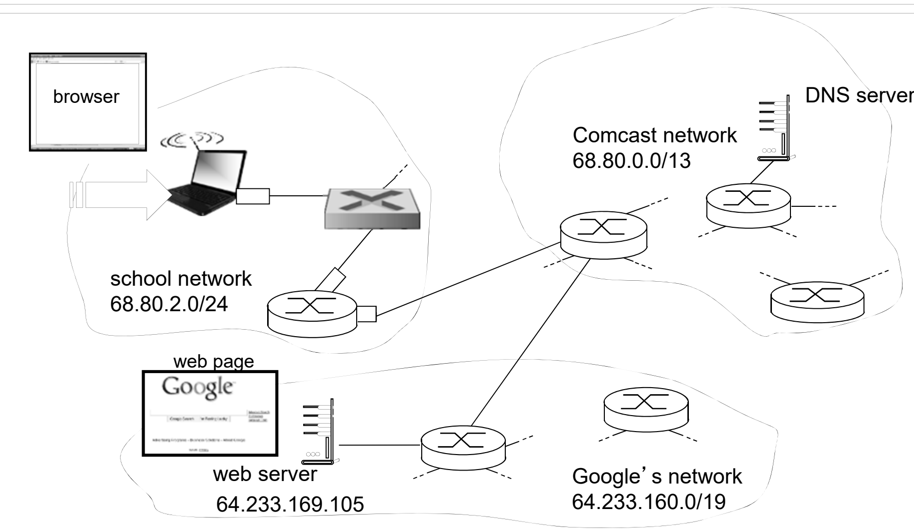
- SDN-computed versus router-computer forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
  - controller sets sender rates based on router-reported (to controller) congestion levels

How will implementation of network functionality (SDN versus protocols) evolve?

Synthesis: a day in the life of a web request

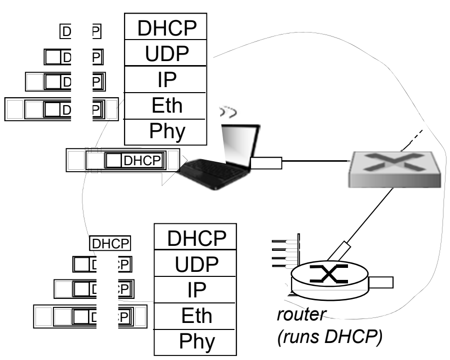
- Putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

A day in the life: scenario



Modified from: Computer Networking: A Top Down Approach
8th edition. Jim Kurose, Keith Ross

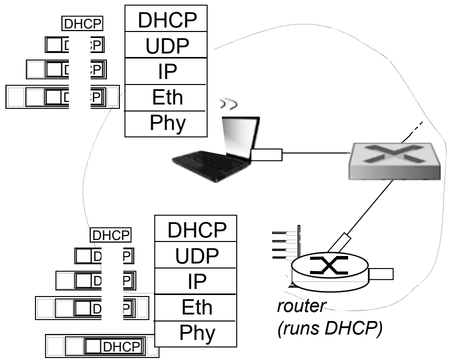
A day in the life… connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use *DHCP*
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.3 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

Modified from: Computer Networking: A Top Down Approach
8th edition. Jim Kurose, Keith Ross

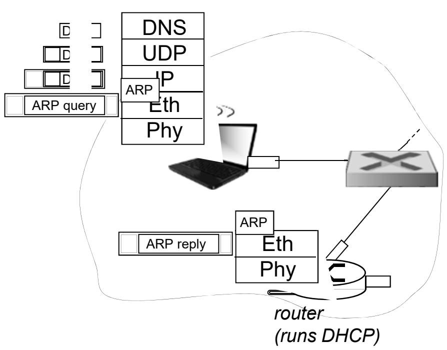
A day in the life… connecting to the Internet



- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

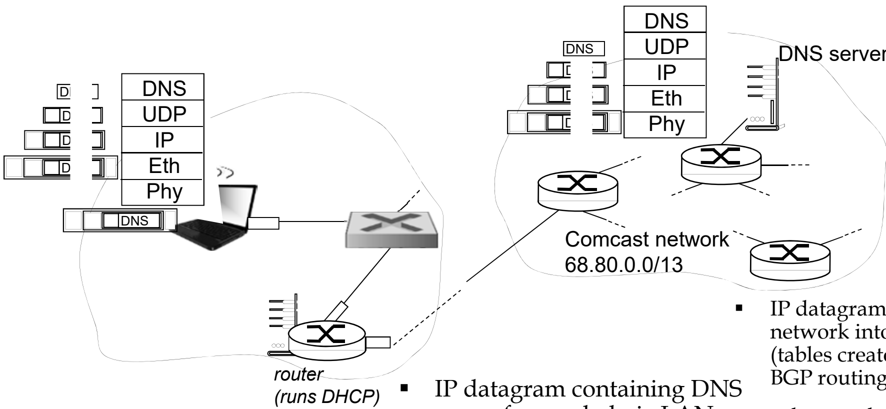
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life… ARP (before DNS, before HTTP)



- before sending *HTTP* request, need IP address of www.google.com: *DNS*
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: ARP
- ARP query broadcast, received by router, which replies with ARP reply giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

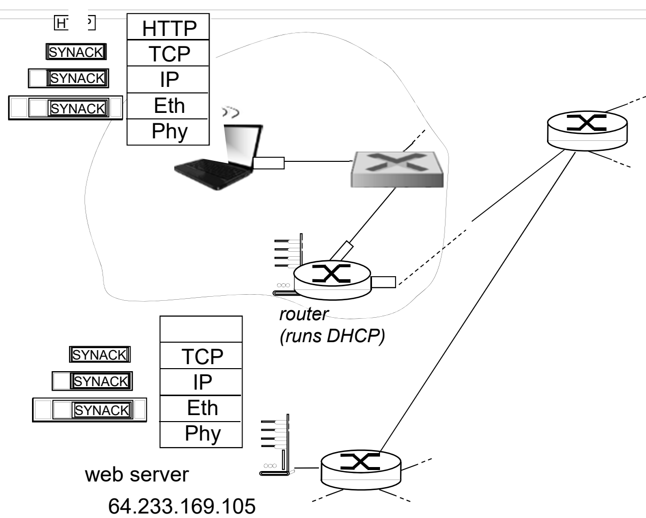
A day in the life… using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by RIP, OSPF, IS-IS and/or BGP routing protocols) to DNS server
- demuxed to DNS server
- DNS server replies to client with IP address of www.google.com

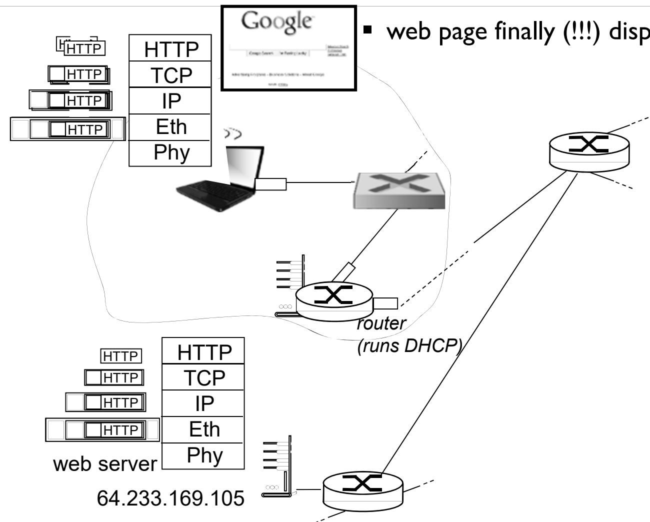
Modified from: Computer Networking: A Top Down Approach
8th edition. Jim Kurose, Keith Ross

A day in the life…TCP connection carrying HTTP



- to send HTTP request, client first opens TCP socket to web server
- TCP SYN segment (step 1 in 3-way handshake) inter-domain routed to web server
- web server responds with TCP SYNACK (step 2 in 3-way handshake)
- TCP connection established!

A day in the life… HTTP request/reply

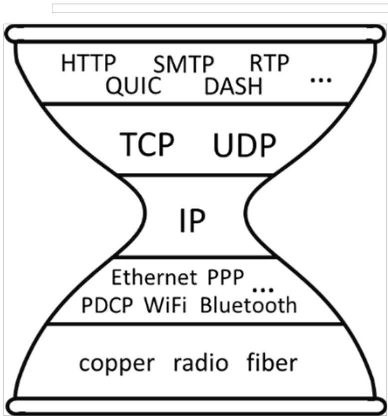


- web page finally (!!!) displayed

- HTTP request sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with HTTP reply (containing web page)
- IP datagram containing HTTP reply routed back to client

Modified from: Computer Networking: A Top Down Approach
8th edition. Jim Kurose, Keith Ross

Internet Architecture: Revisited



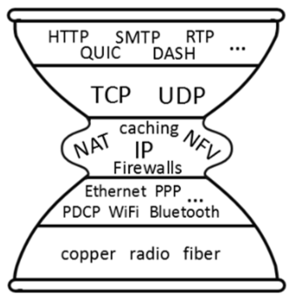
RFC 1958

"Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network."

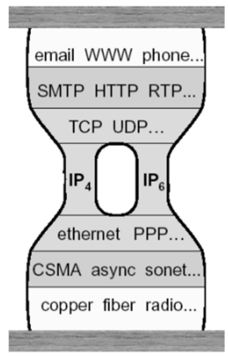
Three cornerstone beliefs:
- simple connectivity
- IP protocol: that narrow waist
- intelligence, complexity at network edge

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

IP Hourglass Architecture: Revisited

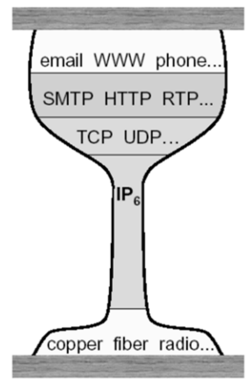


New protocols require more functionality from underlying networks



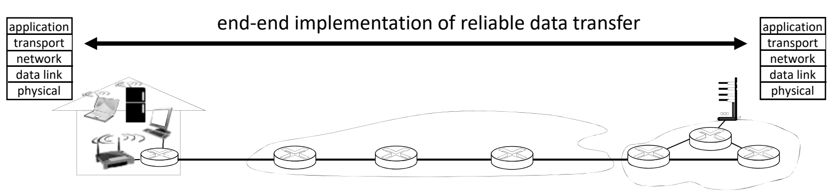
1) Doubles number of service interfaces
2) Requires changes above & below
3) Creates interop problems

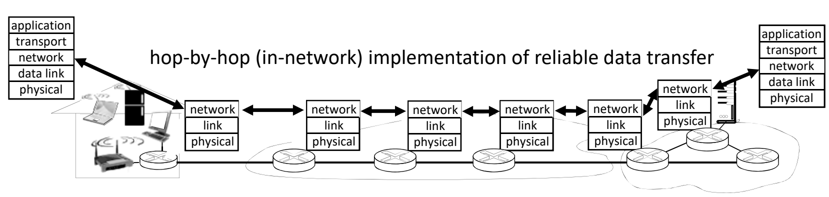
Someday??



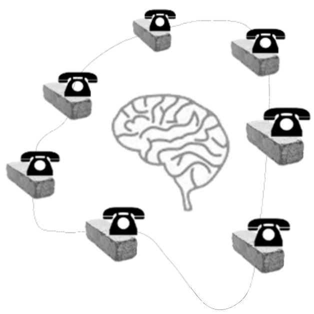
The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in network, or at network edge

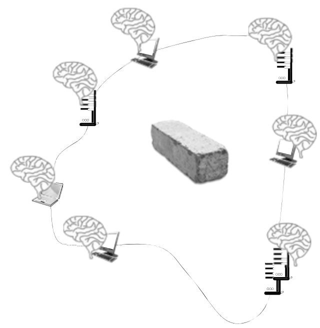




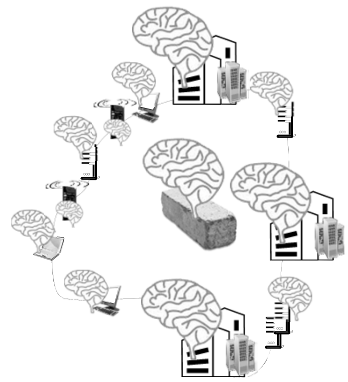
Where’s the intelligence?



20th century phone net:
- intelligence/computing at network switches



Internet (pre-2005)
- intelligence, computing at edge



Internet (post-2005)
- programmable network devices
- intelligence, computing, massive application-level infrastructure at edge

Modified from: 8th edition Jim Kurose, Keith Ross Pearson, 2020

Challenges

- Challenges
  - Trust
  - Network and configuration management
  - Scalability and control of system complexity
  - Predictable performance
  - Performance evaluation and comparison of different architectures
- Approaches and mechanisms are now being woven together into coherent, overarching candidate designs for a future Internet.