Midterm Recap

50.012 Networks

Jit Biswas

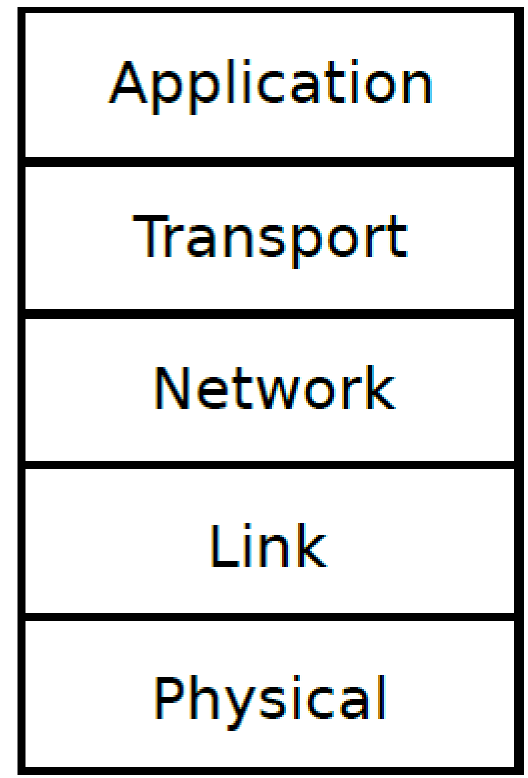
Outline

- Introduction to Networks
- Note: parts of this slide set are based on Kurose & Ross slide set

Introduction to Networks

Protocol Layers

- We can map these different views on different layers
- Here, we use a layer model with 5 layers
- Layers only "see" the neighboring layer
- Reduces complexity in design of system
  - Similar to encapsulation in programming
  - Allows isolated changes to part of system



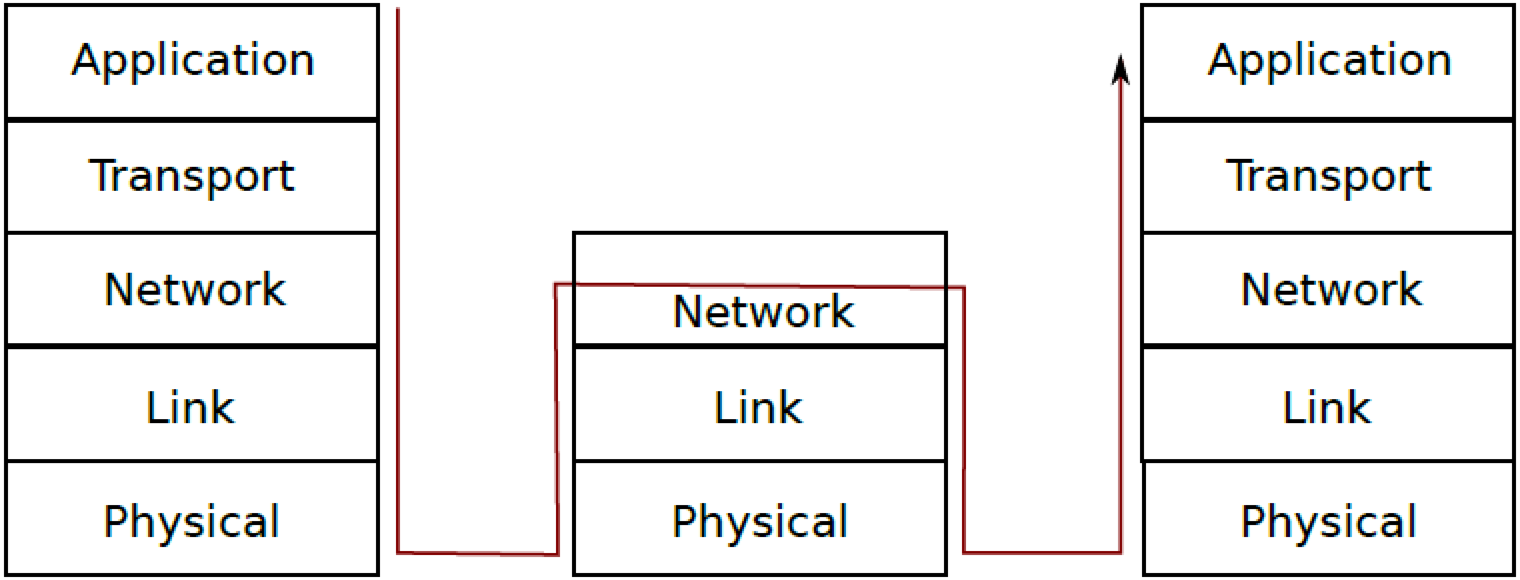
Data processed in each layer

Each layer adds further data, we use different terms to refer to data in each layer:

- Application layer: messages
  - Data, in format understandable by app (e.g. JSON)
- Transport layer: segment
  - Added data: port numbers, flow control, integrity checks
- Network layer: datagram
  - Added data: IP addresses
- Link layer: frame
  - Added data: MAC address, checksums, channel access data
- Physical layer: symbols
  - More like translation of data into analog representations

More on layers

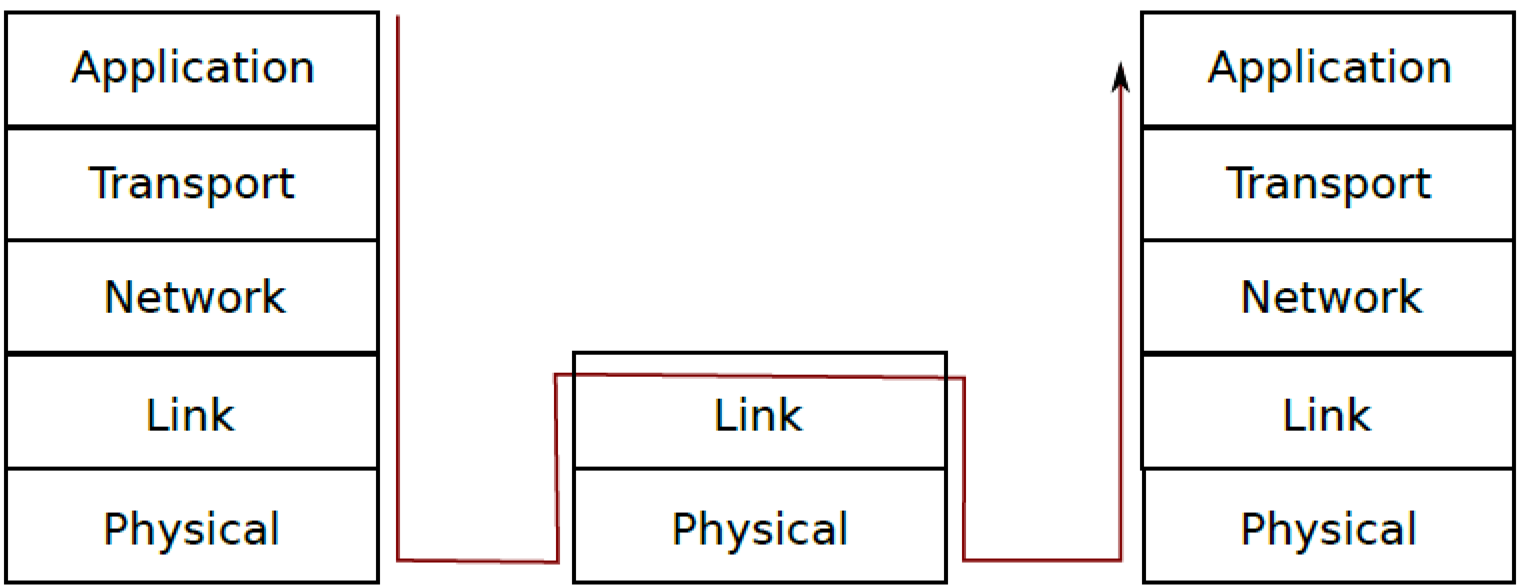
- Network core usually only required lowest three layers
  - Intermediate links could even have only two layers
- Only the network edge has all the complexity, i.e. all five layers



Transport through L3 router

More on layers

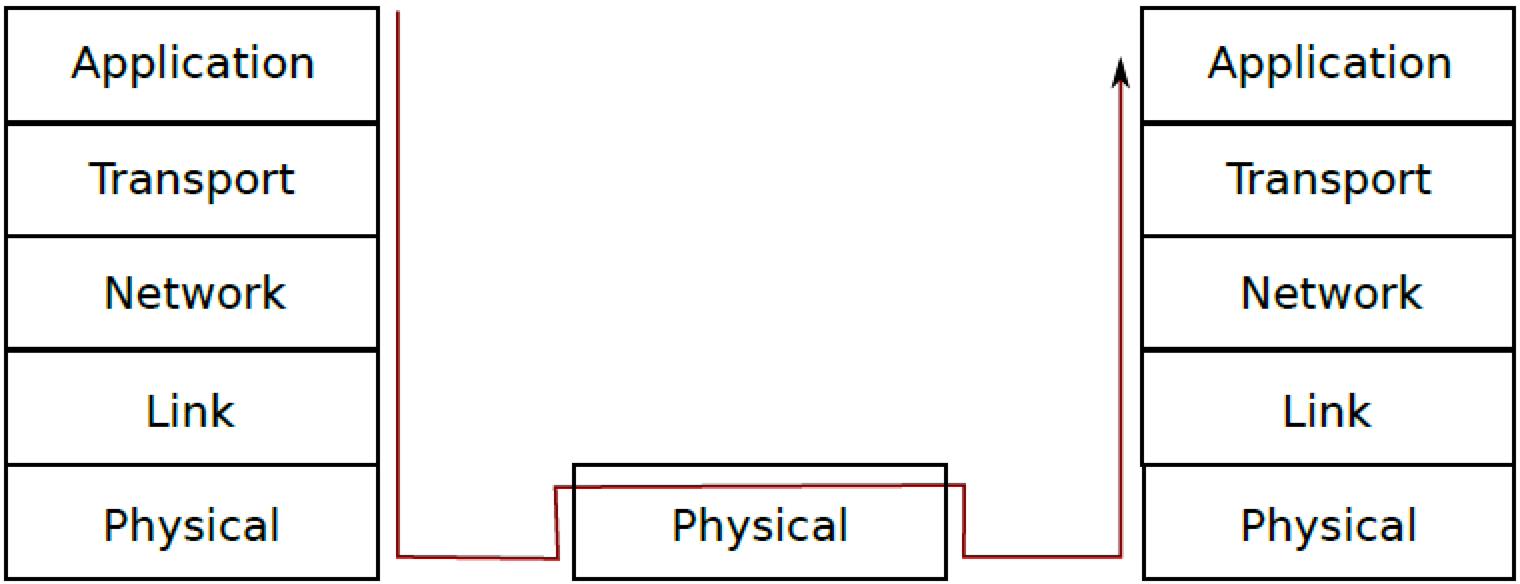
- Network core usually only required lowest three layers
  - Intermediate links could even have only two layers
- Only the network edge has all the complexity, i.e. all five layers



Transport through L2 switch

More on layers

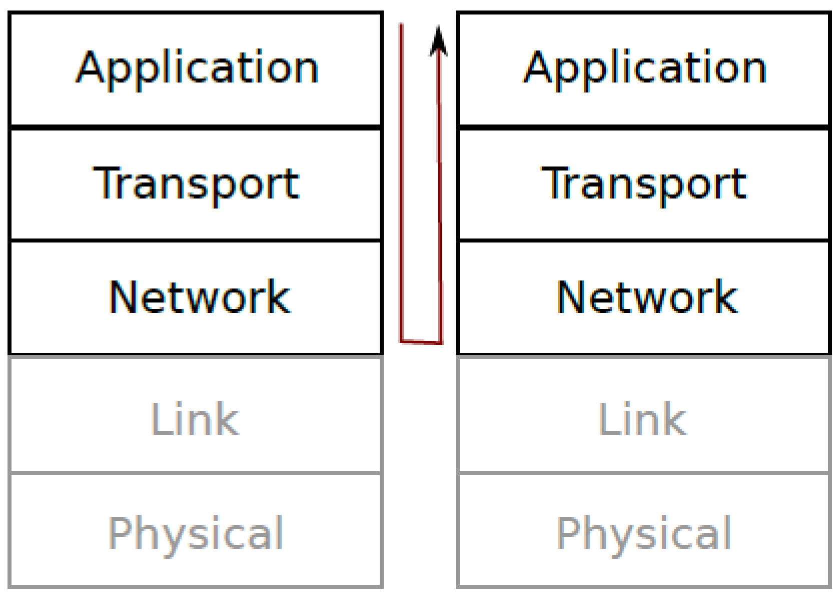
- Network core usually only required lowest three layers
    - Intermediate links could even have only two layers
- Only the network edge has all the complexity, i.e. all five layers



Transport through L1 repeater / hub

More on layers

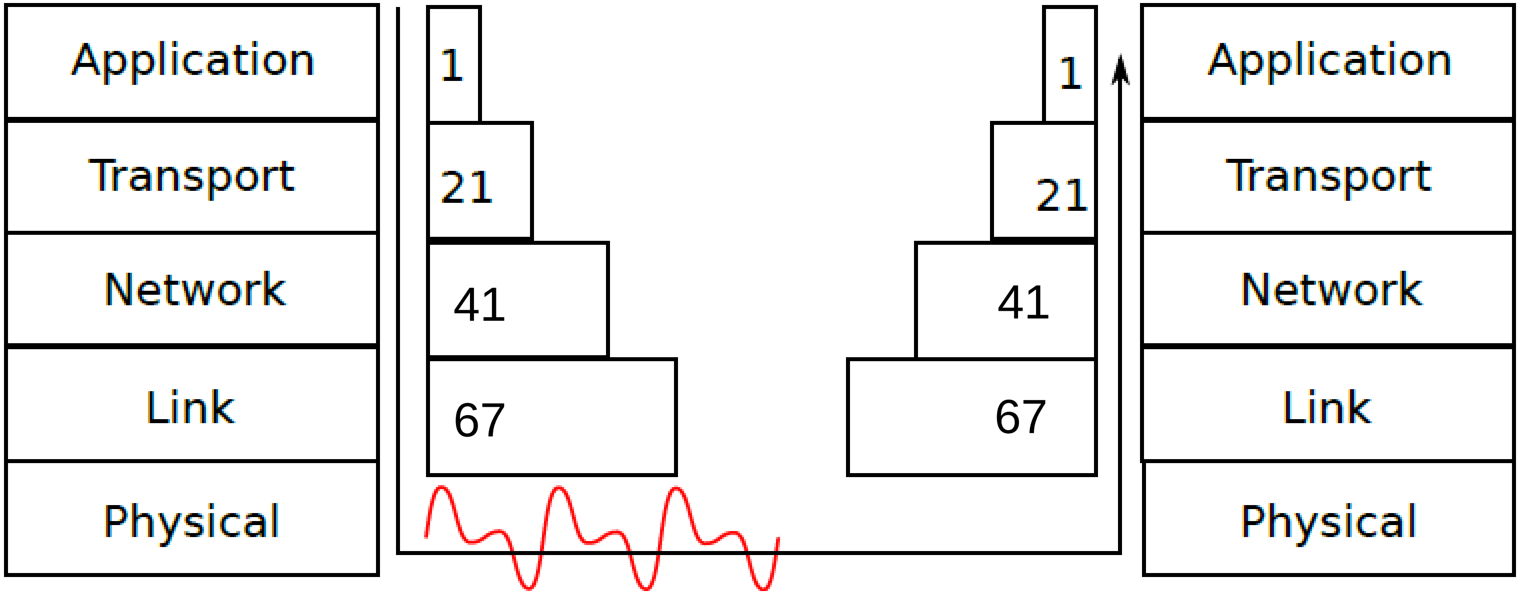
- Network core usually only required lowest three layers
  - Intermediate links could even have only two layers
- Only the network edge has all the complexity, i.e. all five layers
- Communication between applications on the same machine might skip link layer and lower



Local Transport through Loopback interface

Overhead through layers

- Messages sent from one application to another:
  - Descend the layers to physical layer at sender
  - Ascend the layers to application layer at receiver
- With each layer descent, additional information is added
  - A 1 Byte application layer message can be inflated to 67 Bytes (TCP+IP+ETH)



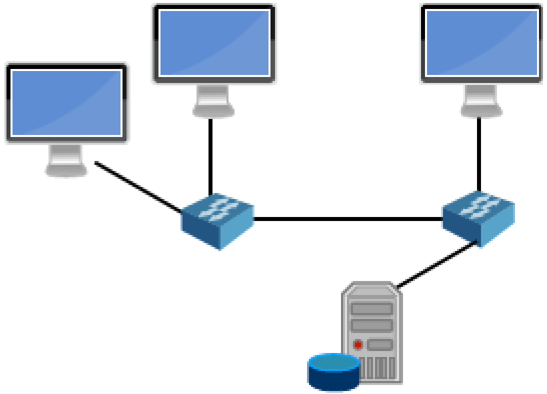
Throughput, Latency, Loss

What are the performance characteristics of networks?

- Throughput: How much data can be transmitted per time slot
- Latency: How long does it take to transmit a packet
  - Transmission delay: size*throughput
  - Propagation delay: How long does the message travel
  - Processing delay: at sender/receiver, intermediate nodes
  - Queuing delay: how long does the message wait in intermediate buffers
- Reliability/Loss: Will all packets be delivered, or will some be lost?

Packet Switching vs Circuit Switching

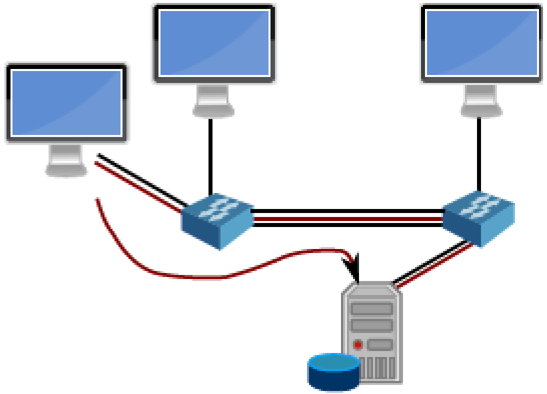
- How are packets transmitted end-to-end?
- Circuit Switching: each link can have multiple circuits
  - Prior to transmission, a sequence of circuits is reserved
  - These circuits are then exclusively used to transmit message
  - While reserved, circuits are only used by message
- Packet Switching: message is split into packets
  - Each packet is routed individually
  - Links are shared with other transmissions
  - No guarantees on max delay, higher throughput



Example Network

Packet Switching vs Circuit Switching

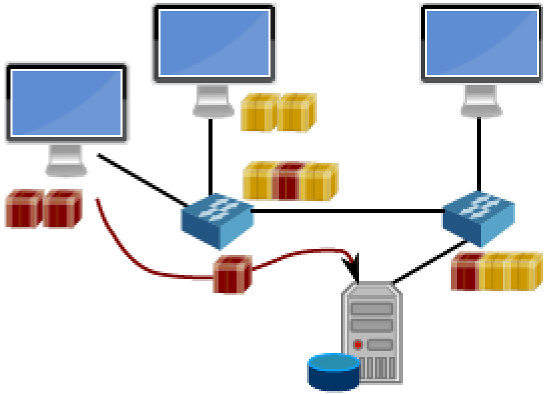
- How are packets transmitted end-to-end?
- Circuit Switching: each link can have multiple circuits
  - Prior to transmission, a sequence of circuits is reserved
  - These circuits are then exclusively used to transmit message
  - While reserved, circuits are only used by message
- Packet Switching: message is split into packets
  - Each packet is routed individually
  - Links are shared with other transmissions
  - No guarantees on max delay, higher throughput



Circuit-switching: Path of circuits reserved for transmission

Packet Switching vs Circuit Switching

- How are packets transmitted end-to-end?
- Circuit Switching: each link can have multiple circuits
  - Prior to transmission, a sequence of circuits is reserved
  - These circuits are then exclusively used to transmit message
  - While reserved, circuits are only used by message
- Packet Switching: message is split into packets
  - Each packet is routed individually
  - Links are shared with other transmissions
  - No guarantees on max delay, higher throughput



Packet-switching: Transmissions as packets through shared links

Packet vs Circuit Switching II

- Circuit switching is the traditional telephone network approach
  - Guaranteed capacity for high quality calls
  - Pricing is based on time, not volume
- Packet switching is key concept of Internet and computer networks
  - Best effort transmission of packets
  - Much better capacity use
  - Great for bursty data
  - Simpler, no call setup
  - Traffic-based billing (higher yield for operator)
  - No transmission guarantees, e.g. queuing loss
- Downside - excessive congestion possible: packet delay and loss
  - protocols needed for reliable data transfer, congestion control

The Internet

What defines the Internet?

The Internet is

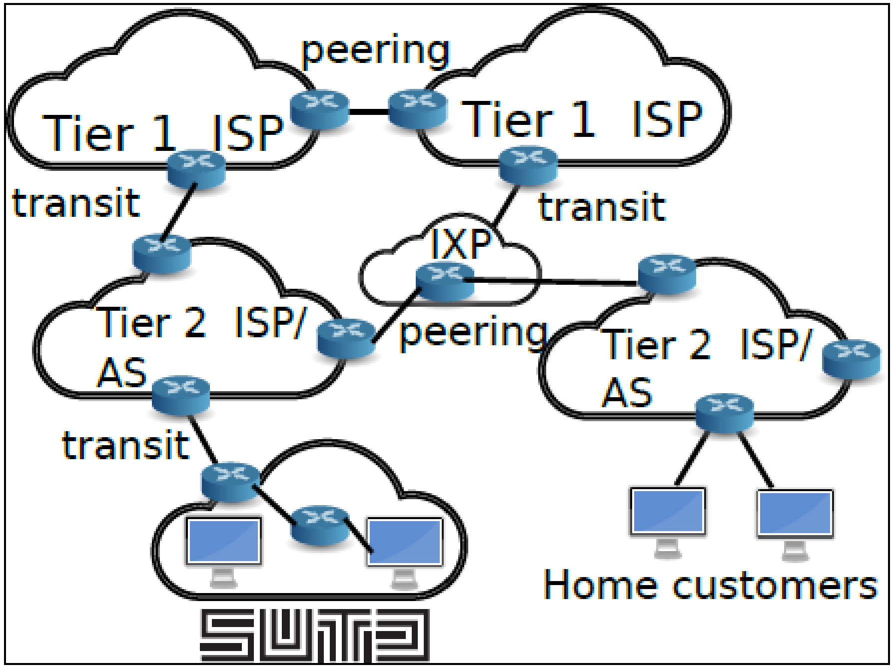
- A collection of connected autonomous systems (AS)
- The language/API between systems is IP (L3)+ UDP/TCP (L4)
- The Internet is more than the WWW
- The Internet is not every world-wide network

Internet Service Providers

- The internet core is consisting of Internet Service Providers and large corporate networks
- The ISPs forming the core are also called Tier 1 ISPs
- A country typically has several Tier 1 ISP, Singapore has 3:
  - Singtel
  - Starhub
  - Pacific Internet
- Below Tier 1, there are intermediate ISPs, and then Access ISP
- Typically, data exchange with higher tier ISPs will be billed
  - So Tier 1 ISPs never pay other ISPs for forwarded traffic
- Direct Data exchange with same-tier ISPs will be free
  - But lower tier ISPs/customers have to pay to access higher tiers

Internet Exchange Points

- Internet Exchange Points (IXPs) provide fast connections to Tier1 ISPs and important backbone connections
- They are often used to connect multiple Tier 1 ISPs together
- Closest large one to Singapore: Hong-Kong (~200 participants/200+GBit/s)
- http://www.internetexchangemap.com/



Design Challenge: Domain names

- IP addresses are not very human friendly
- They also directly contain network-specific parts, so they can change when a service is relocated
- Domain Names offer a convenient abstraction
  - Strings, so often human-readable and easy to remember
  - Higher layer of abstraction, don’t change when location of service changes
- But how can we route traffic to a domain name?
- The Domain Name Service translates from names to IP addresses
  - First, client contacts DNS server for IP of domain
  - Then, traffic is sent to IP address

DNS Hierarchy

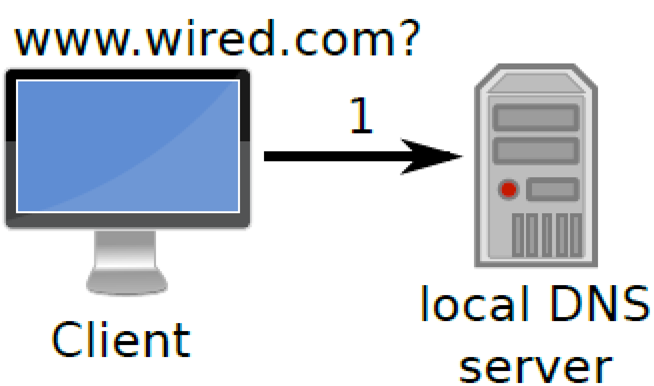
- DNS to IP assignments can change frequently
- One single DNS server for the internet is not going to work
- How do we know which server to connect to?
  - DNS names are hierarchical: thirdLevel.secondLevel.topLevel
  - So, .com, .sg, are top level domains
  - For each, there is a DNS server that has either:
    - A direct mapping to an IP
    - A reference to a lower level DNS server
- Example: foo.bar.com
  - .com server might not have entry
  - Will ask bar.com domain DNS server, that has "foo" entry

DNS bootstrapping

- But how to find the .com DNS server?
- Every network will have a dedicated DNS server
- DNS works recursively, i.e. queries are forwarded
- Local server handles local lookups, or forward to higher servers

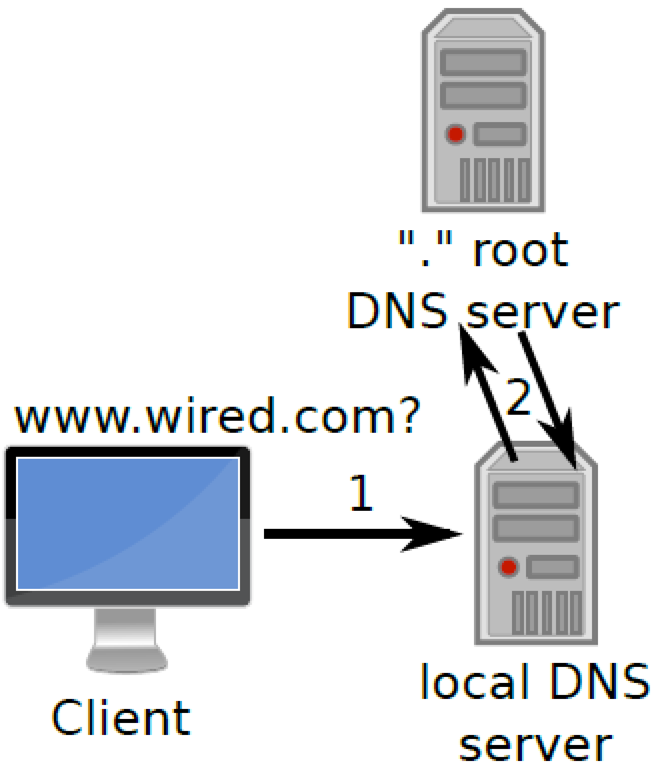
DNS bootstrapping

- But how to find the .com DNS server?
- Every network will have a dedicated DNS server
- DNS works recursively, i.e. queries are forwarded
- Local server handles local lookups, or forward to higher servers



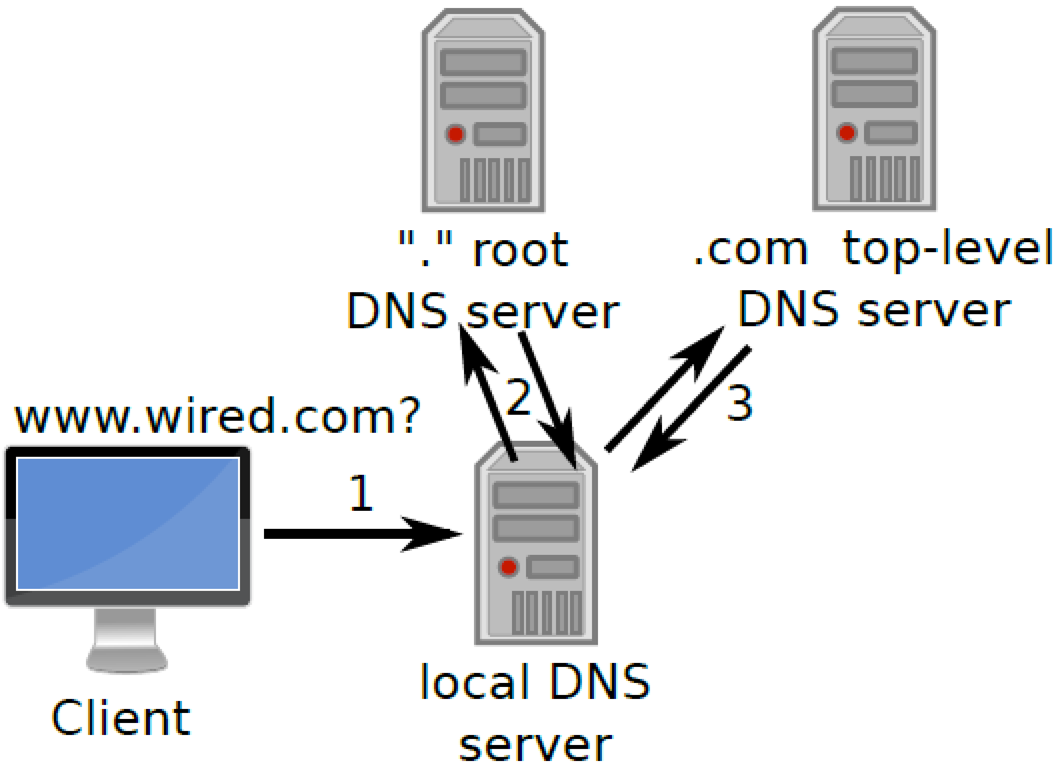
DNS bootstrapping

- But how to find the .com DNS server?
- Every network will have a dedicated DNS server
- DNS works recursively, i.e. queries are forwarded
- Local server handles local lookups, or forward to higher servers



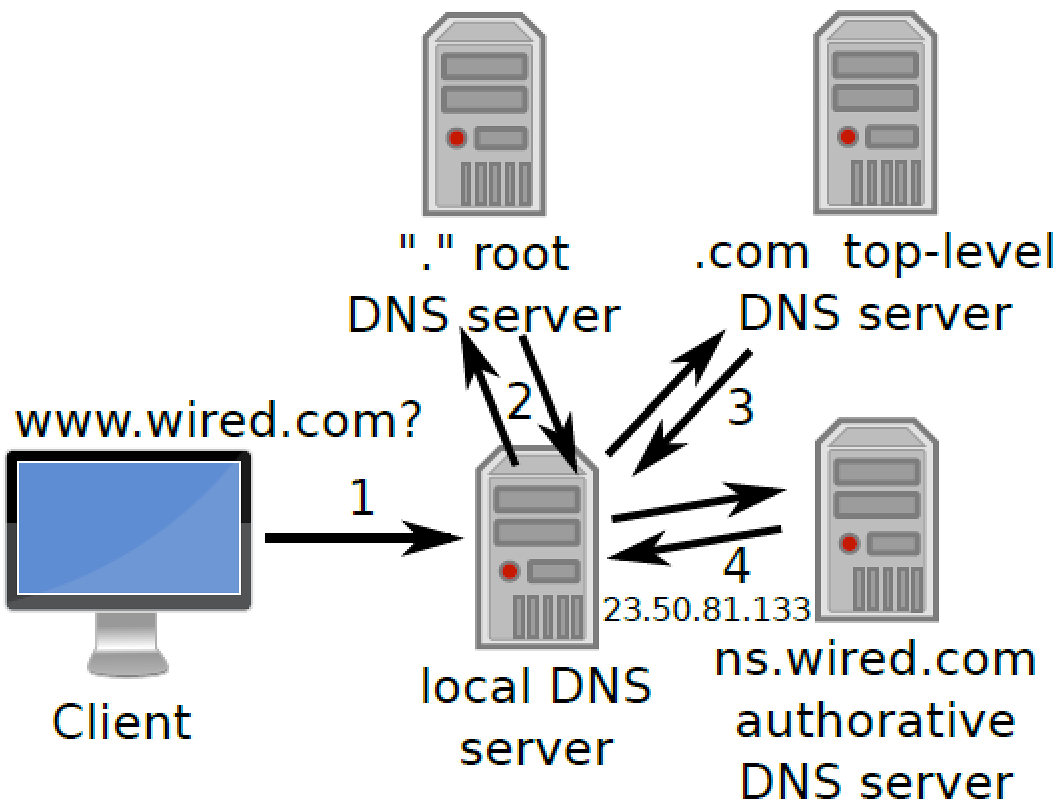
DNS bootstrapping

- But how to find the .com DNS server?
- Every network will have a dedicated DNS server
- DNS works recursively, i.e. queries are forwarded
- Local server handles local lookups, or forward to higher servers



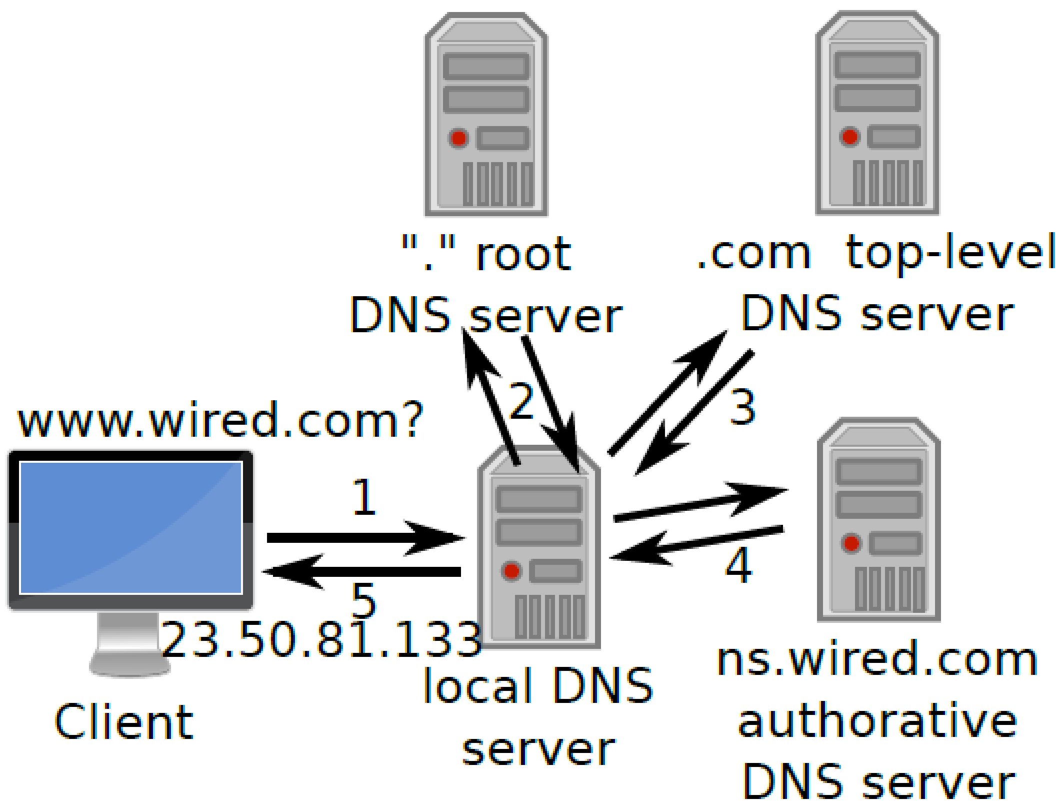
DNS bootstrapping

- But how to find the .com DNS server?
- Every network will have a dedicated DNS server
- DNS works recursively, i.e. queries are forwarded
- Local server handles local lookups, or forward to higher servers



DNS bootstrapping

- But how to find the .com DNS server?
- Every network will have a dedicated DNS server
- DNS works recursively, i.e. queries are forwarded
- Local server handles local lookups, or forward to higher servers



DNS Record Types

- A – Mapping of name to IPv4 Address(es)
- AAAA – Mapping of name to IPv6 Address(es)
- MX – Mail Exchange for name
- CNAME – Canonical name (alias)
- TXT – Text record associated with name
- NS – Name server responsible for domain

Global addressing in IPv4

- IP addresses consist of 4 8-bit numbers
  - e.g. 192.168.0.1, 10.0.4.134 (in decimal)
  - In total the 256^4=4.3 Billion addresses mentioned earlier
- How can we find the route from one IP address to another?
  - Store a route for each target IP? 4.3 Billion routes
  - Maybe a central server that stores all routes?
- How would graphs for both cases look like?

Internet Topology

- The actual topology of the Internet is a hybrid between a fully connected graph, and a star graph
- core: few highly connected nodes with many edges
- edge: consists of many poorly connected nodes with only one edge
- Addressing reflects that
  - "Divide and conquer" is used: divide address space into local subgroups (subnets)
  - Routing is done between subnets
  - Within a subnet, routing is only local
  - In general: "left part" of IP address denotes network, "right part“ denotes host

Class-based Subnets

- Originally, there were three possible classes of subnets: A, B, C
  - Up to 256 Class A networks with each 256^3 addresses
  - Up to 256^2 Class B networks with 256^2 addresses
  - Up to 256^3 Class C networks with 256 addresses
  - Compared to our tree view: two layers, nodes in l1 have up to $256^{1|2|3}$ children
- Companies or providers would get a whole Class A,B,C network
- But these classes were not well matched to realistic use:
  - Class C was too small for companies or universities
  - Class B was too big for almost all users (65k addresses)
  - So nowadays, subnets can have 2^n addresses for variable n

Routing between Subnets

- Routes on the internet are advertised for subnets (prefixes)
  - Subnets always have their host part set to 0 (e.g. 1.2.0.0)
  - "to reach any of 1.2.0.0, come here"
  - To simplify routing, small subnets are aggregated if possible
    - Instead of "to reach 1.2.0.0 or 1.2.1.0 or 1.2.2.0 or 1.2.3.0, come here", you advertise "to reach 1.2.[0-3].0, come here"
    - We will introduce the exact notation for this soon
- If specific routes to target subnet is not available, you can always route towards parent subnet
  - This should get you closer to the target
- Vague similarity to country/area code in phone calls

Private Subnets

- Most IP addresses are globally unique
- Three private range(s) were defined
  - 10.0.0.0 - 10.255.255.255
  - 172.16.0.0 - 172.31.255.255
  - 192.168.0.0 - 192.168.255.255
- These IP addresses will not be globally unique
- Internet edge routers will not forward traffic to these
  - This is why your 192.168.0.1 IP is not reachable from Internet
- To allow private IPs to communicate with Internet:
  - Network Address Translation (NAT) is required (more later)

Classless-Interdomain-Routing (CIDR)

- The CIDR format allows finer-grained division of IP space
- Arbitrary cut-off point between network and individual address
- Allows subnets smaller than Class B, larger than Class C
- Out of the 4x8 bits of an IP address, leftmost n are used to describe the subnet, and 32-n are used for the host
  - Notation: v.x.y.z/n, with n the length of the subnet
  - Example: 1.2.3.4/24 denotes host 4 in subnet 1.2.3.0/24

Link

Subnet Masks

- Example: 1.2.65.4/22
  - What is the subnet address?
  - What is the host address?
- To easily find both, create subnet mask with n Ones and 32-n zeros
  - AND this subnet mask with binary representation of IP address
  - Result is the subnet address
  - Invert mask + apply with AND to get host address

```
IP:   0000 0001 0000 0010 0100 0001 0000 0100
Mask: 1111 1111 1111 1111 1111 1100 0000 0000
Net : 0000 0001 0000 0010 0100 0000 0000 0000
```

So subnet is 1.2.64.0/22, host part is 01 0000 0100, i.e. the 260th host in the sub-range.

Broadcasting

- What if you want to send a message to everyone on the local network?
  - For n recipients, you send n messages? Inefficient!
  - We need a target address that everyone listens to!
- The broadcast address is the highest address in the subnet
  - So, for 192.168.0.0/24 the broadcast address is
    - 192.168.0.255
  - While for 192.168.0.0/23 the broadcast address is
    - 192.168.1.255

The Application Layer

Overview of the application layer

- Contains protocols that are specific to applications
- These protocols can rely on lower layers to provide routing, error correction, etc.
- Examples application layer protocols:
  - FTP and SMTP
  - HTTP traffic with HTML and image content
  - FTP traffic with binary data
  - NTP traffic to synchronize the time
  - Skype traffic for audio/video chat
- Addressing on application layer: URLs/URIs
- Socket: interface between the application and transport layer!

Application layer addressing

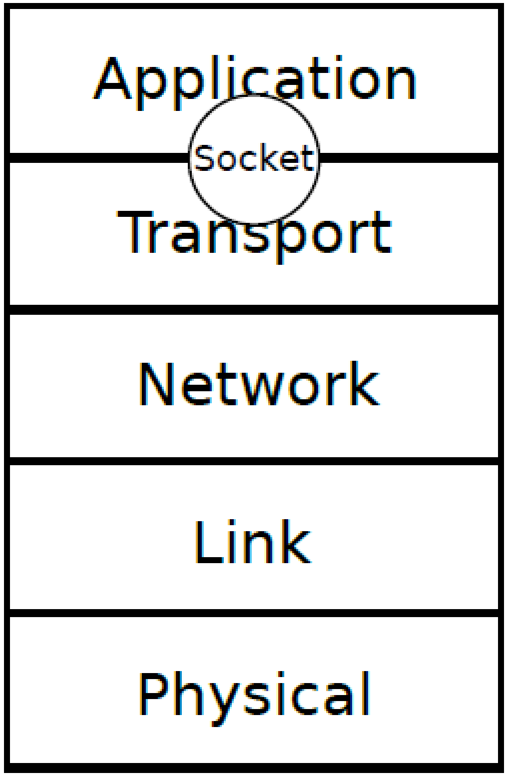
- Servers can be addressed using domain names
- How can we address resources on a certain system?
- Must be globally unique name, may come with protocol identifier (allows client to use correct application to handle URL content)
- This is what uniform resource locators (URLs) provide:
  - protocol://domain.name/resource/path
  - `https://edimension.sutd.edu.sg/webapps/login/`
  - Globally unique, resolves to different IPs from in/outside SUTD
- Custom protocol IDs are possible, if the client can handle them
  - example: Apple's itms:// protocolID for the iTunes store

URI, URN, URLs

- Terminology:
  - URI = Universal Resource Identifier
  - URL = Universal Resource Locator
  - URN = Universal Resource Name
- URNs and URLs are both URIs
- What is the difference?
  - URI = Uniquely identifies a resource
  - URL = identifies + provides location of a resource
  - URN = identifies, persistent in time (e.g., after deletion)

Sockets

- From CS perspective: object that can you can write, read, or listen on
- Create the socket with lower layer information (e.g., transport protocol, port number, IP address)
- Receiver will run application with similar socket
- Data pushed into sender socket will magically appear at receiver socket

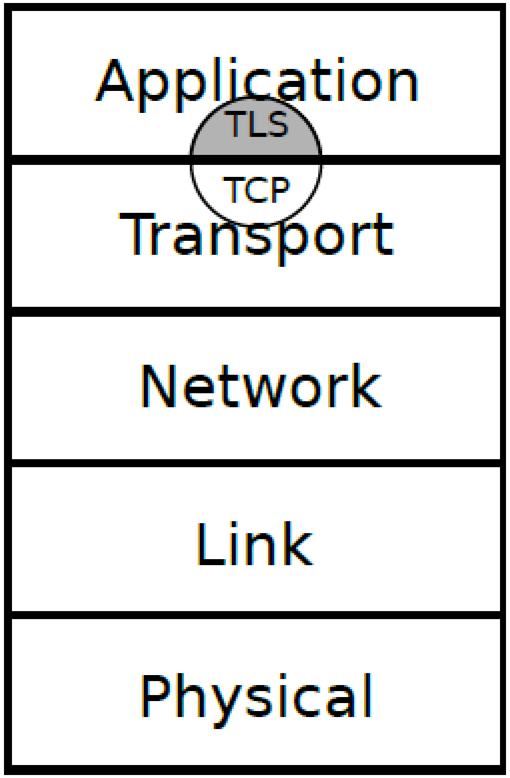


“Services” provided to the lower layers

- "Nice to have" properties for message transport
  - Message integrity checks
  - Good throughput
  - Low transmission delays
  - Security
- This is what is available:
  - TCP (on transport layer) provides integrity and flow control
  - UDP (on transport layer) does not provide any of the properties
- Add "security" through TLS on application layer (over TCP)
- No guarantees on low transmission delays or good throughput

Transport Layer Security (TLS)

- We will discuss TLS in detail in 50.020
- TLS is somewhat "in between" application and transport layer
- It relies on TCP, but provides services to applications
- TLS can be configured in different ways, able to provide
  - Message authentication, confidentiality, integrity
- TLS is often used to improve security of application layer protocols
  - e.g. HTTP over TLS = HTTPS

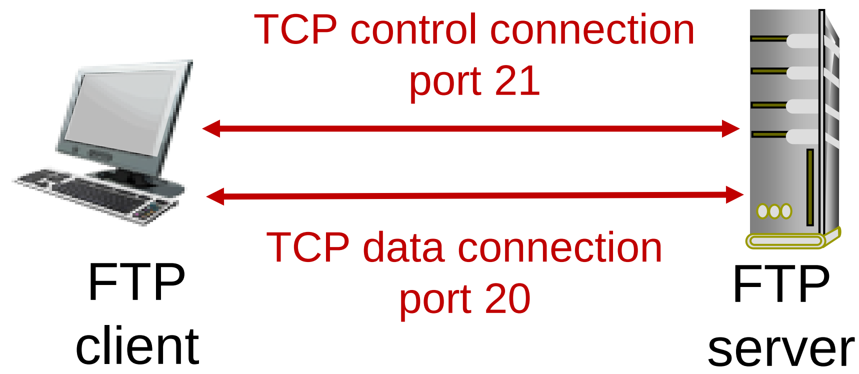


What are protocols?

- In general: "language" spoken between machines
- Protocols generally define four aspects:
  - Types of messages exchanged (e.g., request, response)
  - Message syntax (e.g. format)
  - Message semantics (how to interpret)
  - Rules for processing of messages (how to react)
- Open protocols have these specified publicly, but proprietary protocols also exist (e.g. Skype)
- If protocols do not encrypt the messages, it is often possible to reverse-engineer unspecified protocols

FTP: separate control, data connections

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol
- Client obtains authorization over control connection
- Client browses remote directory by sending commands over control connection.
- When server receives a command for a file transfer, the server opens a TCP data connection to client
- After transferring one file, server closes connection.

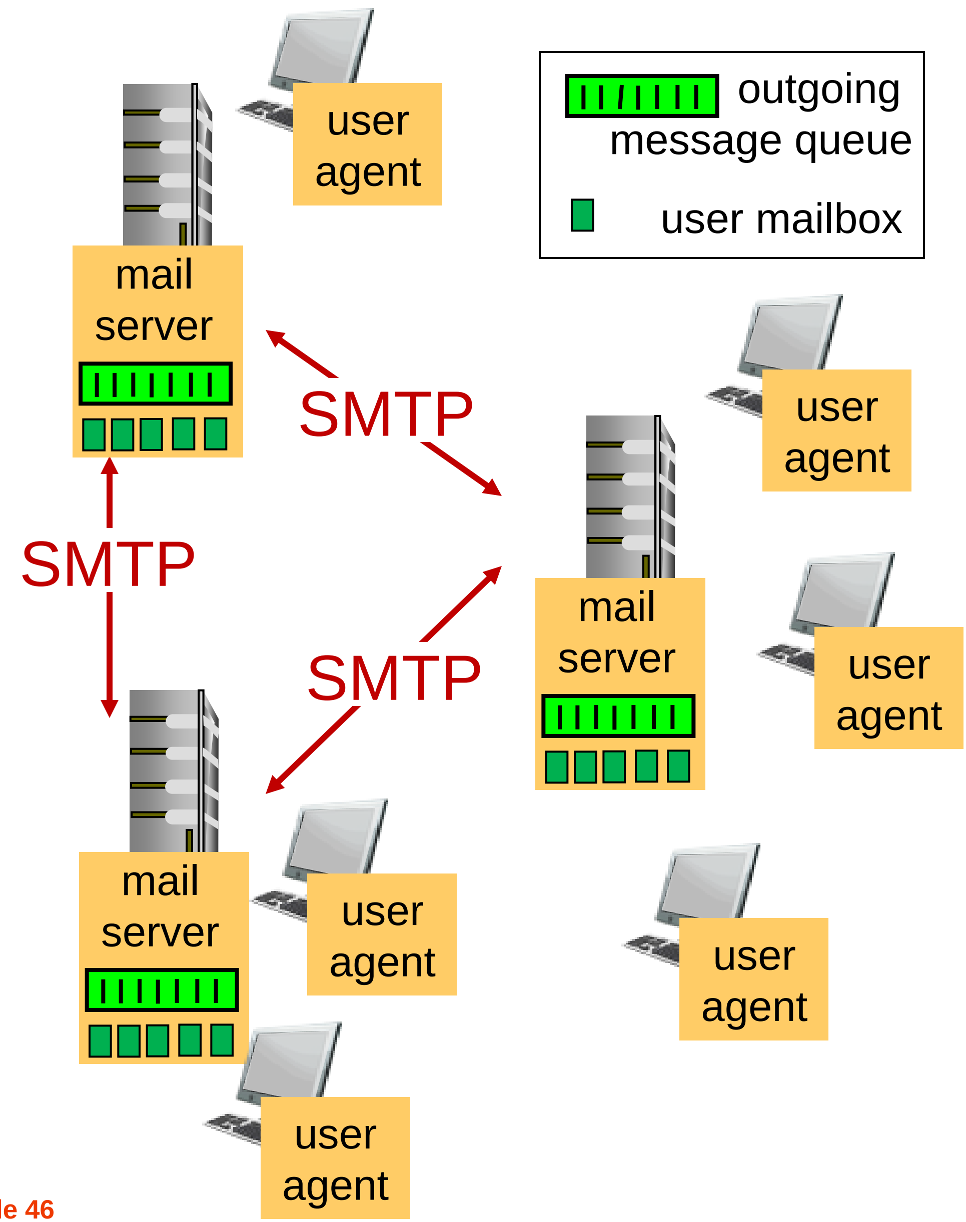


- Server opens a second TCP data connection to transfer another file.
- Control connection: “out of band”
- FTP server maintains “state”: current directory, earlier authentication

Electronic Mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP



<u>User Agent</u>

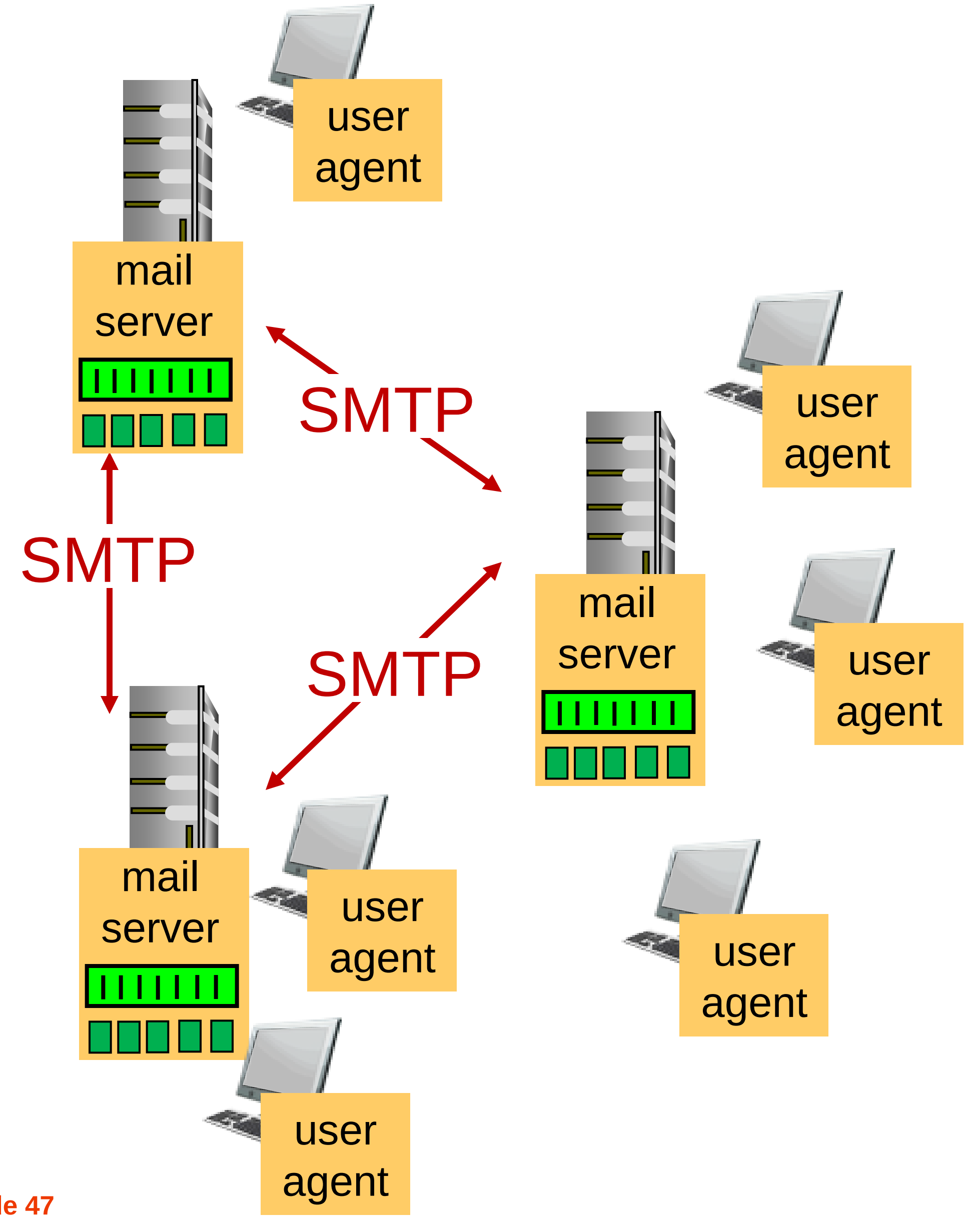
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, elm, Netscape Messenger
- outgoing, incoming messages stored on server

Electronic Mail: mail servers

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Mail Servers

- mailbox contains incoming messages for user
- message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server



Hypertext Transfer Protocol (HTTP)

- HTTP is a protocol that allows you to request files (e.g. html files, images)
  - Did you know that you can send data as well, or delete data?
- Types of messages (HTTP "verbs"): GET, PUT, POST, DELETE, HEAD, OPTIONS, CONNECT, . . .
- Message syntax: HTTP GET is ASCII-based, POST can also use binary data
  - Message start with verb, then the relative address of resources, then protocol version
- Most common message types: GET and POST
- Semantics (interpretation) and rules for processing also defined by RFC

HTTP GET

- HTTP GET is the most common verb: the client requests the server to send a resources identifies by an URL
- Example:

```
GET /demo.asp?name1=value1&name2=value2 HTTP/1.1
host: www.sutd.edu.sg
```

- GET requests are defined to have no side-effect on the server
- Query strings can be sent with request

HTTP POST

- HTTP POST is used to submit data, e.g., forms: the client sends the server some content
- Data can be binary or string, no length limit
- Repeating a POST message will re-post the data (possibly bad)
- Example:

```
POST /demo.asp HTTP/1.1

Host: www.sutd.edu.sg

name1=value1&name2=value2
```

- GET requests are defined to have no side-effect on the server
- Query strings can be sent with request /demo.asp?name1=value1&name2=value2

Idempotence and safe methods

- HTTP uses concepts of idempotence and safe methods
  - Safe methods enable caching and load distribution
  - Idempotence allows to handle lost confirmations by re-sending
- Safe methods will not modify (non-trivial) resources on server
  - Example: GET and HEAD do not change the resource on server
- Idempotent methods may modify resources on the server,
  - But can be executed multiple times without changing outcome
  - Example: duplicate DELETE operations have no additional effect
- POST is not idempotent, multiple POSTs have multiple effects
  - Example: multiple rooms are created for POST /rooms

SOAP

- Simple Object Access Protocol (SOAP)
  - Specific protocol for XML-based data exchange
  - Web service definition language (WSDL) is used to specify available service to client
  - Specific protocol seems to add overhead in many cases
  - Popular in enterprise machine-to-machine communication
- Some IDEs support automatic "learning" of API through WSDL
  - Client-side functions to call APIs are automatically generated
  - *Auto-completion for API calls*

REST

- Representational state transfer (REST)
  - "Architectural style"
  - Data is often exchanged as XML or JSON
  - No WSDL, any interaction is done on nouns /resources, using Create, Read, Update, Delete (CRUD) operations
  - Reduction to nouns and 4 verbs simplifies API
  - Popular for web service APIs (e.g. stackexchange, facebook, imgur)
- Two types of resources in REST
  - Collections: /rooms
    - Container, referencing other things
  - Instances: /rooms/1.502.14
    - Single instance (representing my office)
- Resources are referenced in the HTTP header
  - GET /rooms/1.502.14 HTTP/1.1

What you do not want in your API

- Your API should be focused on resources (nouns) rather than verbs
- Do not build something like this:
  - myserver.com/doStuff
  - myserver.com/doOtherStuff
  - myserver.com/foobar
  - This will not scale, you end up with unstable API

REST Resource naming

- Instead, you will access resources via the HTTP verbs
  - GET, PUT, POST, HEAD, DELETE
- Resources are Collections or Instances
- Examples:
  - example.com/customers/33245/orders
  - example.com/products/66432
  - example.com/customers/33245/orders/8769/lineitems/1
- There could be multiple URLs to access the same data
- GET parameters can also be used, best used for optional arguments
  - `http://bibs.scy-phy.net/bibs?author=tippenhauer&reverse=True`
- More here:

`http://www.restapitutorial.com/lessons/restfulresourcenaming.html`

REST authentication

- Three basic options (without HTTPS client auth):
  - No authentication
  - Identification (claiming an ID)
  - Authentication (verifying an ID)
- Identification: e.g., Google Maps API key for billing
  - Possession of ID value is enough
  - You can distribute this in APP
- Authentication:
  - Basic HTTP authentication in header field (username:password)
  - More advanced authentication tokens (e.g. OAuth)
  - Basic HTTP authentication is often good enough if used over HTTPS

Synthesis: a day in the life of a web request

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

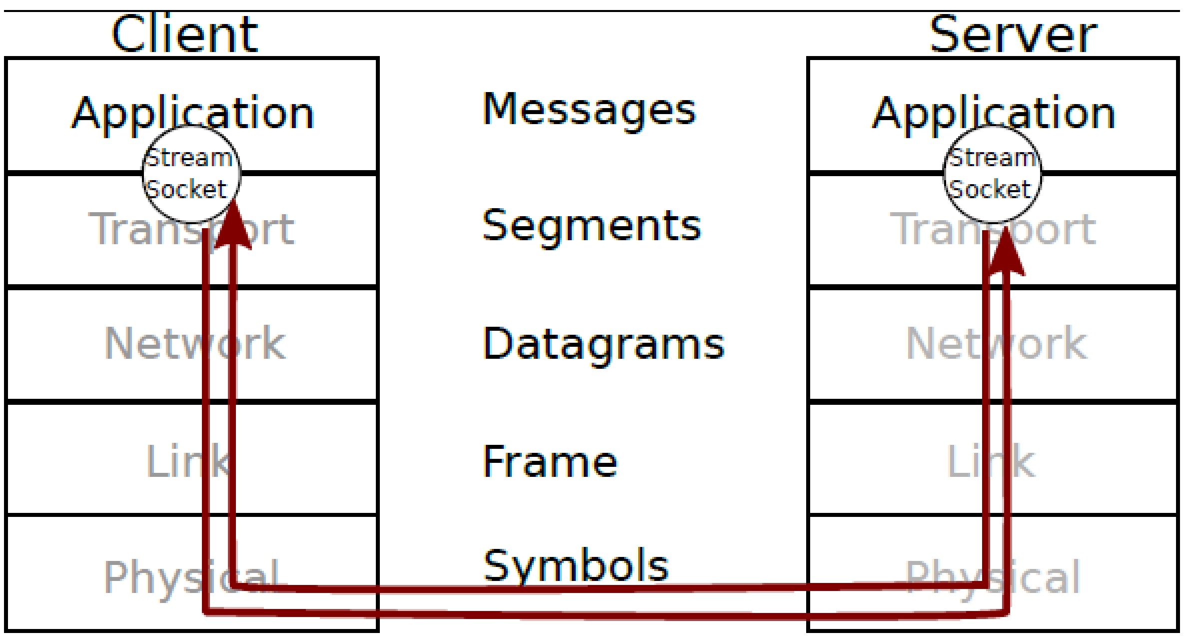
- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

Source: K&R Slide set – Chapter 5

The Transport Layer

Overview

- Transport Layer transmits segments and streams
- Application layer Message is converted to segments
- Segments are then passed on to network layer for transmission
- At the receiver side, segments are re-assembled on Transport layer and passed towards Application layer
- Transport layer is very generic: two protocols (UDP and TCP) are used for almost everything!



Ports and Multiplexing

- To address a remote service, the IP address, port number, and transport layer protocol has to be known
- While the IP address identifies the host, the port number (and protocol) identifies the process that is listening
- A process can listen on multiple ports, but usually at most one process listens to each port1
- The sender can choose an arbitrary source port

Reserved Port Ranges

- Port numbers are 16 bit long (0-65535)
  - IANA (Internet Assigned Numbers Authority) suggests mapping
- Ports from 0-1023 are system ports, most standard protocols.
- On Linux, only root can create sockets using these
  - Example: 22:SSH, 23: telnet, 80: http, 123:NTP
- Ports from 1024-49151 are registered ports for user applications
  - Example: 8080 for http proxy, or web server run as user
- Port from 49152-65535 are to be used as ephemeral ports
  - Automatically assigned by applications
  - On Linux: 32768-65535

UDP

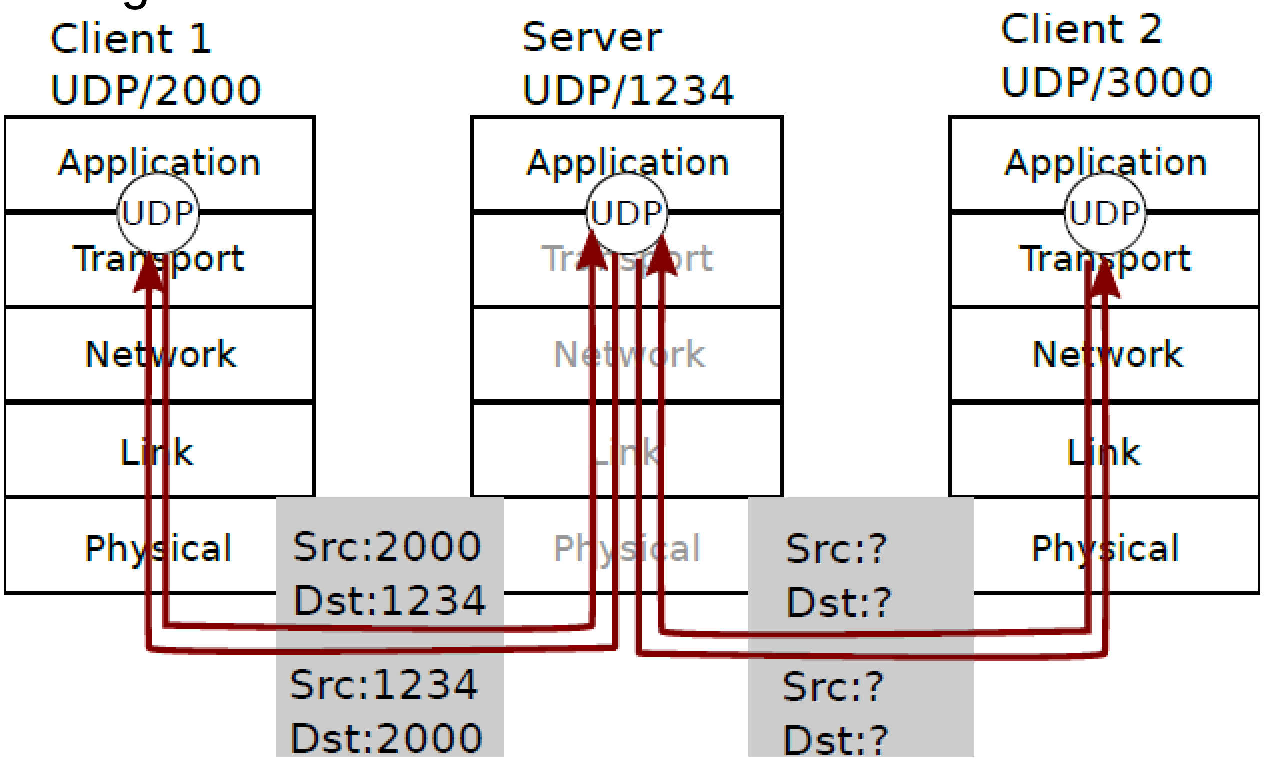
- The User Datagram Protocol (UDP) enables no-frills transport
- It contains the bare minimum: source+destination port, length, checksum
- The UDP header only introduces 8 bytes of overhead to any payload message

<table>
<tr><td>Source Port
16 bit</td><td>Dest Port
16 bit</td></tr>
<tr><td>Fragment length
16 bit</td><td>UDP Checksum
16 bit</td></tr>
<tr><td colspan="2">Data (up to 65,519 Bytes
without 8 Byte header)</td></tr>
</table>

UDP protocol segment header + body

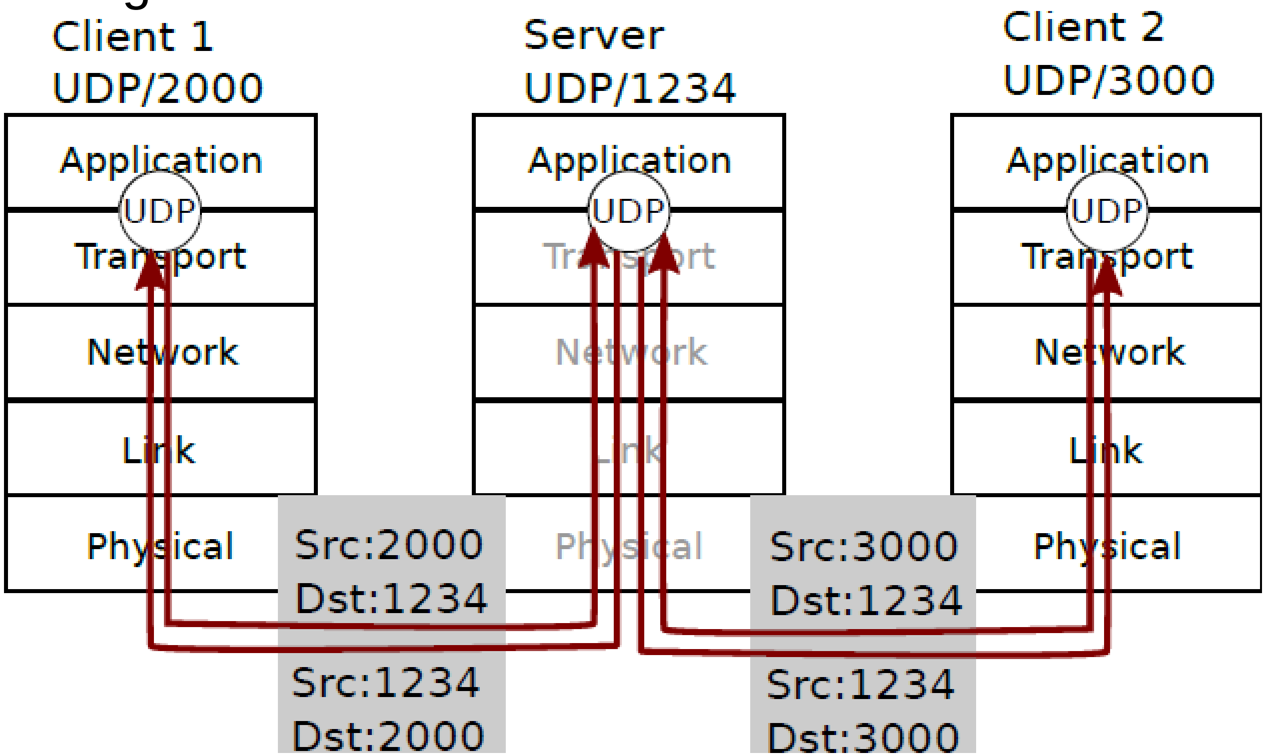
Connection-less Communication

- Consider a UDP server process listening on a socket using port 1234, and handling two clients at the same time.



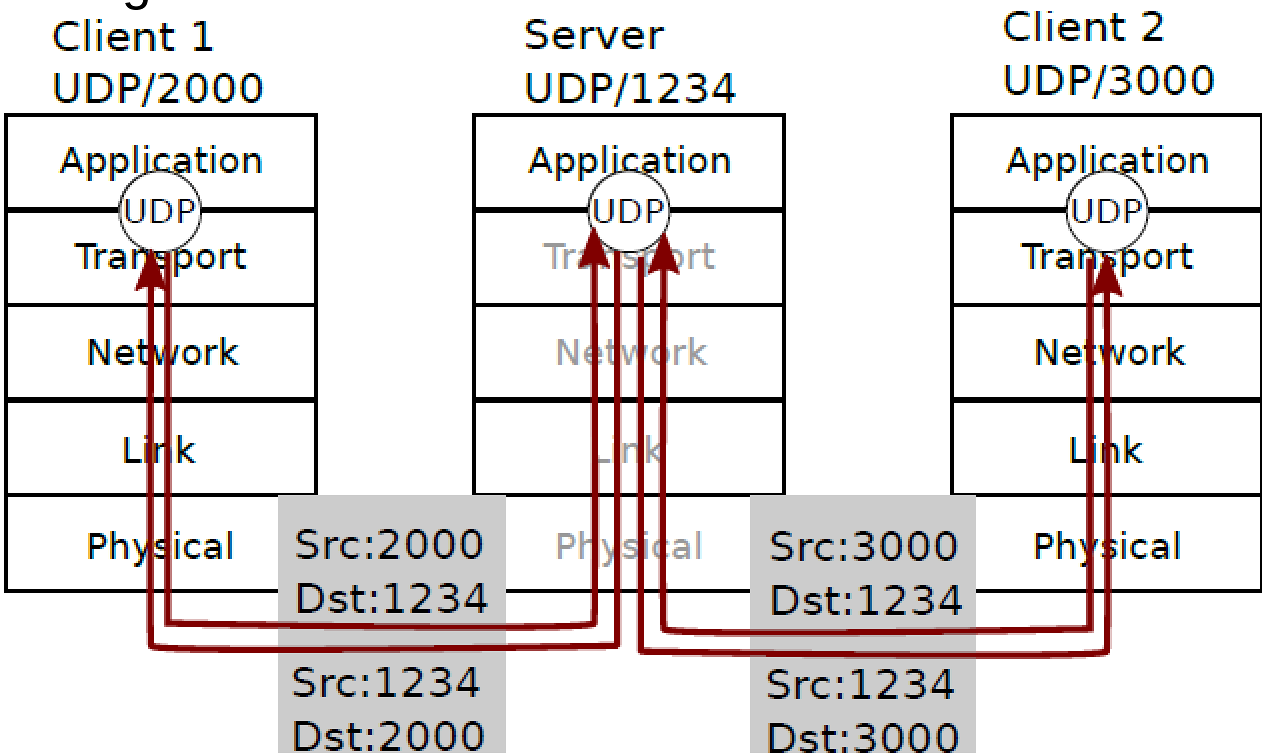
Connection-less Communication

- Consider a UDP server process listening on a socket using port 1234, and handling two clients at the same time.



Connection-less Communication

- Consider a UDP server process listening on a socket using port 1234, and handling two clients at the same time.



- Transport layer passes datagrams to process using dest. port
- UDP Applications can then de-multiplex based on src IP/port

Broadcasting

- Imagine you want to distribute same data to many receivers
  - Example: Status messages, Video streaming
- UDP is a good choice for this, why?

Broadcasting

- Imagine you want to distribute same data to many receivers
  - Example: Status messages, Video streaming
- UDP is a good choice for this, why?

- UDP is connectionless - the sender does not need to perform handshakes or similar with receiver
- Connection-based protocols cannot easily (or at all) be used for broadcasting

UDP Checksums

- UDP Checksums detect transmission errors within a segment
- Checksum is computed as following:
  - UDP segment is interpreted as 16-bit unsigned integers array
    - Including header (apart from checksum field)
  - All values are then added together, a carry is wrapped around
    - Checksum = ones' complement of that sum (XOR with 0xFFFF)
- Example, lets assume we have 10 fields of 16bit/4hex each

```
4500 + 0030 + 4422 + 4000 + 8006 + 0000 + 8c7c + 19ac
+ ae24 + 1e2b = 2BBCF
```

- Wrap around the carry (2): `2+BBCF=BBD1`
- Compute the complement: `BBD1 XOR FFFF=442E`
- Why like this exactly? Was easy to implement 40 years ago

ACKs, NAKs

- How do the ACKs and NAKs look like?
- For the moment, they could just have 33 bit:
  - 16 source port
  - 16 destination port
  - 1 bit ACK/NAK flag (0=acknowledged, 1=not acknowledged)
  - In practice, length is going to be multiple of 8bit, why?
- Do we need anything more? Why is short better?
- Alternative: explicit ACKs
  - Advantages/Disadvantages?

Corruption of ACK / NAK

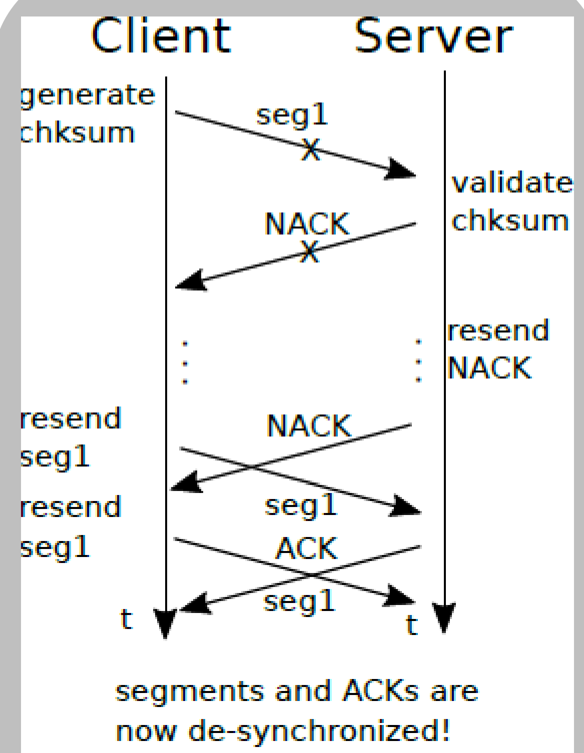
- How to handle corruption of ACK / NAK?

Corruption of ACK / NAK

- How to handle corruption of ACK / NAK?
- Why is retransmission of (N)AK bad?

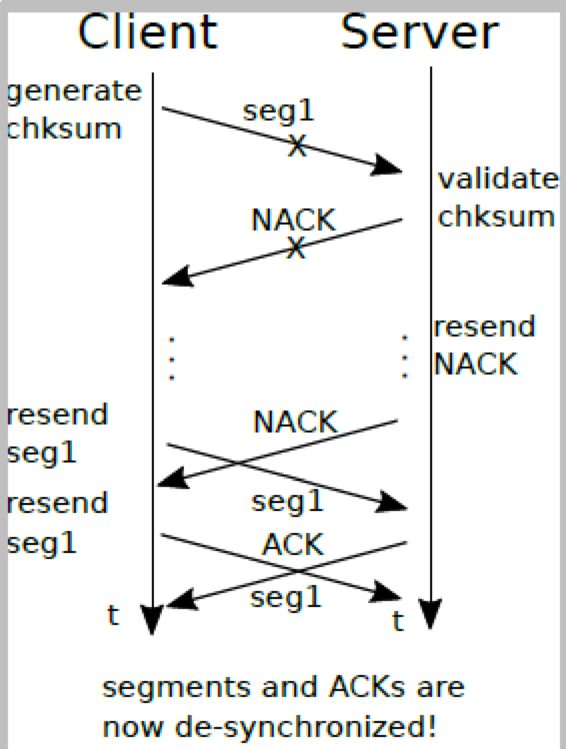
Corruption of ACK / NAK

- How to handle corruption of ACK / NAK?
- Why is retransmission of (N)AK bad?
- Let's assume there is a timeout, after which the sender re-sends ACK/NAK
  - What happens if timeout is too short?



Corruption of ACK / NAK

- How to handle corruption of ACK / NAK?
- Why is retransmission of (N)AK bad?
- Let's assume there is a timeout, after which the sender re-sends ACK/NAK
  - What happens if timeout is too short?
- NAK will be received for next packet

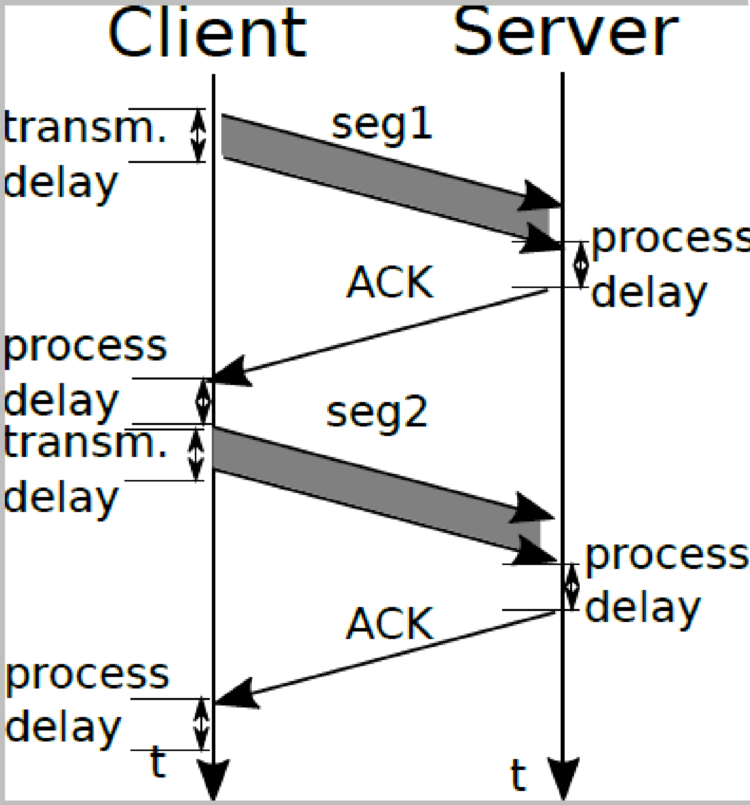


Idle waiting time

- So far, each datagram is ACK'ed
- Transmitting message and getting reply takes time
  - 2x latency + jitter
  - Timeout needs to be at least that time
- This leads to long waiting times
- How does this affect our throughput?

Link Utilization

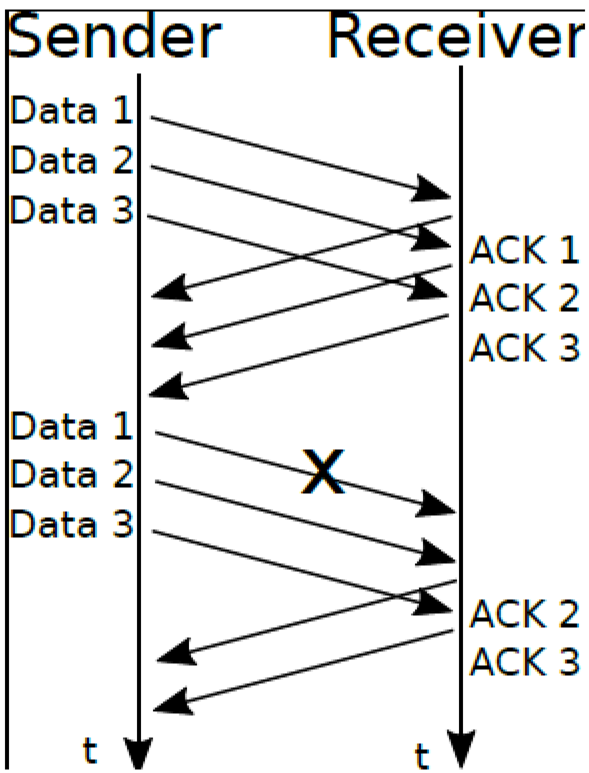
- Lets assume we use ACKs + timeout
- Need ACK before next segment
- Lets assume 1 Gbps connection
  - No queuing delay for now
- Transm. delay of 1kB: $\frac{8000}{10^9} = 8\mu s$
  - ACK trans. delay negligible
- Assume propagation delay 1 ms
- Data/ACK processing delay: 1 ms
- $\rightarrow$ 4ms + 0:008ms per segment
- Effective transmission speed: 250 segments/s or 250kB/s!



RTPv2: Pipelining

Idea to improve throughput:

- Use explicit ACKs, with timeout at sender
- Send multiple segments while waiting for ACKs
- Each segments contains segment ID
- Each ACK contains segment ID
- How big should the segment number be?
  - Trade-off size vs. number of segments in pipeline
  - TCP uses 32 bit (counting bytes, not #datagrams)



RTPv2: Retransmission

- Segment number will be used in ACKs
  - This allows us to re-send ACKs
  - The sender can correlate them to exact segments sent
- If many ACKs are outstanding, sender could start sending less data. . .
- How much overhead introduced through segment IDs

Efficiency/Overhead

- Lets define the effective rate of a protocol
  - R=|m|/|f|
  - With |m| length of payload, |f| length of segment
- UDP has low overhead, only adds 8 Byte
- For a 56k Byte long payload:
  - R=0.999. . .
- Bit errors will reduce efficiency
  - If we cannot recover from one error, R=0!

Re-ordering

- In real life, segments could also arrive in wrong order
- Segment IDs allow re-ordering of segments
- Advantage if IDs can be tied together (sequence)
- This can then also naturally handle re-sent segments

Cumulative ACKs

- Transmission schemes can be enhanced with cumulative ACKs
- Instead of sending individual ACKs, send one ACK for several segments
- For example: If receiver successfully received 10 sequential segments, ACK'ing the last shows that all were received

Go-Back-N vs Selective repeat

Two approaches to transmission pipelining

- Selective Repeat
  - Up to n packets can be un-ACK'ed by sender
  - Each packet is ACK'ed individually (also out-of-order ones, are buffered)
  - Each sent packet has its own timer for timeout
- Go-Back-N
  - Sender can have up to n un-ACK'ed segments in pipeline
  - Receiver only sends cumulative ACK
    - Out-of-order segments are not ACK'ed
  - Sender has timeout for oldest un-ACK'ed segments
    - When that timer runs out, all un-ACK'ed segments are re-transmitted

Window Size

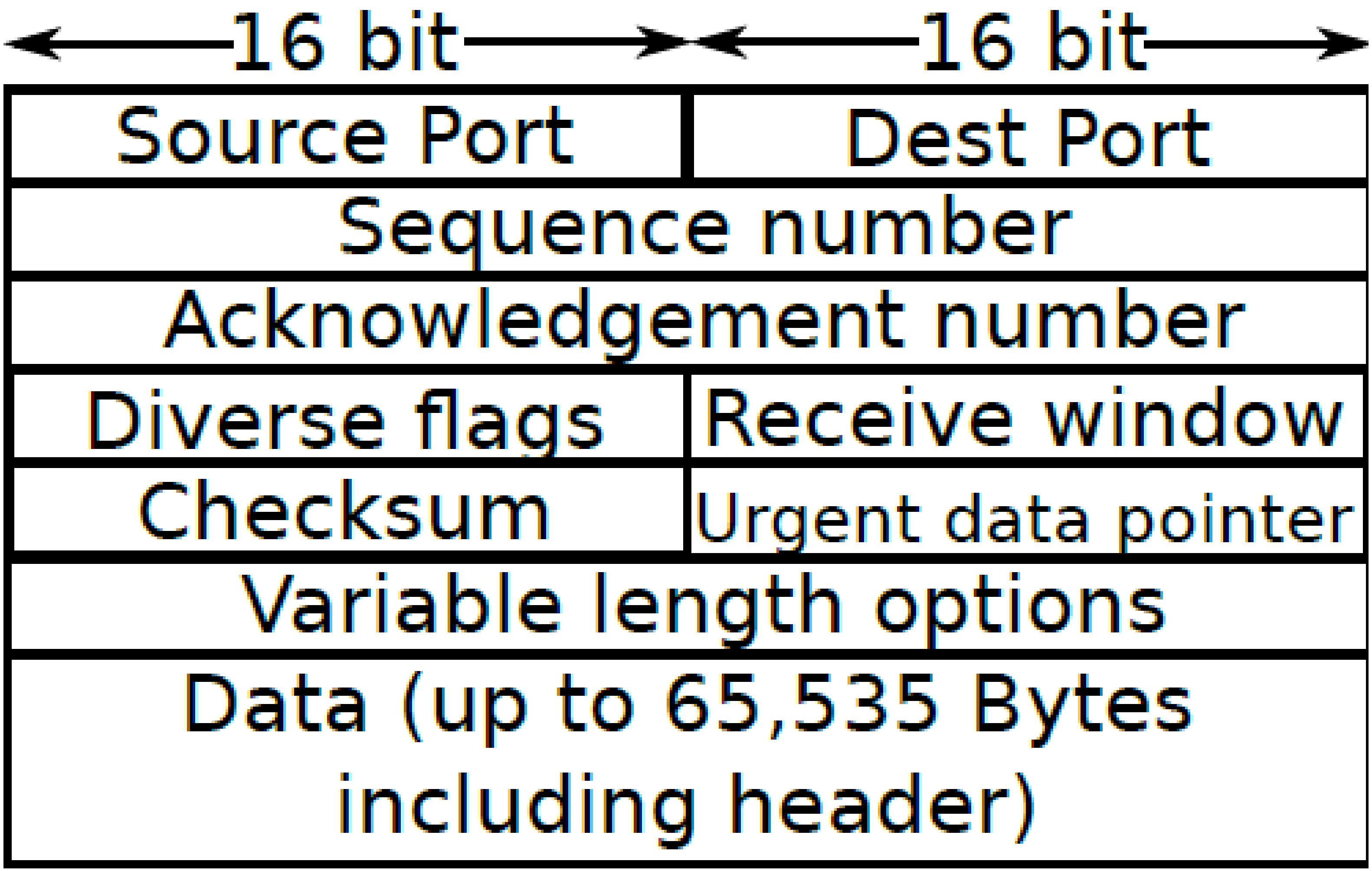
- How big should the window size n be?
- A transmission window size of 2 already helps, why?
    - Time spent waiting for ACK is almost cut by two compared to no pipeline case
- Other suggestions on the window size?
    - What is limiting us?
    - Buffer sizes at the receiver and sender.

The Transmission Control Protocol

TCP Overview

- The Transport Control Protocol (TCP) is the most commonly used transport protocol
- Streaming protocol, as it provides a continuous connection between single sender and receiver
  - Initial handshake is used to set up connection
- Pipelined cumulative ACKs, based on flow and congestion control
  - Sender adapts rate to buffer size of receiver

TCP header structure



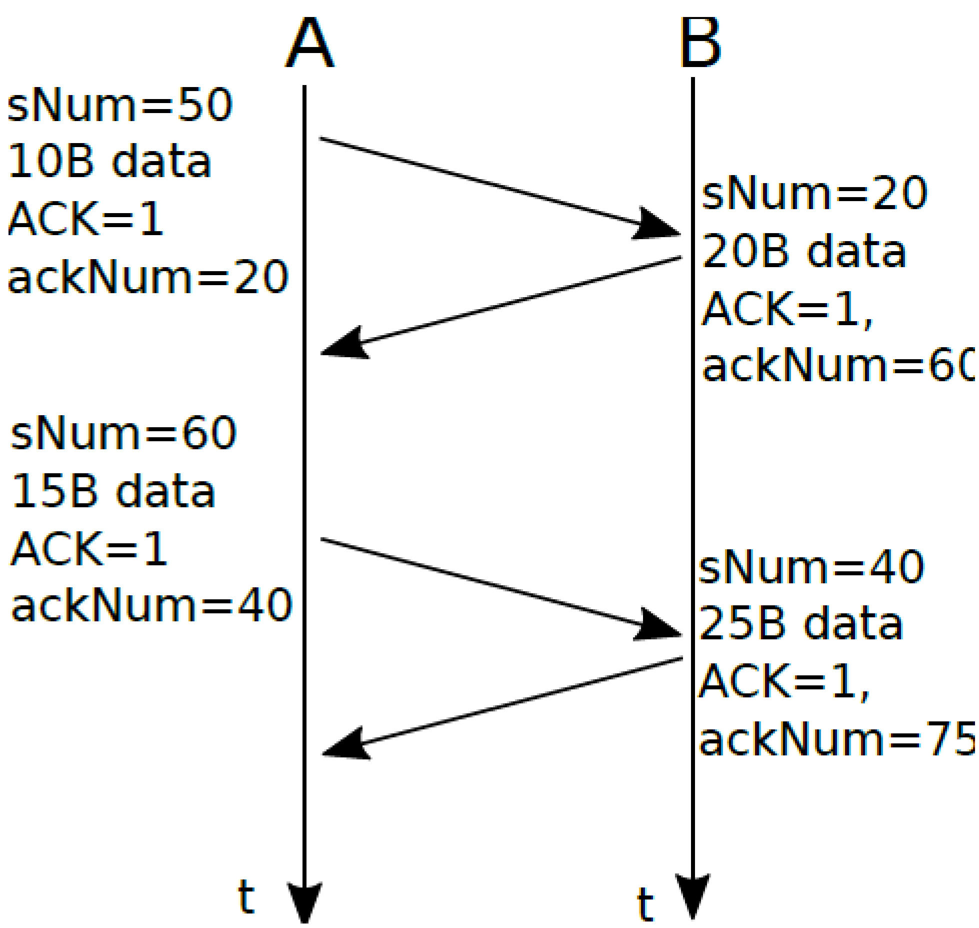
Flags in the TCP header

- The 16 *diverse flags* field contains:
  - Header length (due to variable length options)
  - Some unused bits
  - Urgent flag (generally not used)
  - ACK flag (consider ack number acknowledged)
  - Push flag (generally not used)
  - RST, SYN, FIN flags: for connection management, e.g. handshaking

Acknowledgments in TCP

- Segment number is incremented by Byte size of data payload
  - ID numbers can be large - 32 bit ~ 4GByte until overflow
- Acknowledgement number is referring to expected next segment
- ACKs can be integrated in normal segments
- TCP header has explicit acknowledgement number field and ACK flag
- ACK is cumulative - i.e., receiver acknowledges receipt of all previous payload
- ACK transmission can be delayed by Receiver, wait for 500ms if next packet comes

TCP Full Duplex ACK



Bi-directional (full duplex) ACK and data exchange

TCP timeout

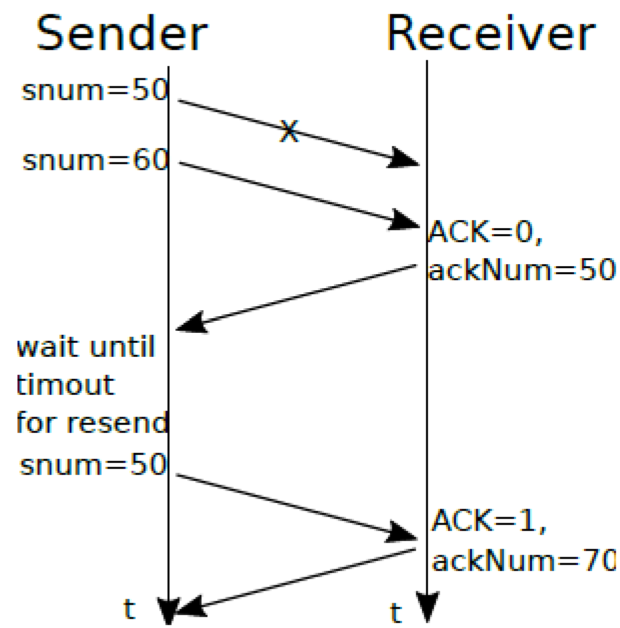
- The TCP timeout for ACKs is based on continuous measurements
- EstimatedRTT= (1 - α) * EstimatedRTT + α * SampleRTT
- Choice of α determines how quickly RTT window adapts
  - Typical choice: α = 0:125
- Safety margin is then added with second variable
  - DevRTT= (1 - β) * DevRTT + β * | SampleRTT - EstimatedRTT |
- TCP timeout is typically larger than 500*ms*

TCP timeout handling

- Timer is counting for oldest unACK'ed segment
- If ACK is received: start timer for next segment
- If ACK is not received in time: retransmit old segment, restart timer
  - If receiver has following segments in receive buffer, he will ACK that content as well

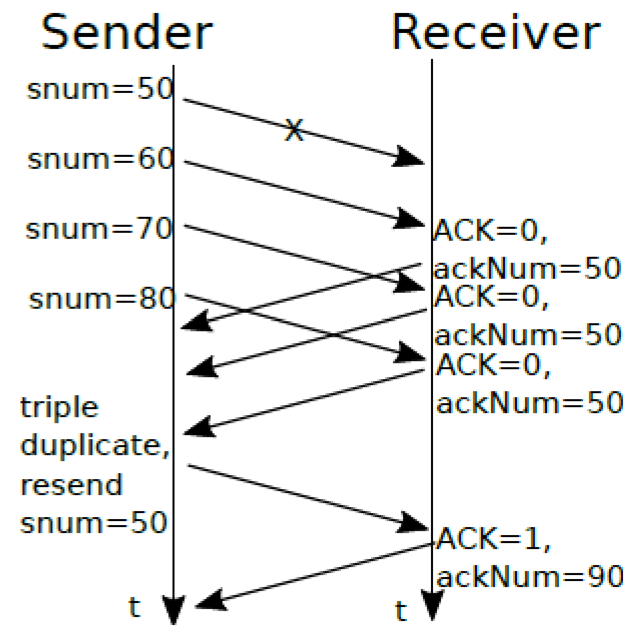
Duplicate ACK

- If receiver receives segment with higher number than expected, she sends duplicate ACK
  - Send a TCP message with the ack number for the segment she is waiting for
  - Without setting the ACK flag
- This tells the sender that the receiver is still waiting for some content



Triple Duplicate ACKs

- If sender receives three of such duplicate ACKs, she will not wait for ACK any longer
  - She will immediately re-send the requested segment
  - Receiver will then send acknowledgement that includes out-of-order segments with higher number



TCP Flow control

- TCP header contains receive window field
- This specifies the free buffer size on the receiving end
- Sender should make sure that size of un-ACK'ed segments is smaller than this buffer
- This is called flow control
- Example: slow receiver, but fast sender
  - E.g. embedded device downloading data from server
  - Embedded device might be slow in processing buffer content
  - You set the maximal buffer size in your Python code, typically 4096 Bytes

Connection Establishment

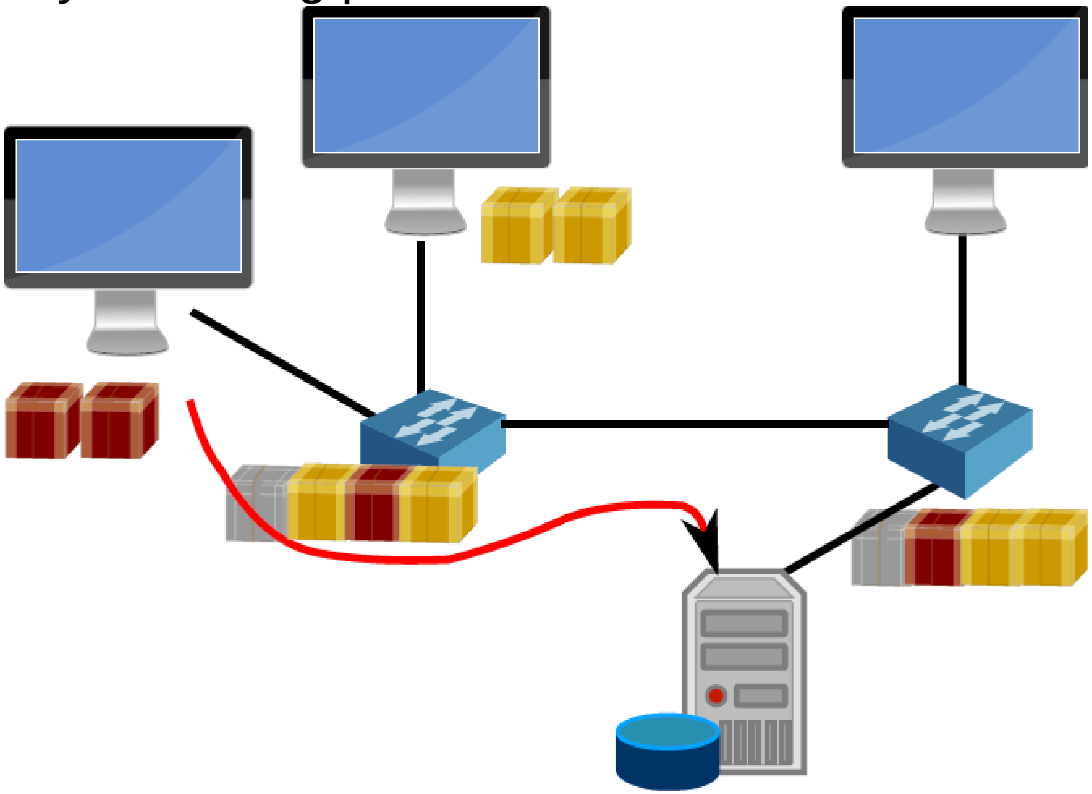
- TCP uses a 3 message handshake to establish connection
- Client starts with some random sequence number s_c, and sets the SYN bit
- Server chooses its own random sequence number s_s, and sets the SYN and ACK bit
  - Also increments the client's sequence number by one: $a_s = s_c + 1$
- Client responds with ACK of server's sequence number, and increased acknowledge number $a_c = s_s + 1$?

Connection Termination

- 2 Messages required for connection termination
- Either side sends segment with FIN set to 1
- Other side replies with ACK (and possibly FIN) bit set
- Until both sides sent FIN, connection is still alive
  - This enables the receiver of FIN to finish his transmission of data

Congestion

- Congestion is a network layer problem:
  - Network link cannot transmit packets fast enough
  - Forwarding routers will start to fill up transmission buffers
- Buffers at routers are finite
  - So eventually, incoming packets at router cannot be stored and are discarded

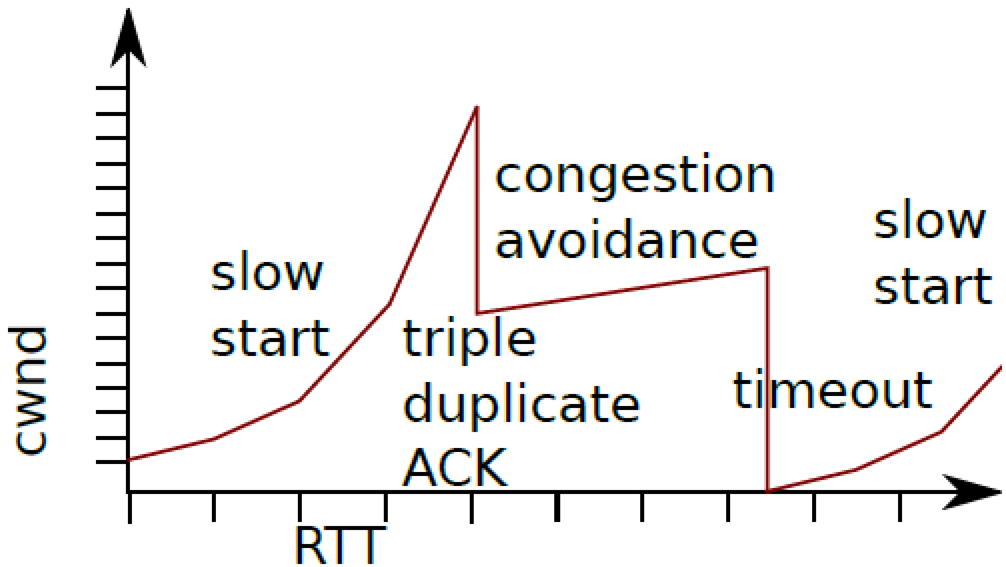


TCP Congestion control

- Approach: maximize transmission rate in fair way
  - Flow control ensures that received can handle data
  - Congestion control ensures that intermediate links are not overloaded
- Two phases
  - *Slow start* mode at beginning of TCP connection
  - *Congestion avoidance* mode after first dropped ACK
- In both phases the *congestion window* (cwnd) is adapted

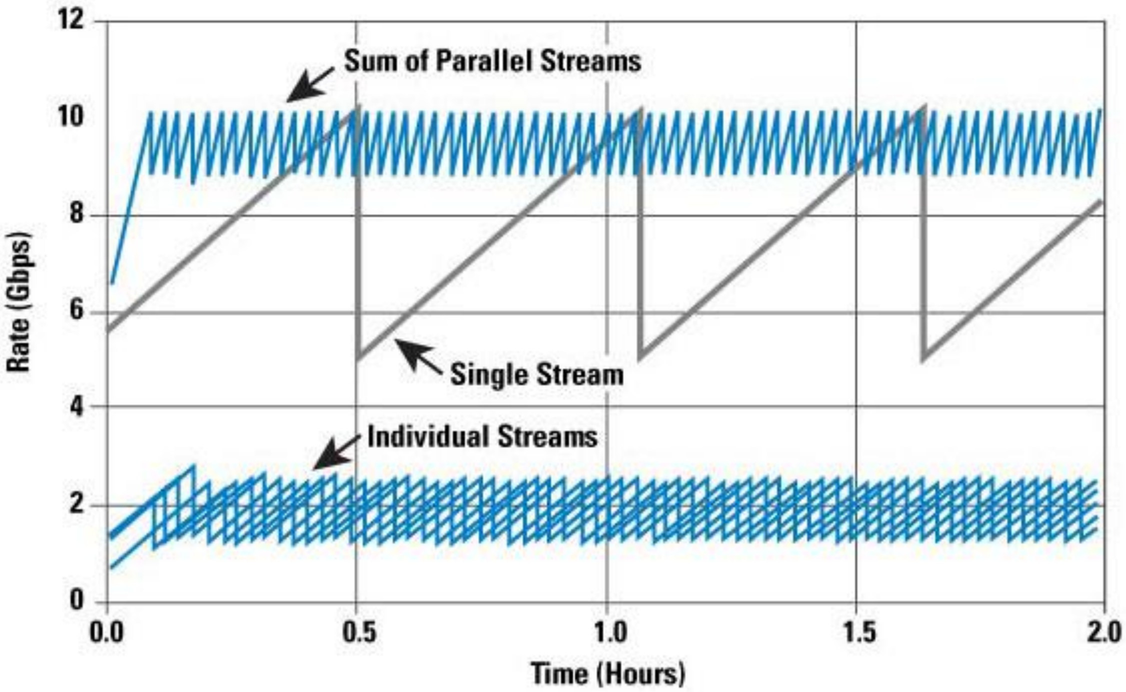
TCP Slow start

- At connection start: *slow* start phase, cwnd=1
- cwnd is then *increased* by one for each ACK until loss occurs
- When loss is detected
  - By triple-duplicate-ACK: cwnd is cut by half and *congestion avoidance mode*
  - By timeout: start in slow start phase from 1 again



TCP Congestion avoidance mode

- Increases rate additively (+1 per RTT time) until loss occurs
- again
- This will result in a saw-tooth pattern for the rate



TCP and Fairness

- With multiple parallel TCP streams, every stream gets roughly the same share of available bandwidth
- Every stream is increasing transmission window with roughly same speed (+1/RTT)
- Every stream backs off by cutting transmission window in half
- Hosts can get a somewhat unfair advantage by creating *n* parallel TCP connections
  - They will get ~ *n* times the bandwidth they would get otherwise

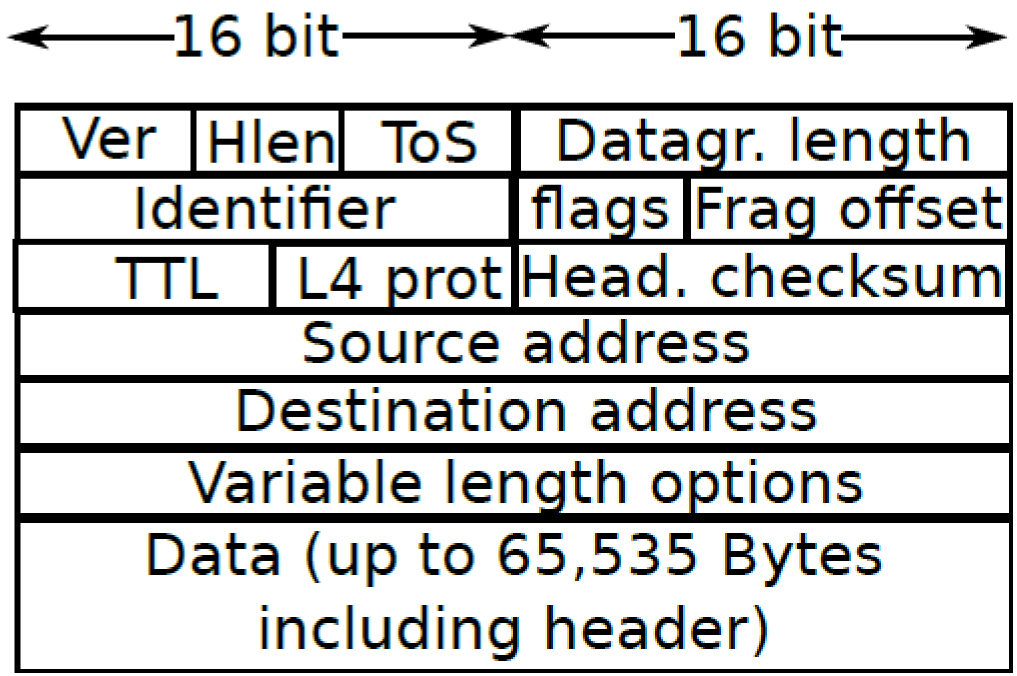
Example: parallel TCP connections by your browser to a website

The Network Layer

The Network Layer

- The Network layer provides logical connection between hosts
  - Protocols: IP, ICMP, IGMP
  - Addressing is based on IP addresses
- Remember:
  - App layer: connection between applications
  - Transport layer: connection between processes
  - Link layer: single-hop connections between interfaces
- The network layer is domain of routers and Internet backbone
- Main services of Network layer:
  - Routing: find best path for datagram from hostA to hostB
  - Forwarding: move datagrams from input to output interface

IPv4 Format



The IP packet format

IP fragmentation

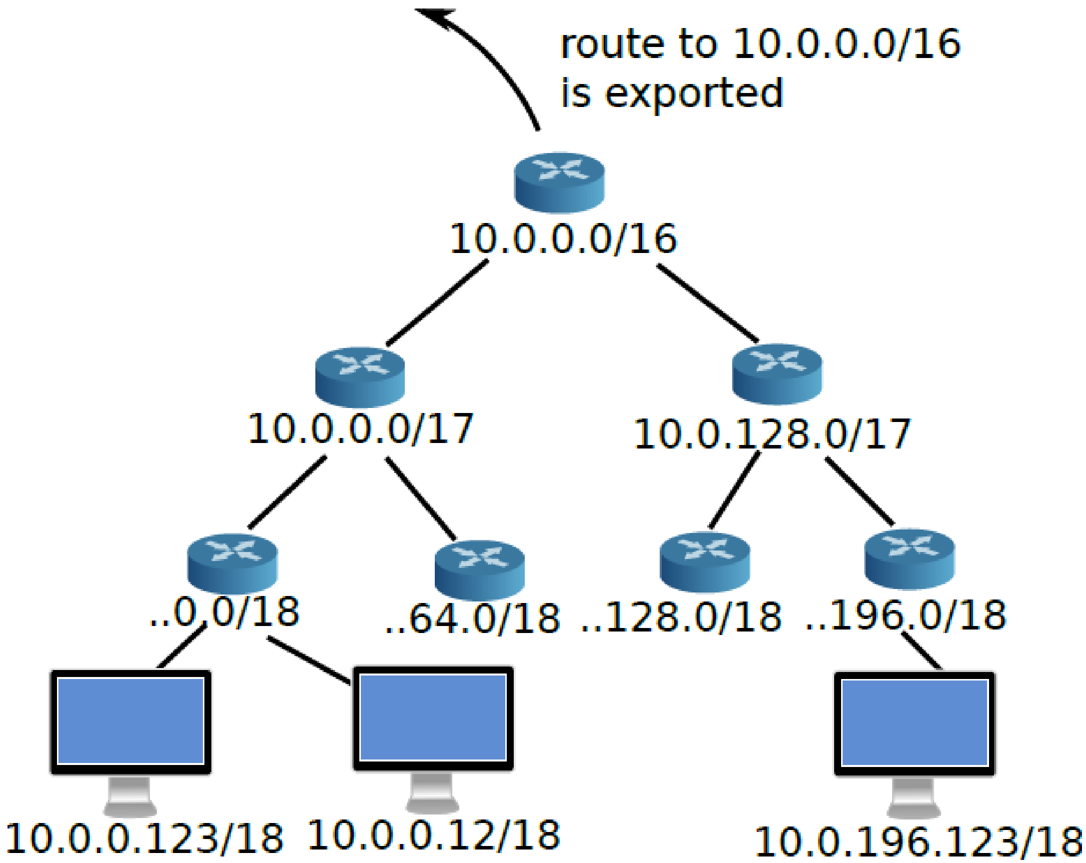
- First layer that single links are considered
- Links can have different maximal transmission unit/size (MTU)
- Transport layer segments might have to be fragmented
- Segments are split into *fragments*, re-assembled at receiver
- Identifier, flags, *Fragment* offset part of IP header used to keep track of network layer fragmentation of single datagram
- MTU is usually lower in link layer (~1.5kB)

TTL

- The time-to-live (TTL) field is used to prevent infinite loops
- Its maximum initial value is 255, typical value is 64
- Every hop/router reduces the TTL by 1
- If TTL reaches 0, the packet is discarded
- This prevents a packet from being caught in infinite routing loops

Subnet Masks and CIDR

- Recap: What are subnetworks used for again?
- From client perspective: *know when to send via GW*
- From routing perspective: enable hierarchical routing



How are routing decisions made?

- Routing decisions have to be taken fast
  - One for every datagram, millions of datagrams per second
- Basic approach: Routing table lookup
  - Each row contains target CIDR network and interface
  - Target IP is matched to one or more rows
  - Most specific entry is used

Destination	Iface
0.0.0.0/0	eth4
101.4.0.0/16	eth1
11.2.5.192/28	eth4
11.2.0.0/20	eth3
101.4.5.192/28	eth2
55.14.85.0/24	eth4
111.61.51.0/24	eth4
171.14.5.0/24	eth2
206.76.5.0/24	eth3
119.9.1.192/30	eth4
12.45.51.0/24	eth2

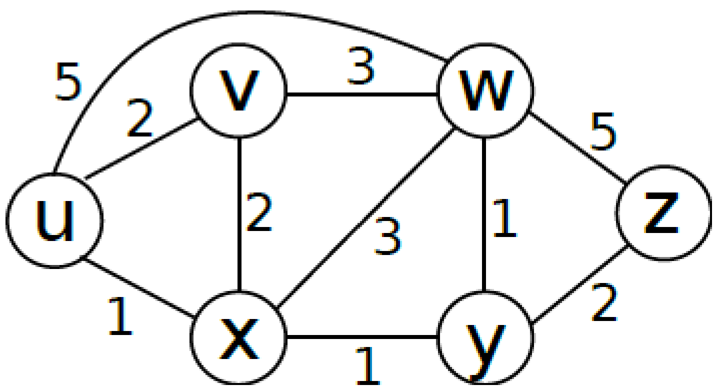
An example routing table

Creation of the routing tables

- So, where do these routing tables come from?
- They are not created on-the-fly:
  - Some routes are learned from other routers
  - Some routes are manually set by administrators
  - Existing routes are often updated over time
- They are exchanged via protocols such as
  - BGP (between different autonomous systems)
  - OSPF (within the same autonomous system)
- More on algorithms and protocols used later

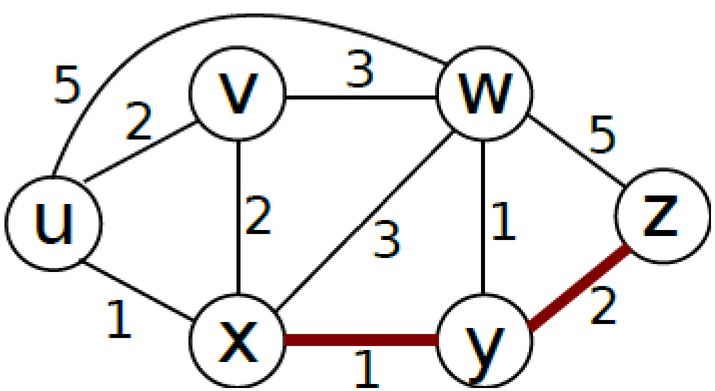
Graphs

- Quick recap on graph data structures:
  - G={N,E} (N=Nodes and E=Edges)
- When using in a network context:
  - Nodes are simply our hosts (routers, PCs, servers)
  - Network links are undirected edges with weights
  - E={(x,y),(x,z),(y,z)}
  - Weights are costs of that link, a label to each tuple
- Graphs can also be used to represent more abstract links
  - TCP streams
  - Overlay networks



Costs of links and paths

- A link is an edge in the graph, represented by 2-tuple
- A path is a sequence of one or more edges, an n-tuple
- A cost function for a network will yield cost of link or path
  - Cost can be related to delay, inversely to bandwidth, or congestion
- Usually, the cost of links can be added up:
  - If c(x,y)=1 and c(y,z)=2, then c(x,z)=3 (if this is shortest path)
- The routing algorithms will try to find a minimal cost path between two hosts



Routing Algorithms

Two classes of algorithms

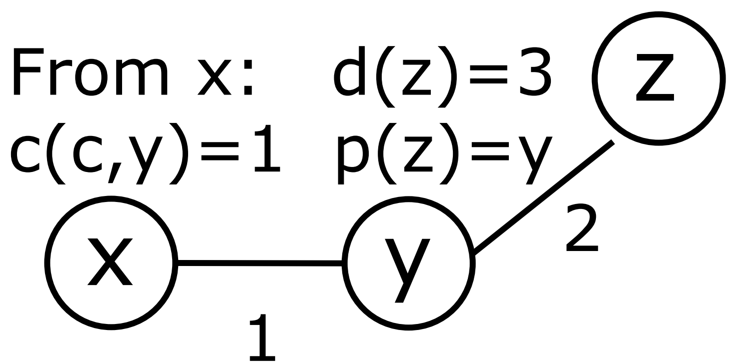
- Global view: shared global info on topology state, cost
  - called link state algorithms
- Local view: every host has own view, must talk to neighbors
  - Only topology and cost to neighbors is known
  - Algorithms typically converge over time
  - called distance vector algorithms
- path vector algorithms mix both:
  - Routers announce routes throughout the network
  - Intermediate routers forward shortest routes and add themselves
- The system can be considered static or dynamic
  - In the latter case, partial updates of routes should be possible

The shortest path problem

- The single-pair shortest path problem :
- Find the shortest path from x to y
  - Edges have weights
  - All nodes are connected to graph
- Related problems:
  - Single source shortest path (from x to each other node)
  - Single destination shortest path (to y from each other node)
  - All-pairs shortest path: single-pair for all possible pairs
- Cost for single-pair: O(E + V log V) (Dijkstra's algorithm)

Dijkstra's algorithm setup

- Link state algorithm, complete network has to be known
- Iterative algorithm that builds a set of shortest paths
- *d(v)* is the currently known cost of the shortest path from *x* to *v*
- *p*(*v*) is the predecessor of v on the currently shortest path from *x* to *v*
- *N':* set of nodes with known absolute shortest path



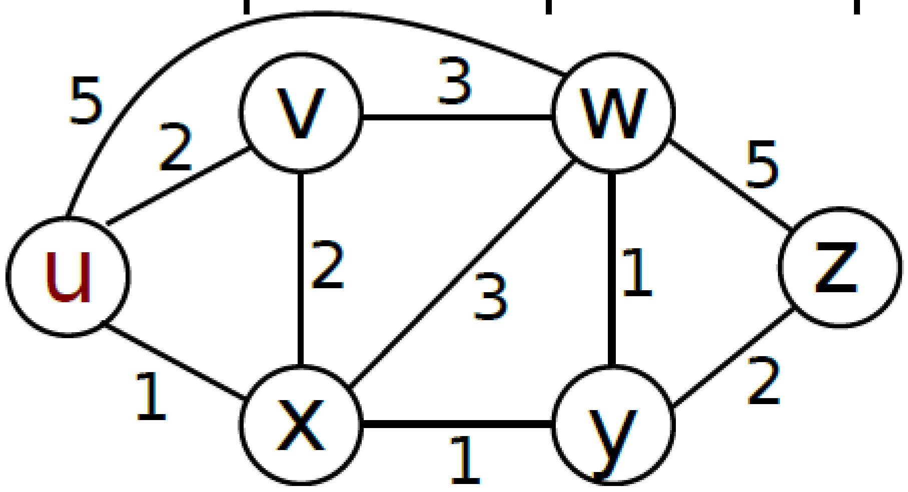
Dijkstra's algorithm

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

 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'
```

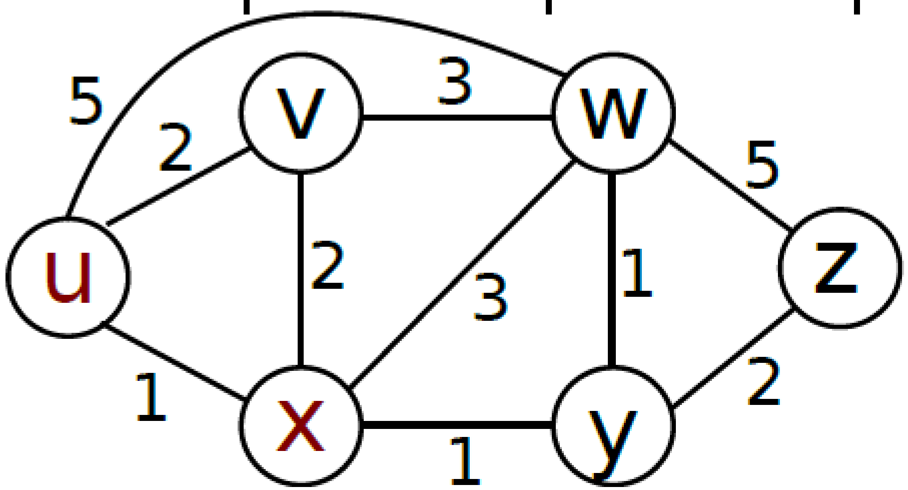
Dijkstra example

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	infty	infty



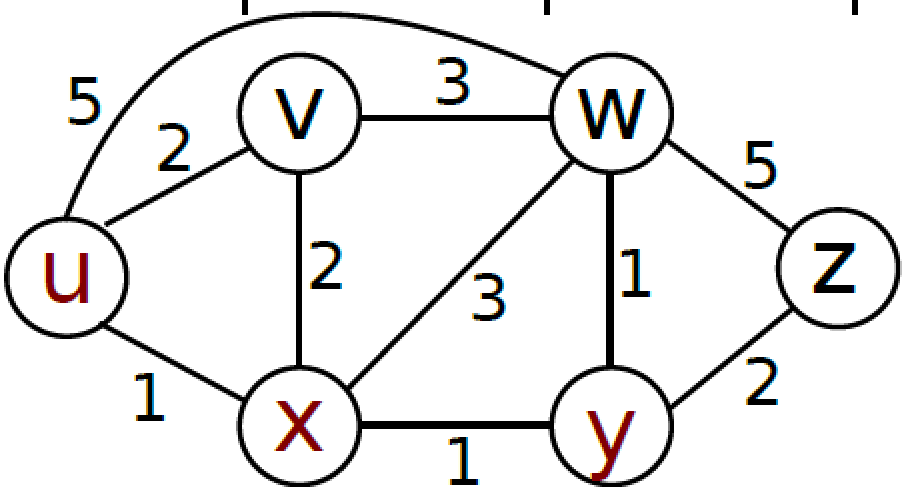
Dijkstra example

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	infty	infty
1	ux	2,u	4,x		2,x	infty



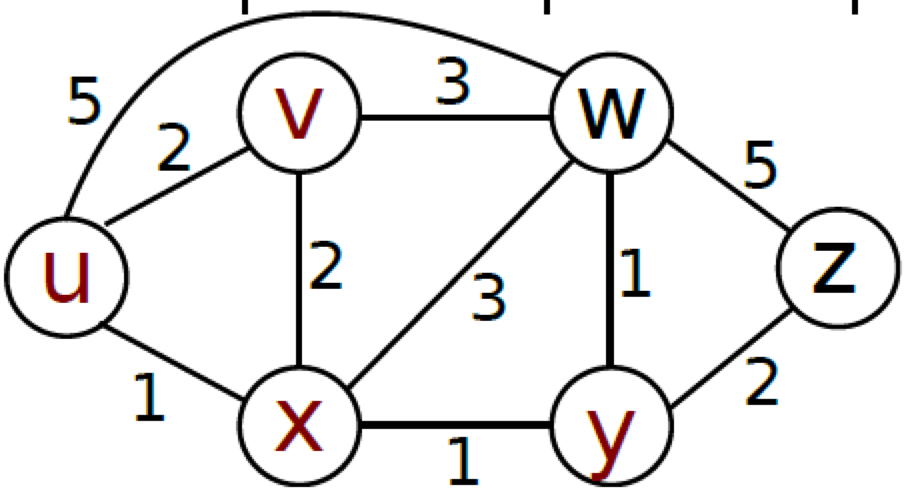
Dijkstra example

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	infty	infty
1	ux	2,u	4,x		2,x	infty
2	uxy	2,u	3,y			4,y



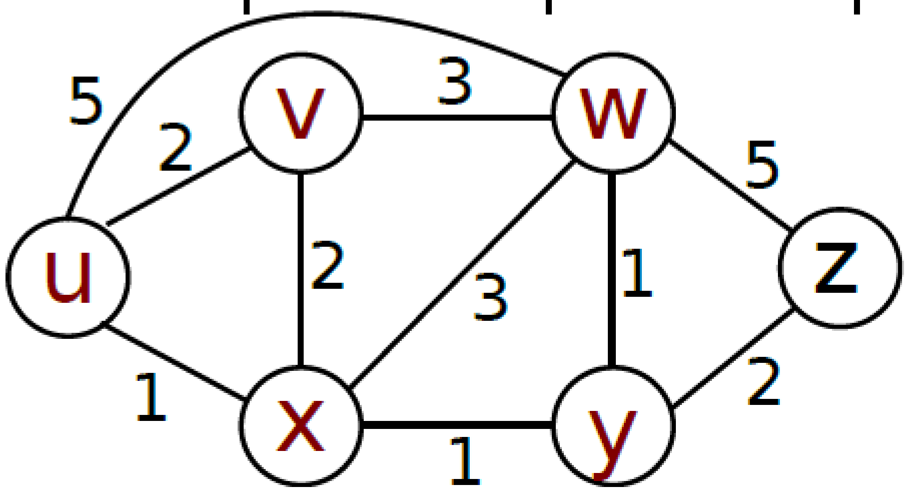
Dijkstra example

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	infty	infty
1	ux	2,u	4,x		2,x	infty
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y



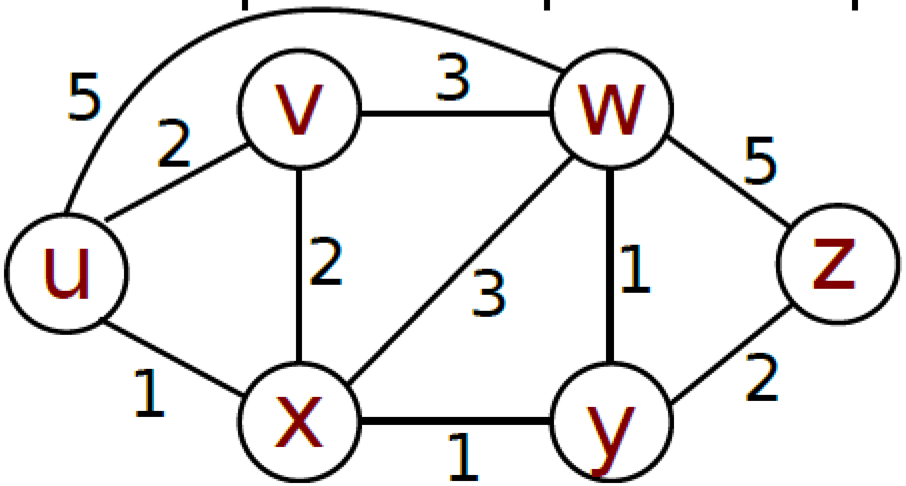
Dijkstra example

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	infty	infty
1	ux	2,u	4,x		2,x	infty
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y



Dijkstra 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	infty	infty
1	ux	2,u	4,x		2,x	infty
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Link state routing

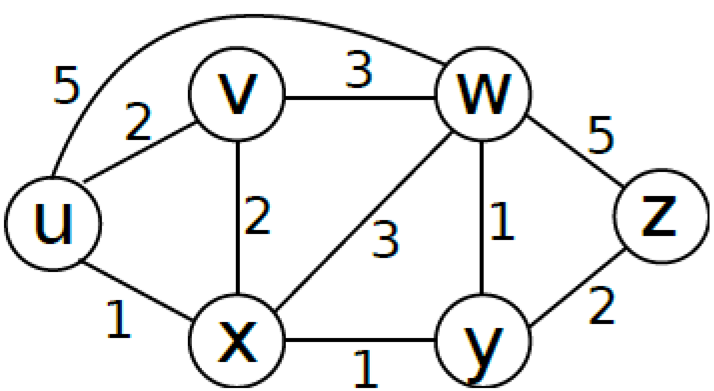
- Dijkstra’s algorithm was an example for link state algorithms
- We start from one node, and iteratively include knowledge of all other nodes
- Messages that need to be exchanged: O(|E|*|V|)
  - Each router sends cost information for all neighbors
- This does not scale on the Internet
  - |E| and |V| is huge -> product is bigger
  - We would have to run Dijkstra’s algorithm for each router
- We need an algorithm that builds on local information

Bellman-Ford

- Bellman-Ford is a dynamic programming solution
  - You split the big problem into smaller problems
  - Find optimal solutions for small problems
  - Use these to find optimal solution for big problem
- With $d_x(y)$ = cost of least-cost path from *x* to *y*
- Bellman-Ford solves:
  - $d_x(y) = \min_v \{ c(x; v) + d_v(y) \}$
  - We ask all our neighbors for their cost to y (i.e., $d_v(y)$)
  - And add our link cost to them
  - The neighbor with lowest sum of both will be selected as next hop

Example Bellman-Ford

- *u* has neighbors *x*, *v*, *w* and knows
- $c(u, x) = 1$; $c(u, v) = 2$; $c(u, w) = 5$
- *u* has the distance vectors of his neighbors:
  - $d_v (z) = 5$
  - $d_w (z) = 3$
  - $d_x (z) = 3$
- *u* selects lowest sum of $c(u, ?) + d_?(z)$ which is
- $c(u, x) + d_x (u) = 4$, so that is shortest path)



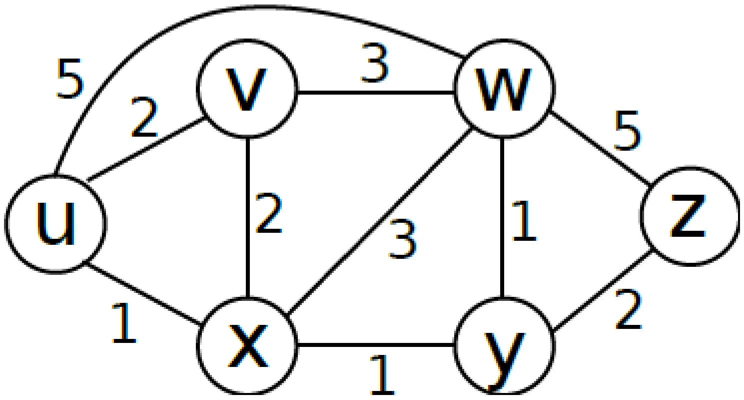
Routing based on Bellman-Ford

- Bellman-Ford can be used to build a routing protocol
- Each node is a router with local distance vector
- Neighbors exchange distance vectors (with n entries for size n graph)
  - Every node with k neighbors will hold $k * n$ entries locally (+own)
- If changes occur in tables, receiver will re-compute own table
  - Send out own vector if changed
- Under natural conditions, this will converge to the *actual* least cost paths
  - Proof omitted here, see *Algorithms* slides

Example on table updates

Distance vectors:

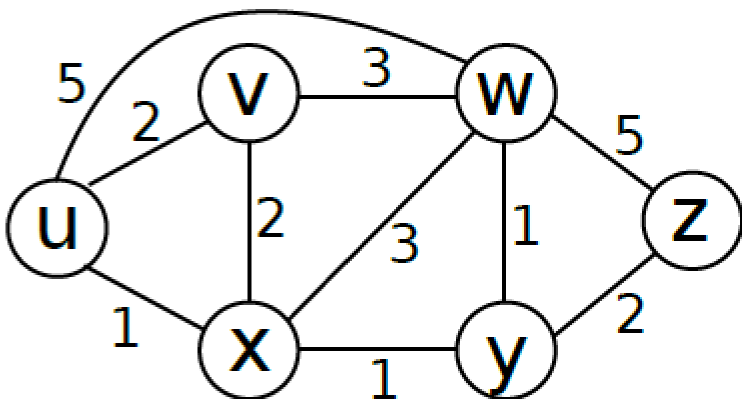
from \ to	u	v	x	y	w	z
u	0	2	1	-	5	-
v	2	0	2	-	3	-
x	1	2	0	1	3	-
y	-	-	1	0	1	2
w	5	3	3	1	0	5
z	-	-	-	2	5	0



Example on table updates

What is on a path for u and y?

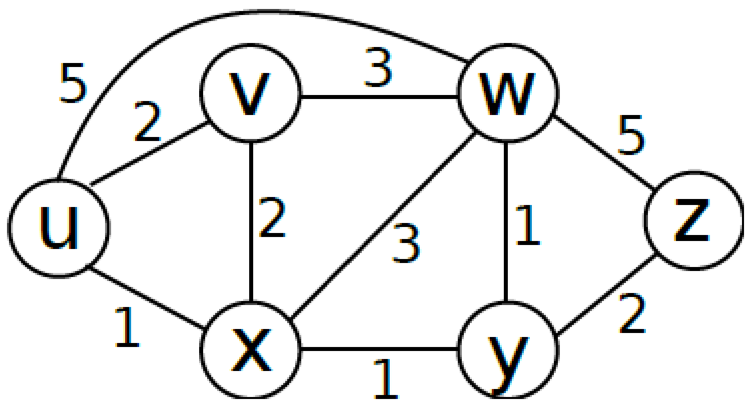
from \ to	u	v	x	y	w	z
u	0	2	1	?	5	-
v	2	0	2	-	3	-
x	1	2	0	1	3	-
y	-	-	1	0	1	2
w	5	3	3	1	0	5
z	-	-	-	2	5	0



Example on table updates

u (and y) compute new path

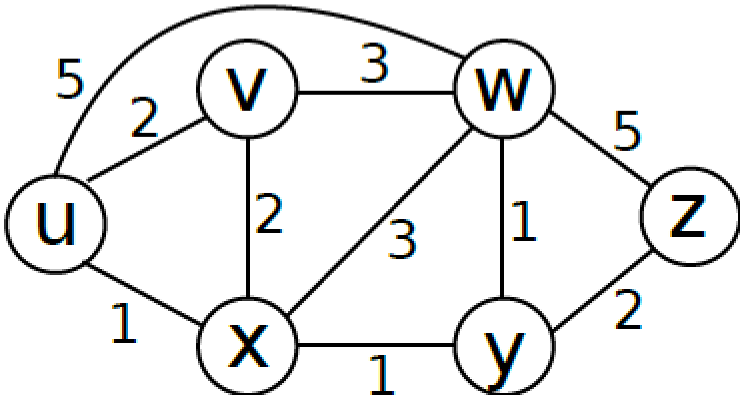
from \ to	u	v	x	y	w	z
u	0	2	1	2	5	-
v	2	0	2	-	3	-
x	1	2	0	1	3	-
y	2	-	1	0	1	2
w	5	3	3	1	0	5
z	-	-	-	2	5	0



Example on table updates

all distance vectors converge

from \ to	u	v	x	y	w	z
u	0	2	1	2	4	4
v	2	0	2	3	3	5
x	1	2	0	1	2	3
y	2	3	1	0	1	2
w	4	3	2	1	0	3
z	4	5	3	2	3	0

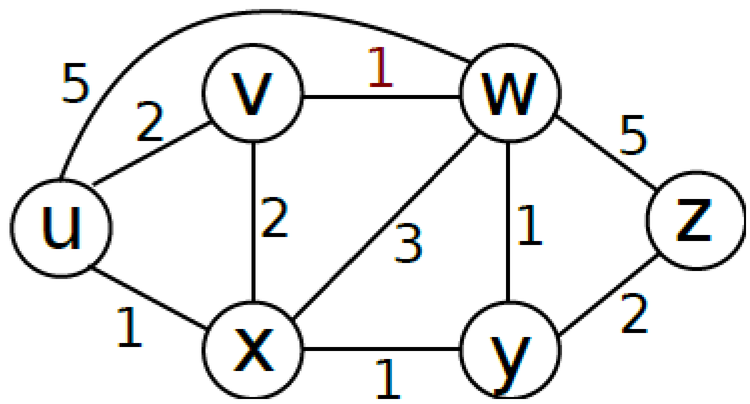


Example on table updates

What if vw cost changes to 1?

to

from	u	v	x	y	w	z
u	0	2	1	2	4	4
v	2	0	2	3	$\boxed{1}$	5
x	1	2	0	1	2	3
y	2	3	1	0	1	2
w	4	$\boxed{1}$	2	1	0	3
z	4	5	3	2	3	0

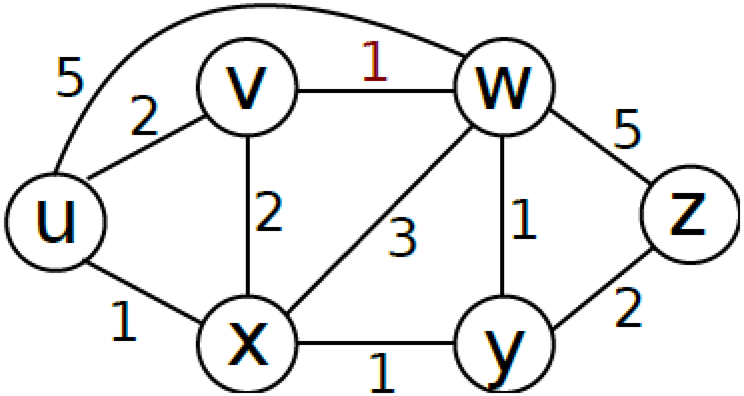


Example on table updates

All routers will update paths

to

from	u	v	x	y	w	z
u	0	2	1	2	$\boxed{3}$	4
v	2	0	2	3	1	5
x	1	2	0	1	2	3
y	2	3	1	0	1	2
w	$\boxed{3}$	1	2	1	0	3
z	4	5	3	2	3	0



Error Propagation

- In link state (LS) algorithms, routers report on cost of own links
  - An error in cost reporting will only affect neighbors
  - Error can be fixed with updated link cost announcement
- In distance vector (DV) algorithm, routers propagate path cost
  - This path cost will influence other's path cost
  - Error spreads through system, hard to fix
    - For example: count-to-infinity problem

Error Propagation

- In link state (LS) algorithms, routers report on cost of own links
  - An error in cost reporting will only affect neighbors
  - Error can be fixed with updated link cost announcement
- In distance vector (DV) algorithm, routers propagate path cost
  - This path cost will influence other's path cost
  - Error spreads through system, hard to fix
    - For example: count-to-infinity problem
- How to fix?
  - Remember path nodes, make sure to not visit twice
  - Or: put limit on cost of path (+constant link cost)
  - Or: detect and remove dead neighbors

Count-to-infinity Problem

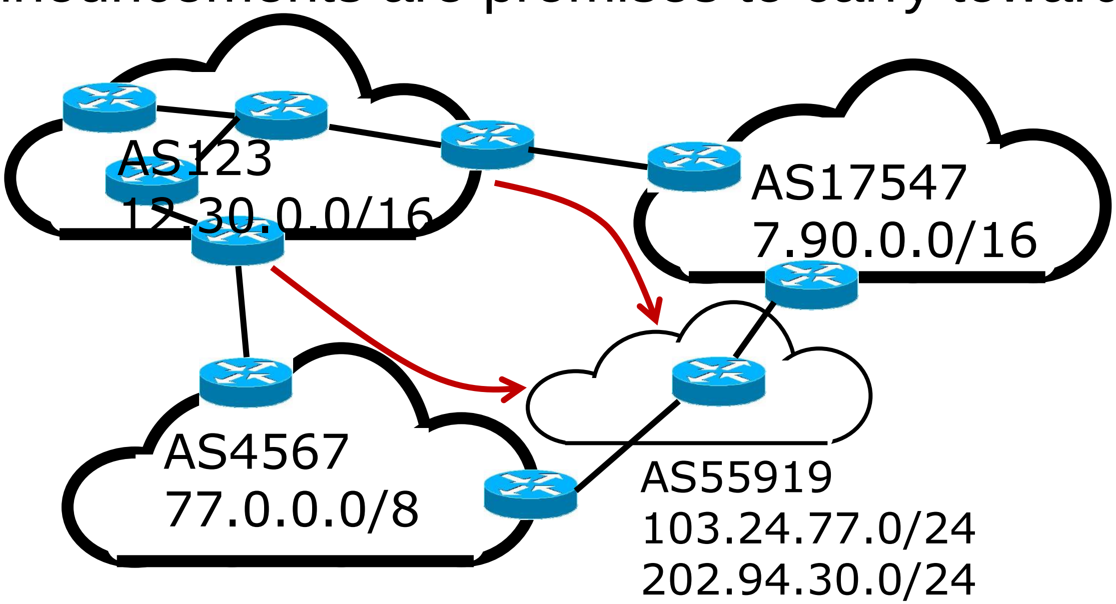
- Changes in link cost propagate differently
- If link cost decreases: no problem
  - Good news spreads fast
- Potential problem: if link cost increases, or link disappears
  - Bad news spreads slow
  - Distance vector entry does not tell us if link FOO is part
  - There might be old path entries that are implicitly based on FOO
  - These can have lower cost than real paths
  - If no path exists any more, nodes can get stuck in *counting-to-infinity* loop

Intra-/Inter-AS routing

- Different routing algorithms are used within and outside an AS
- Within an AS:
  - Limited number of links and routers
  - Link state algorithms can be feasible
- Common Intra-AS protocols
  - RIP: Routing Information Protocol (DV)
  - OSPF: Open Shortest Path First (LS)
- Inter-AS routing:
  - LS infeasible, DV (or PV) needed
  - BGP is the most common PV protocol

Border Gateway Protocol (BGP)

- BGP is a path vector protocol
  - Paths are announced, without distances
  - Paths actually include sequence of nodes on the path
- Two variants:
  - eBGP for communication between AS
  - iBGP to propagate routes internally within AS
- eBGP is based on TCP connections between routers
- eBGP announcements are promises to carry towards target prefix



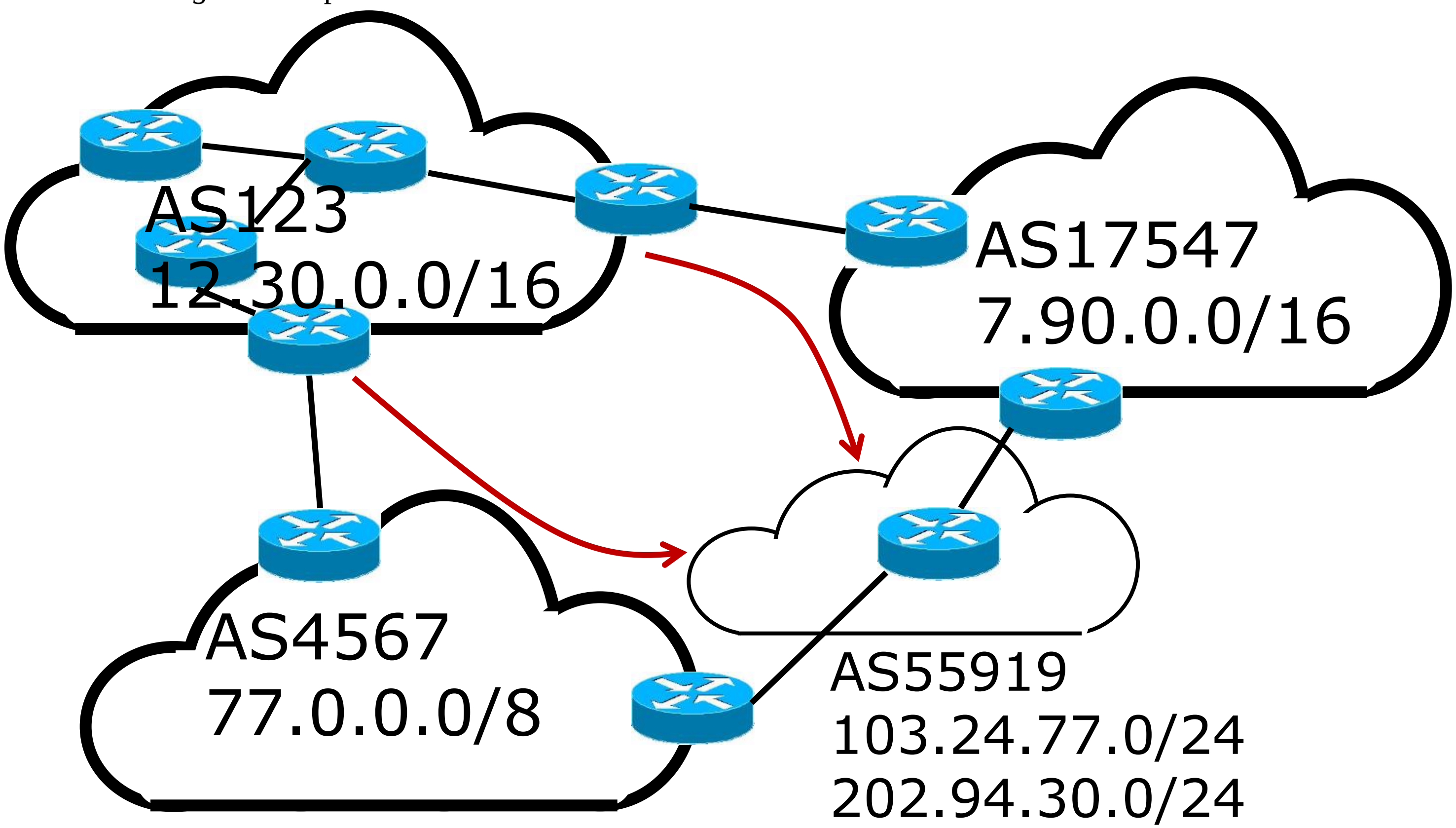
http://bgp.he.net/AS55919#_asinfo

eBGP announcements

- Advertised route contains:
  - Target prefix (CIDR network)
  - BGP attributes
- Two main attributes:
  - AS-PATH: list of AS that are on path
  - NEXT-HOP: IP address of router to reach current AS
- The receiver of announcement can decide to import route
  - Depends on local *policies*
- Example for SUTD AS
  - 103.24.77.0/24
  - AS-PATH AS55919
  - NEXT-HOP 103.24.77.1

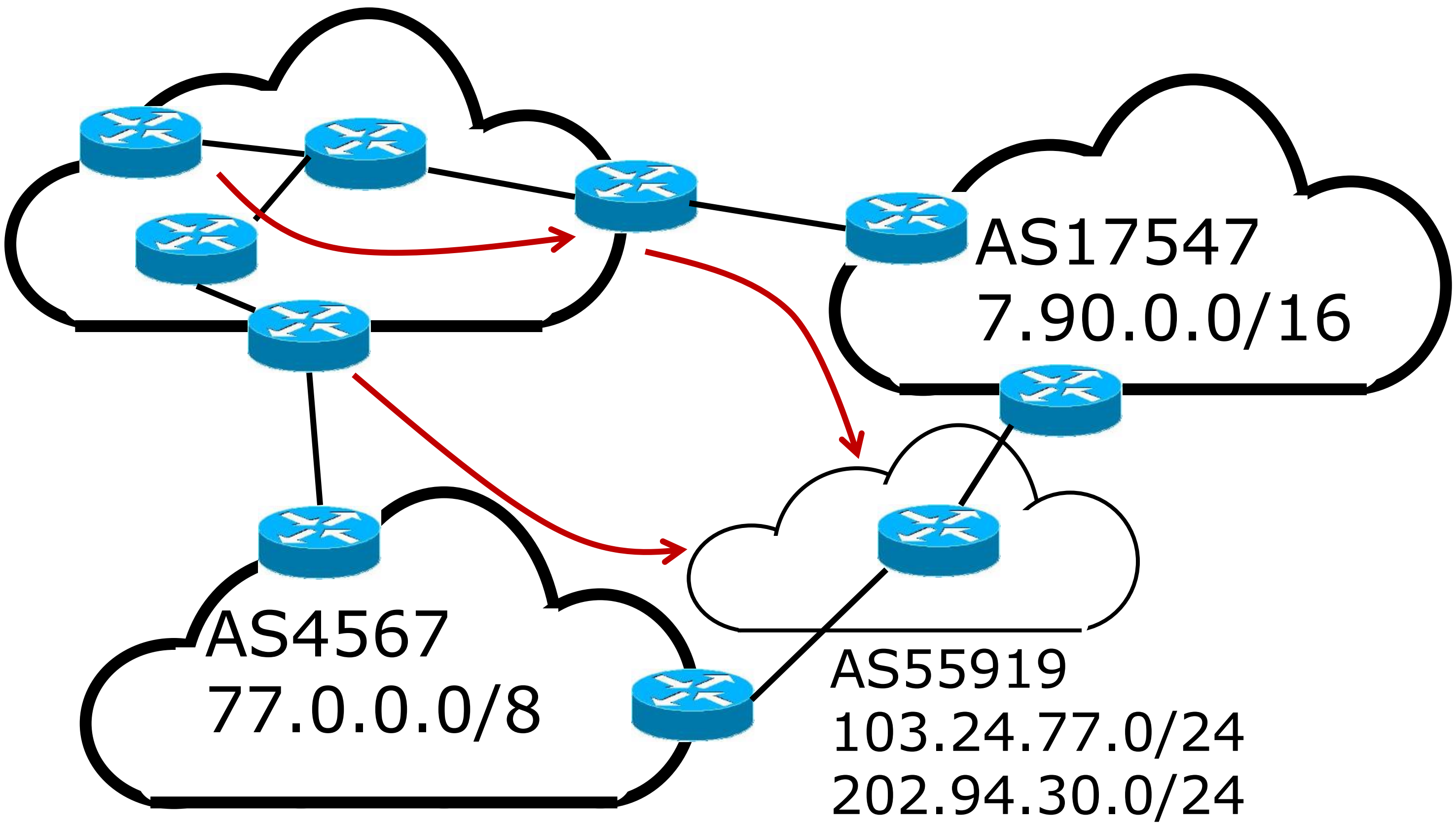
eBGP Route Selection

- Receiver of BGP announcement might already have route entries to some announced prefix
- There should only be one entry to every prefix
  - How to select which one?
- Selection based on:
  - Length of AS-PATH (not directly hops, number of AS)
  - Local policies (do not route through NUS)
  - *Hot Potato Routing*: least hops to NEXT-HOP of route



Update of local routing table

- After receiving new route entry on border router:
  - Use OSPF to find best route to NEXT-HOP
  - Identify first hop on best route, use its interface for routing table



Network Address Translation

Network Address Translation

- Private IP addresses cannot receive traffic from the internet
- How can we enable networks with only private addresses to *connect* to the Internet?

Network Address Translation

- Private IP addresses cannot receive traffic from the internet
- How can we enable networks with only private addresses to *connect* to the Internet?

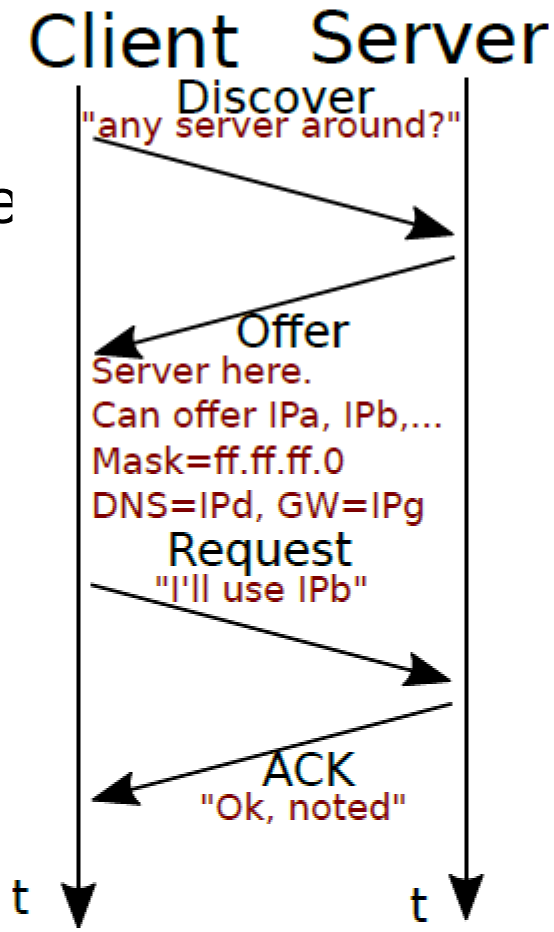
- An intermediate gateway-like device with public IP and private IP can forward traffic from private IP users
  - Gateway uses its public IP as source of traffic
  - Gateway uses own source port for connection
- When responses are received by Gateway, it translates back
  - To private source/destination IP
  - To original source/destination port

Network Address Translation (NAT)

- This technique is called NAT, and it is a fundamental aspect of modern networking
- NAT'ing only works for connections initiated from the private IPs
  - Private IPs cannot be directly reached from internet
- The NAT will set up a tuple for each forwarded connection
  - [sourceIP,destIP, sourcePort,destPort, natPort]
- Using this tuple, response messages can be translated back
- NAT'ing can also translate between public networks, but is not often used that way
- NAT'ing works on Transport layer, not Application layer
  - The NAT does not care about application layer protocol

Dynamic Host Configuration Protocol

- More commonly known as DHCP
- Protocol to provide Network Layer and above configuration
- Managed by DHCP server running on the same Link layer *Broadcast domain*
  - As clients do not have an IP address, no sender IP is possible, no gateway is known
  - Four messages are exchanged in total
  - To allow DHCP requests to reach a server in another broadcast domain, *DHCP forwarders* can be used



The Link Layer

The Link Layer

- Provides single-hop connection for *frames* between interfaces
  - Protocols: 802.3 Ethernet, 802.11 Wlan, ARP
  - Addressing is based on MAC addresses
- Remember:
  - App layer: connection between applications
  - Transport layer: connection between processes
  - Network layer: connections between hosts
- Main services of Link layer:
  - Addressing, framing, medium access control (MAC)
  - Error detection & correction

Link layer services

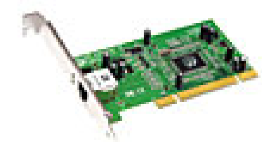
- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - we learned how to do this already (chapter 3)!
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - *Q:* why both link-level and end-end reliability?

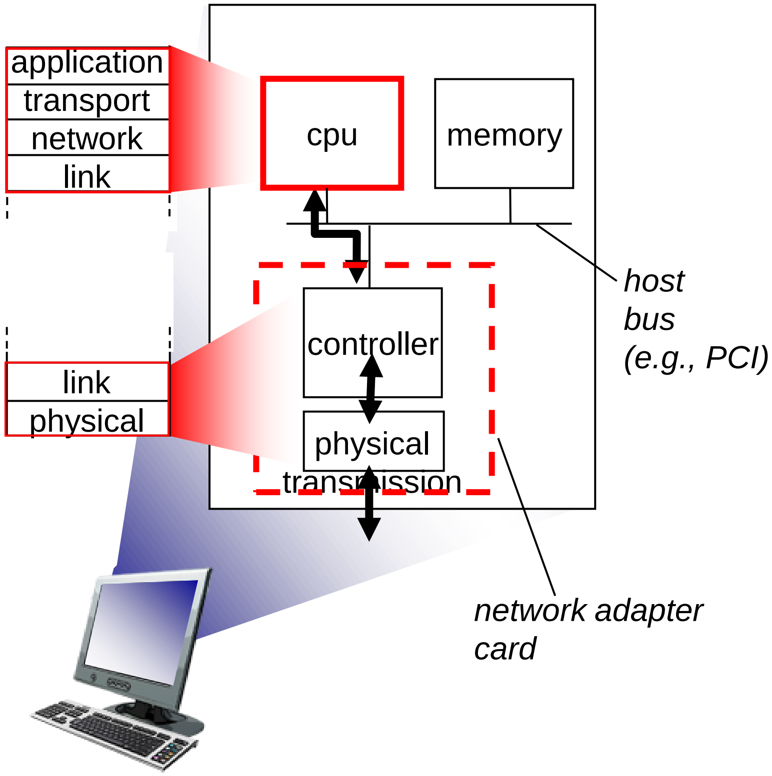
Link layer services (more)

- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error detection*:
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- *error correction:*
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

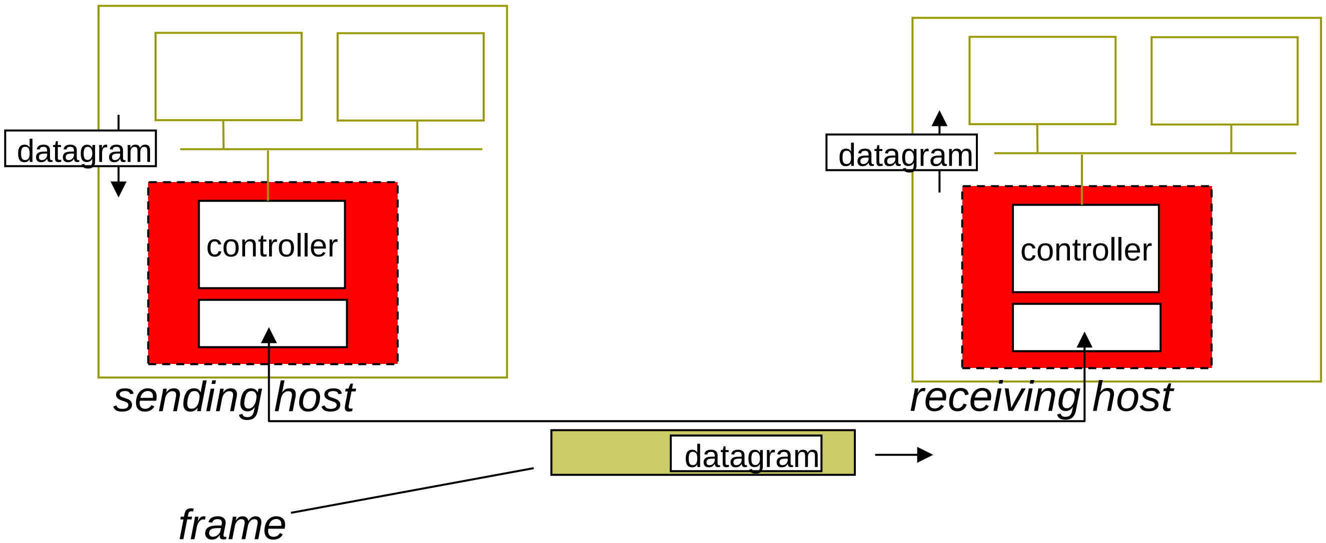
Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC) or on a chip
  - Ethernet card, 802.11 card; Ethernet chipset
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware





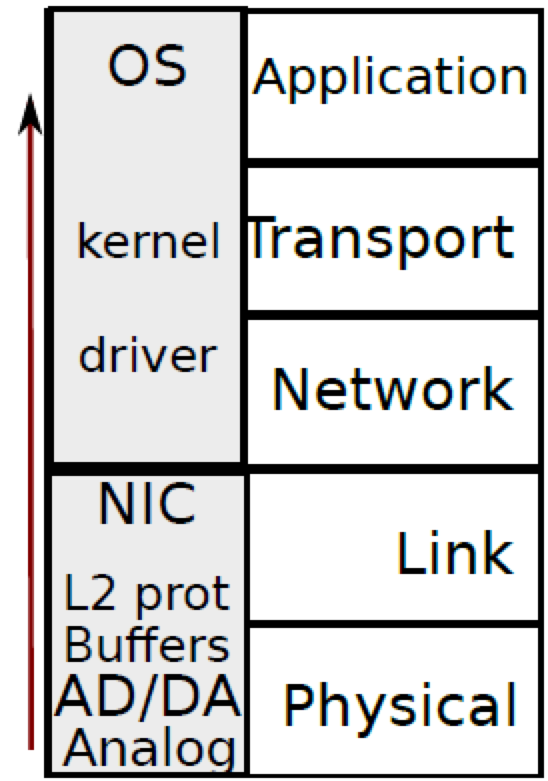
Adaptors communicating



- sending side:
  - encapsulates datagram in frame
  - adds error checking bits, rdt, flow control, etc.
- receiving side
  - looks for errors, rdt, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

Network Interfaces

- Netw. interface card (NIC) handles L1+L2
  - Application layer data is provided to the operating system/ processes
- NIC is processing in soft-/ or hardware
- Memory for input buffers
  - (e.g., IP and TCP fragments)
- Memory for output buffers
  - (e.g. TCP non-acked fragments)



IEEE 802.1, 802.3, 802.3, …

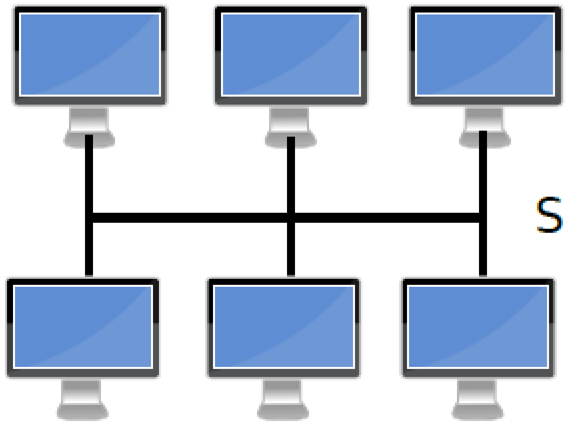
- Numbers denote working groups within IEEE
  - subversion numbers usually denote standards
- 802.1 is working on the following (and more)
  - 802 architectures, security
  - Internetworking between different 802 networks
- 802.2 is working on Link Layer Control (LLC)
  - Essentially, different Link Layer Headers
- 802.3 is working on wired LANs/ Ethernet
  - 802.3u is normal 100Mbit ethernet
- 802.11 is wireless working group
  - 802.11 (a,b,g,n) are different wireless standards

Ethernet

IEEE 802.3: Wired Ethernet

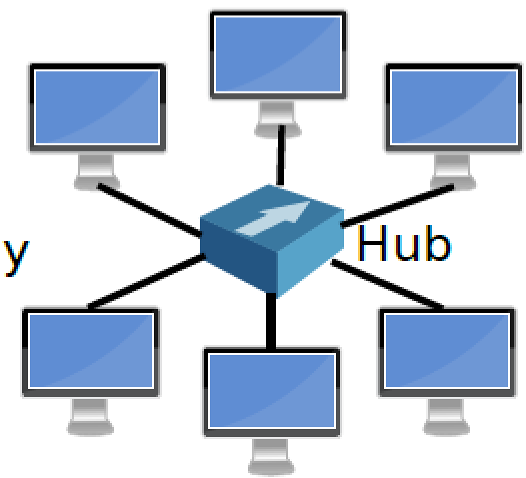
- The de-facto standard for wired LANs
- 802.3 specifies properties of Link and PHY layer
- Many physical layer variants (fibre, copper)
- Most common use: 802.3u, cheap twisted pair copper wires
- Originally started with 10Mbit/s
    - Coaxial cables, bus/ring topology
    - Shared medium, hubs (L1 repeaters)
- Extended over time, 10Gbit/s and more now
- Nowadays: Star architecture with L2+ switch
    - Medium not shared anymore, all peer-to-peer

Ethernet Shared Medium Topologies



Shared Bus Topology
10BASE2 (coax)

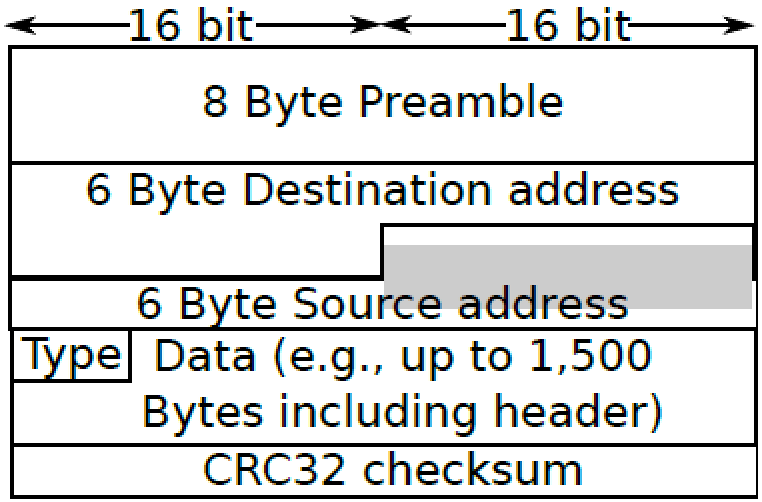
Shared Star Topology
10Base-T



Ethernet Protocol

- Ethernet consists of different 802.3X standards
  - Copper/Fibre for PHY, headers very similar
- Connectionless - Like UDP, no handshakes etc.
- Unreliable - No retransmission protocol, but CRC
  - Cyclic Reduncancy Checksum allows to detect errors
  - Algorithm is based on math (GF(), see security lecture)
- Medium Access - Wired Ethernet can handle collisions
  - Without full duplex: CSMA/CD with exponential backoff
  - More on that later (PHY Layer)
- Why is a shared medium a problem/ how can it be solved?

Ethernet Frame overview



- 802.3 (Ethernet II) header provides PHY-specific services
- *Preamble* required for sender/receiver synchronization
- Error detection: *cyclic redundancy checks*, e.g., CRC32
- Optionally: Flow control, error correction, etc (e.g., WLAN)
- Preamble and CRC are not provided to wireshark by interface

Ethernet Addresses

- Also called MAC addresses, identify *interface adapter*
  - 48 bit (IPv4:32 bit)
  - Example: 5e:3e:c1:32:d1:41
- Supposed to be globally unique
  - Must be unique in local network
  - Assigned by manufacturer
  - Manufacturers get sub-space assigned
- Addresses meant to be permanent (lifetime)
  - In some OS you can change MAC if you want

Address Resolution Protocol

MAC addresses and ARP

- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
  - function: *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
  - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation
(each "number" represents 4 bits)

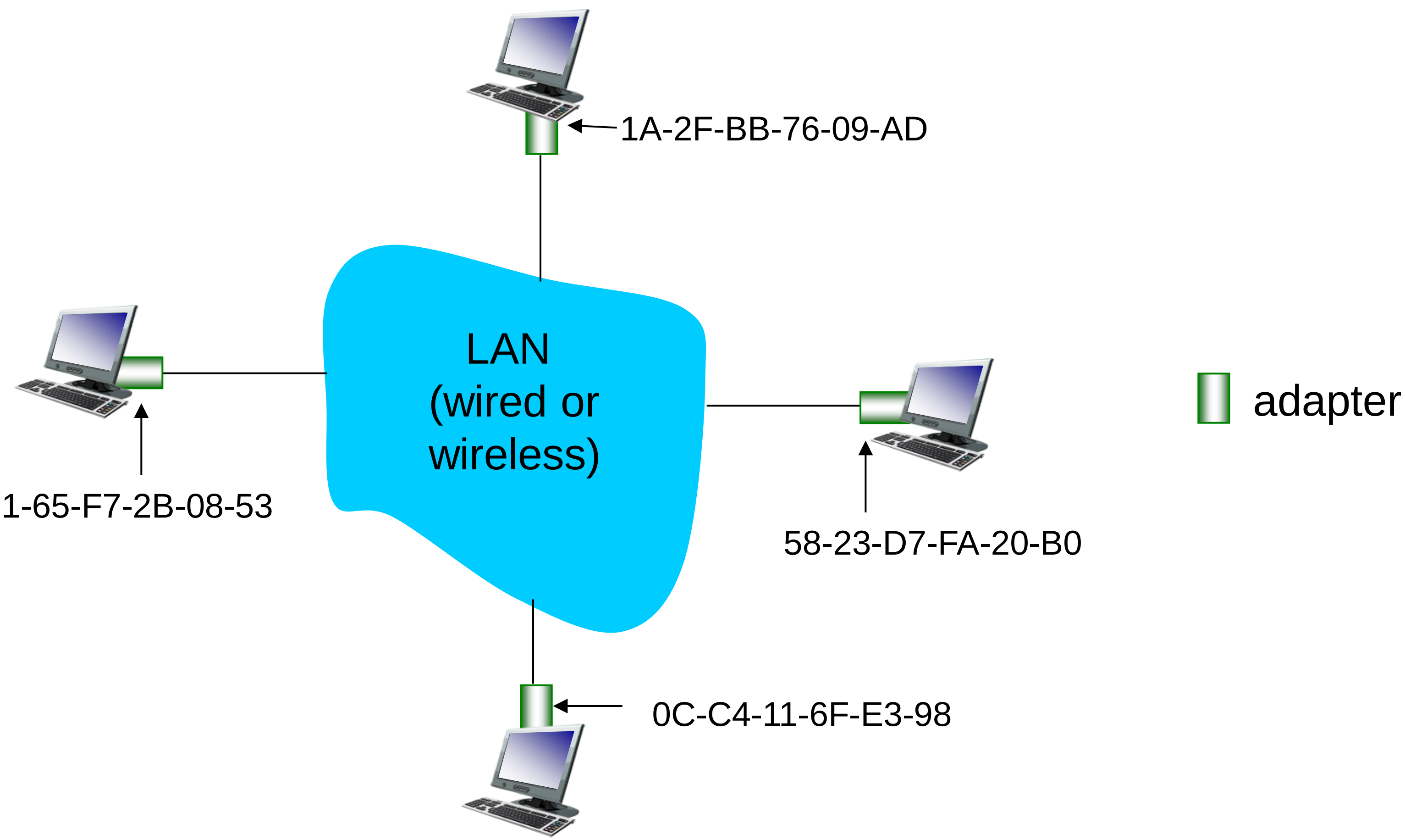
Address Resolution Protocol (ARP)

- How are URLs/IP+Port/IP mapped to MAC address?
- On network layer, next hop on path is found
  - This yields an IP address
- The sender then needs to find the MAC address of that host
  - Done with a simple broadcast request using ARP protocol
  - Target IP host will answer with his MAC
- MAC-addresses/ARP are only used in local networks
  - Single-hop, e.g., with Layer 2 switches
- Try for yourself (on Linux): arp will show your cache
  - Or the newer ip -s neighbor show

LAN addresses and ARP

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

each adapter on LAN has unique *LAN* address

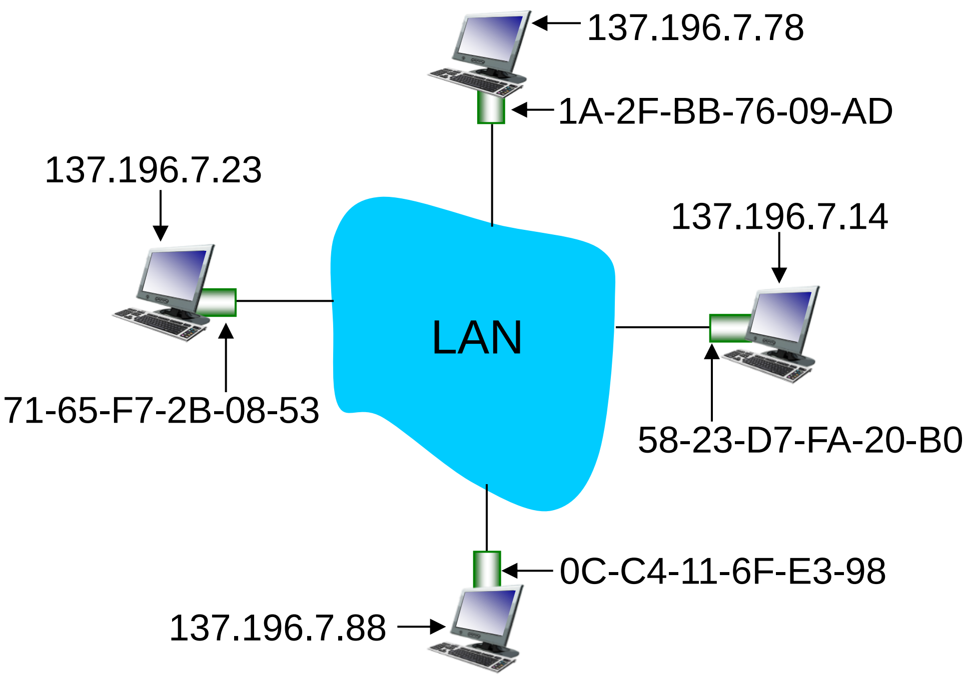


LAN addresses (more)

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:
  - MAC address: like Social Security Number
  - IP address: like postal address
- MAC flat address ➜ portability
  - can move LAN card from one LAN to another
- IP hierarchical address *not* portable
  - address depends on IP subnet to which node is attached

ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>

- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

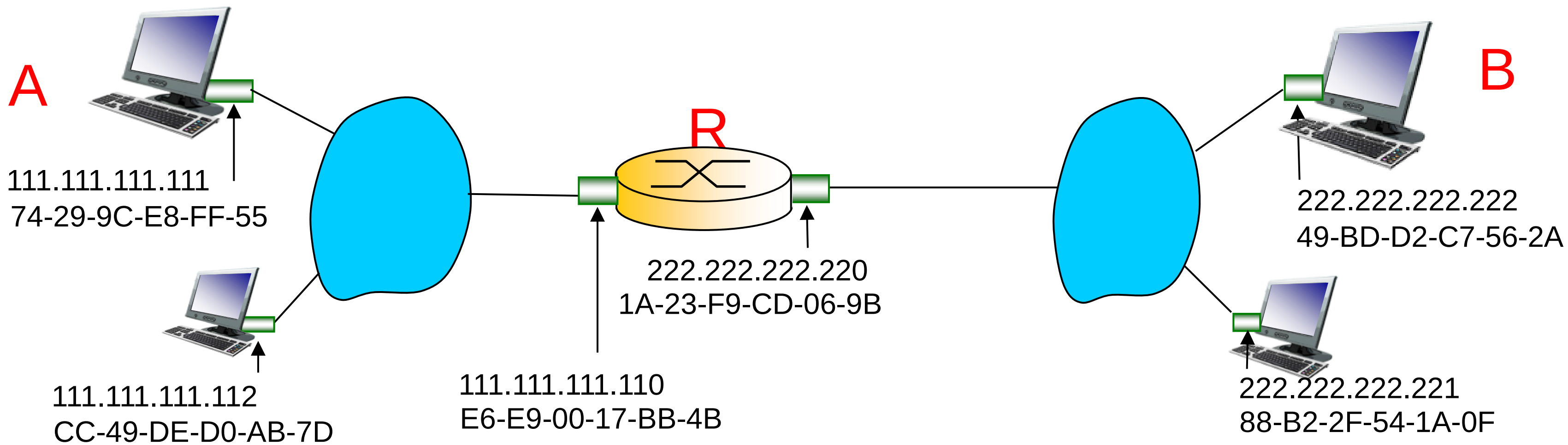
ARP protocol: same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - dest MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables *without intervention from net administrator*

Addressing: routing to another LAN

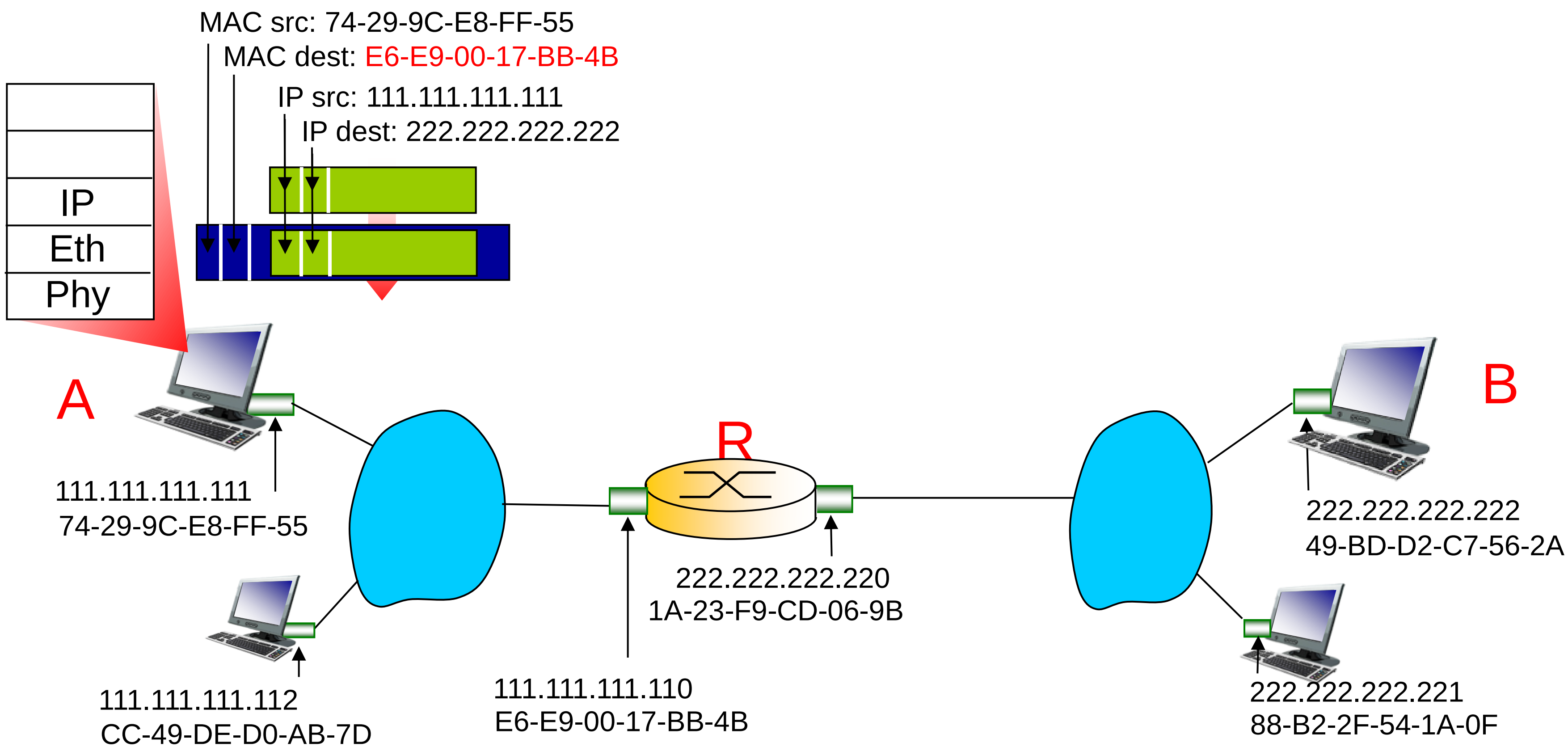
walkthrough: send datagram from A to B via R

- focus on addressing – at IP (datagram) and MAC layer (frame)
- assume A knows B's IP address
- assume A knows IP address of first hop router, R (how?)
- assume A knows R's MAC address (how?)



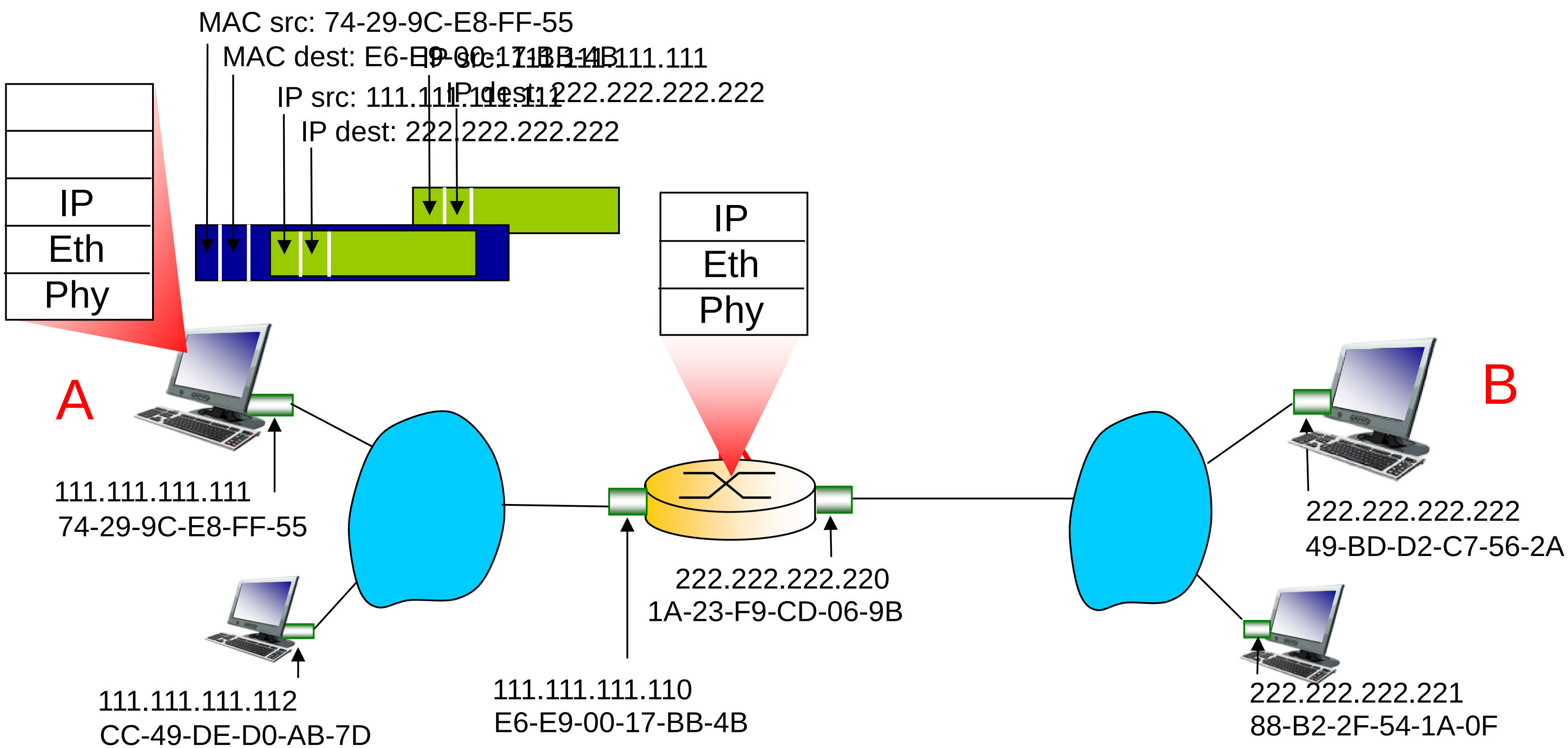
Addressing: routing to another LAN

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame with R's MAC address as dest, frame contains A-to-B IP datagram



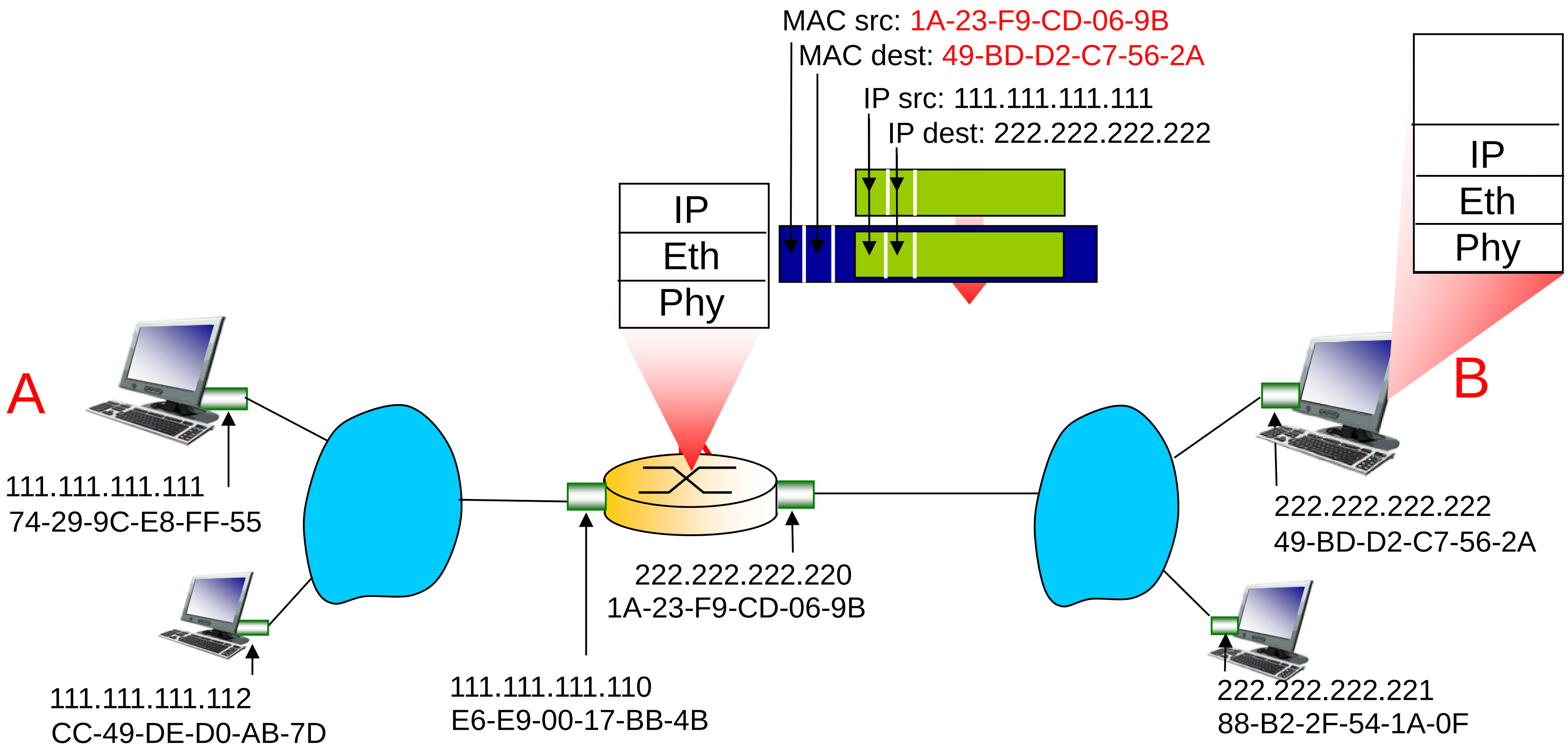
Addressing: routing to another LAN

- frame sent from A to R
- frame received at R, datagram removed, passed up to IP



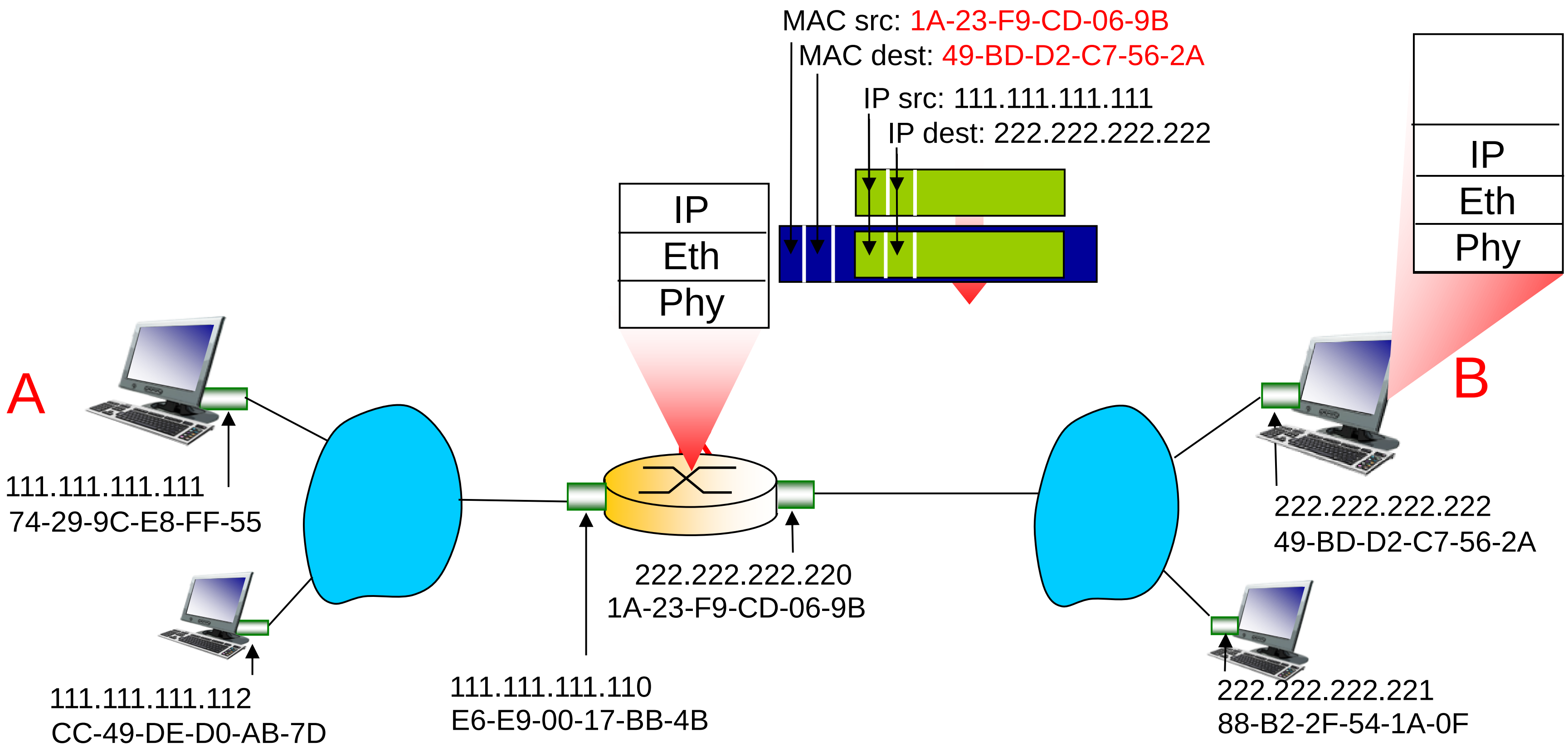
Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



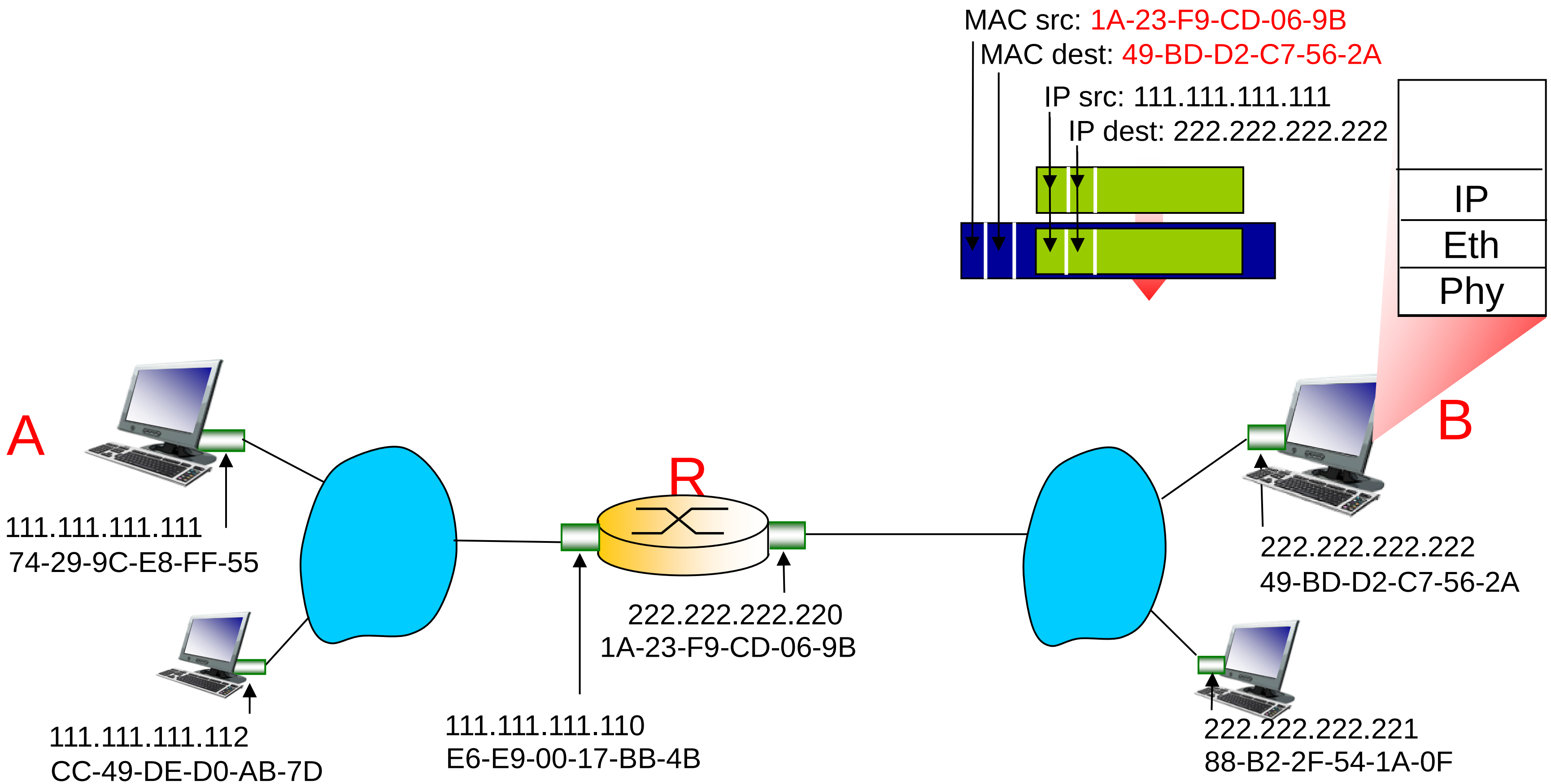
Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram



Addressing: routing to another LAN

- R forwards datagram with IP source A, destination B
- R creates link-layer frame with B's MAC address as dest, frame contains A-to-B IP datagram

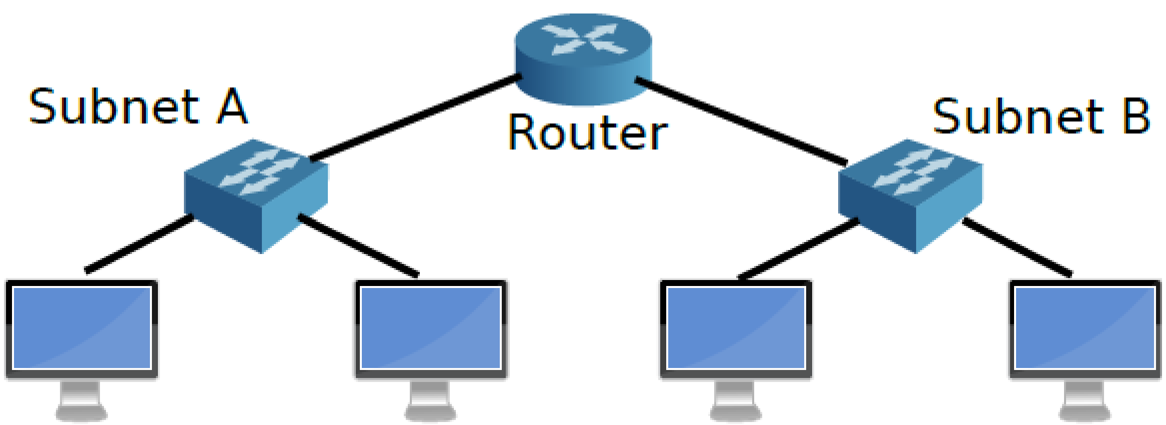


Hubs/Switches/Routers

- Hubs (not really used any more):
  - Layer 1 repeaters, no own MAC address
  - Re-transmit received packets on all ports
- Layer 2 Switches:
  - Passively learns MAC addresses from traffic
  - Store these in a CAM table (no active ARP)
  - Forward incoming packets to correct port (if known)
  - Hubs/Layer 2 switches are invisible to other network members
- Layer 3 Routers:
  - Router itself has MAC/IP address
  - Sender path would have router MAC as first hop
  - Router itself uses ARP/ MAC table like any host

Link-layer Broadcast Domain (LLBD)

- LLBD spans all *single-hop destinations*
  - L1 Hubs/ L2 Switches do not count as hop
  - Usually use one Network-layer subnet (>1 possible)
  - Network-layer subnets can only have one LLBD
  - Security/usability constraints
    - No filtering through firewall
    - Locally broadcasted services (printer, shared music, etc)
- Intuition: *physical proximity* implies *logical proximity*
- How to separate neighbors into two broadcast domains?



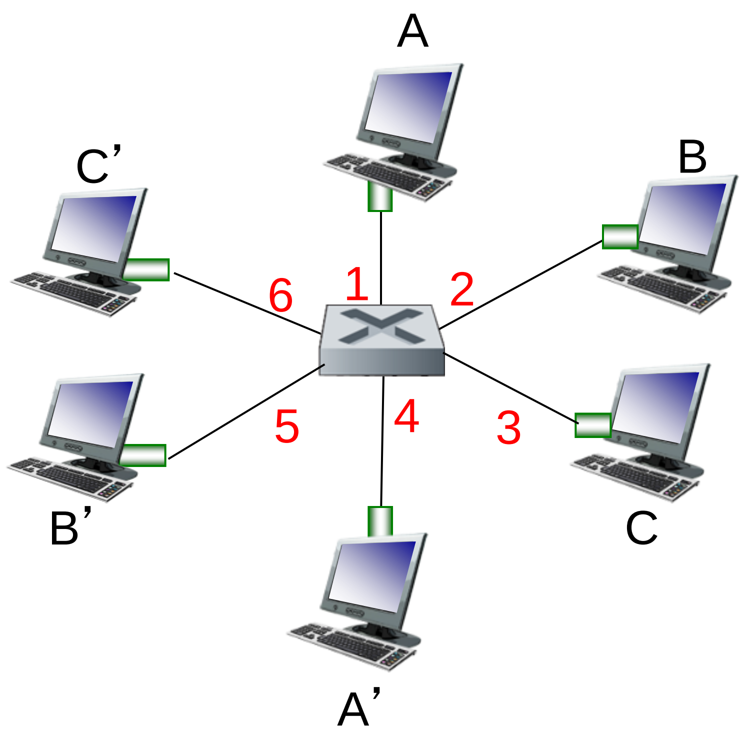
Two subnets connected by a router

Ethernet switch

- link-layer device: takes an *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

Switch: *multiple* simultaneous transmissions

- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A' and B-to-B' can transmit simultaneously, without collisions

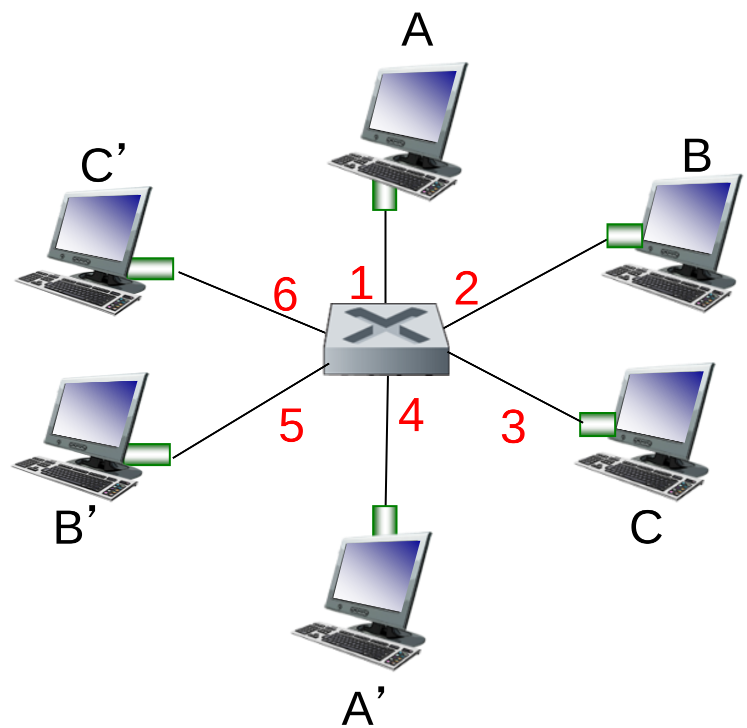


switch with six interfaces (1,2,3,4,5,6)

Switch forwarding table

Q: how does switch know A' reachable via interface 4, B' reachable via interface 5?

- A: each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
  - looks like a routing table!



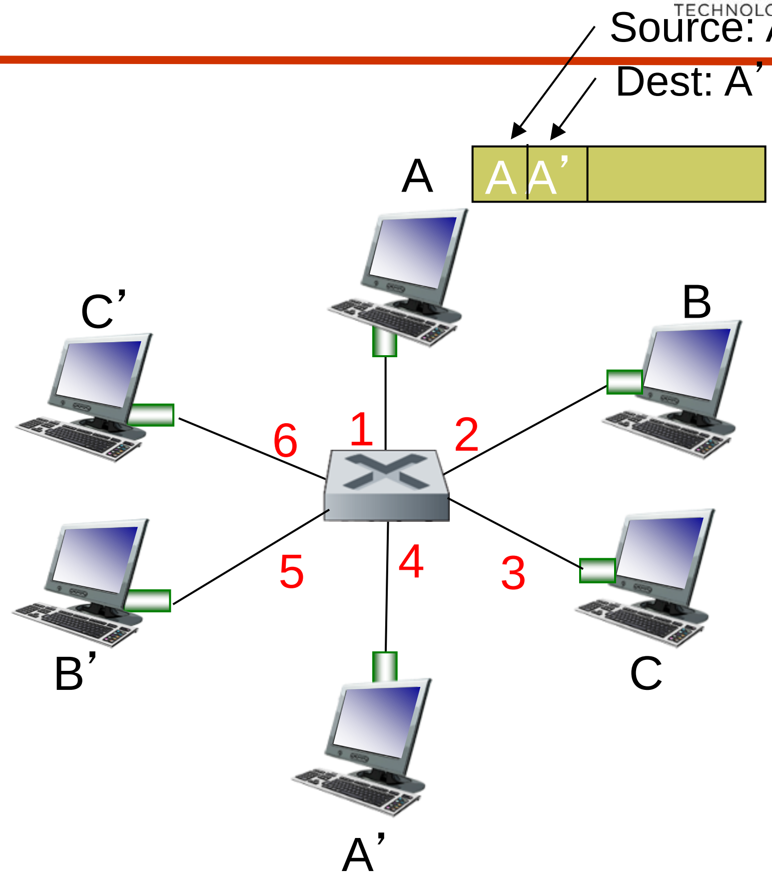
switch with six interfaces (1,2,3,4,5,6)

Q: how are entries created, maintained in switch table?

- something like a routing protocol?

Switch: self-learning

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

Switch: frame filtering/forwarding

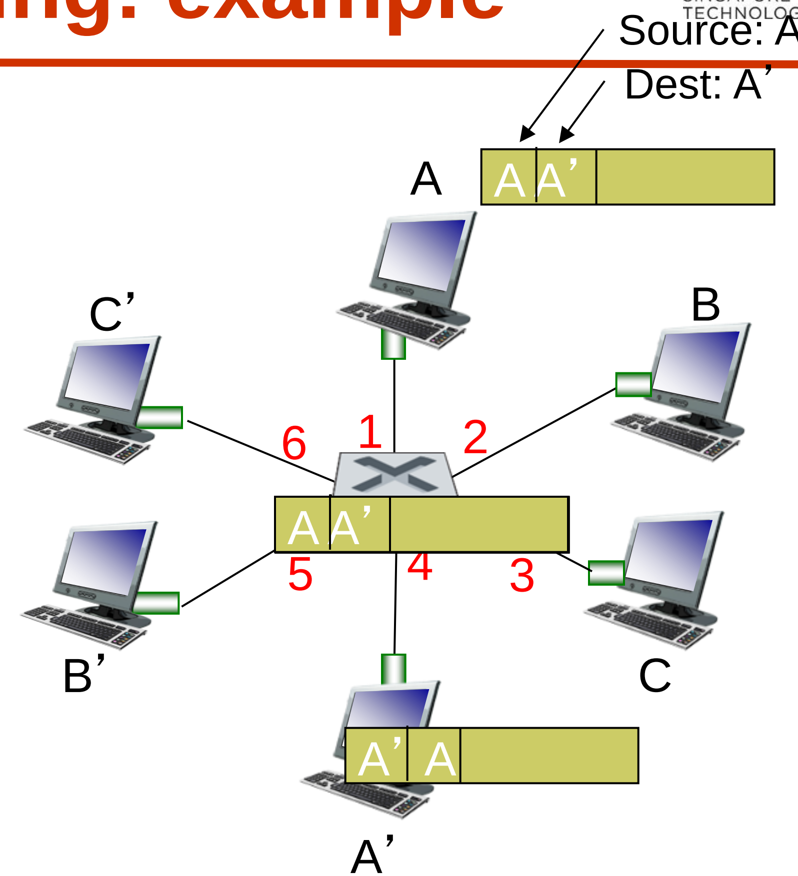
when frame received at switch:

1. record incoming link, MAC address of sending host
2. index switch table using MAC destination address
3. if entry found for destination
   then {
   if destination on segment from which frame arrived
   then drop frame
   else forward frame on interface indicated by entry
   }
   else flood /* forward on all interfaces except arriving interface */

Self-learning, forwarding: example

- frame destination, A', locaton unknown: *flood*

○ destination A location known: selectively send on just one link

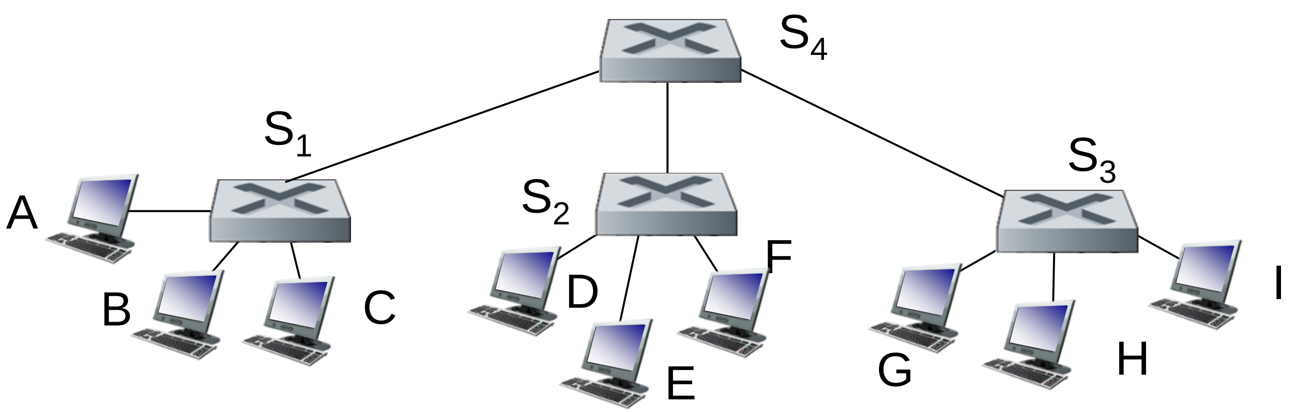


MAC addr	interface	TTL
A	*1*	*60*
A'	*4*	*60*

switch table (initially empty)

Interconnecting switches

- switches can be connected together



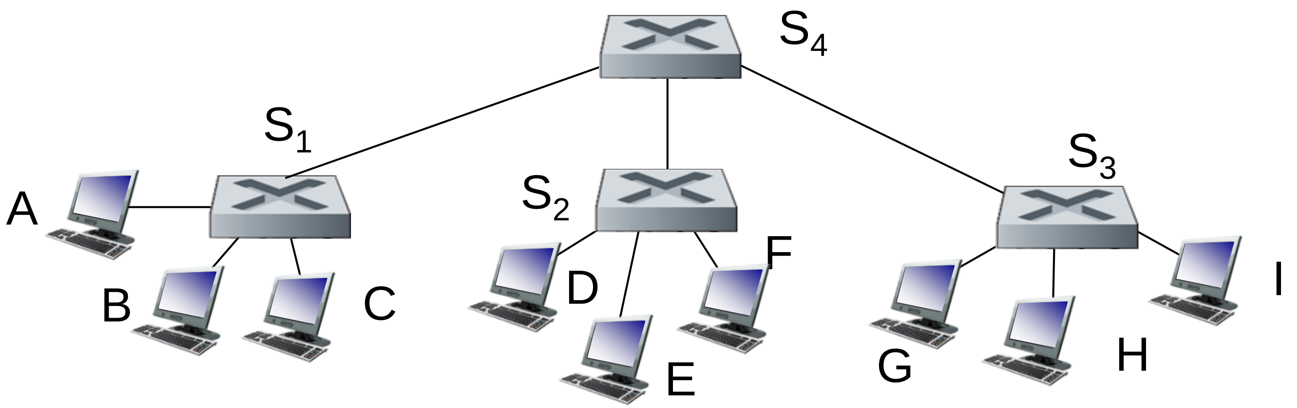
Q: sending from A to G - how does S_1 know to forward frame destined to F via S_4 and S_3?

- A: self learning! (works exactly the same as in single-switch case!)

Self-learning multi-switch example

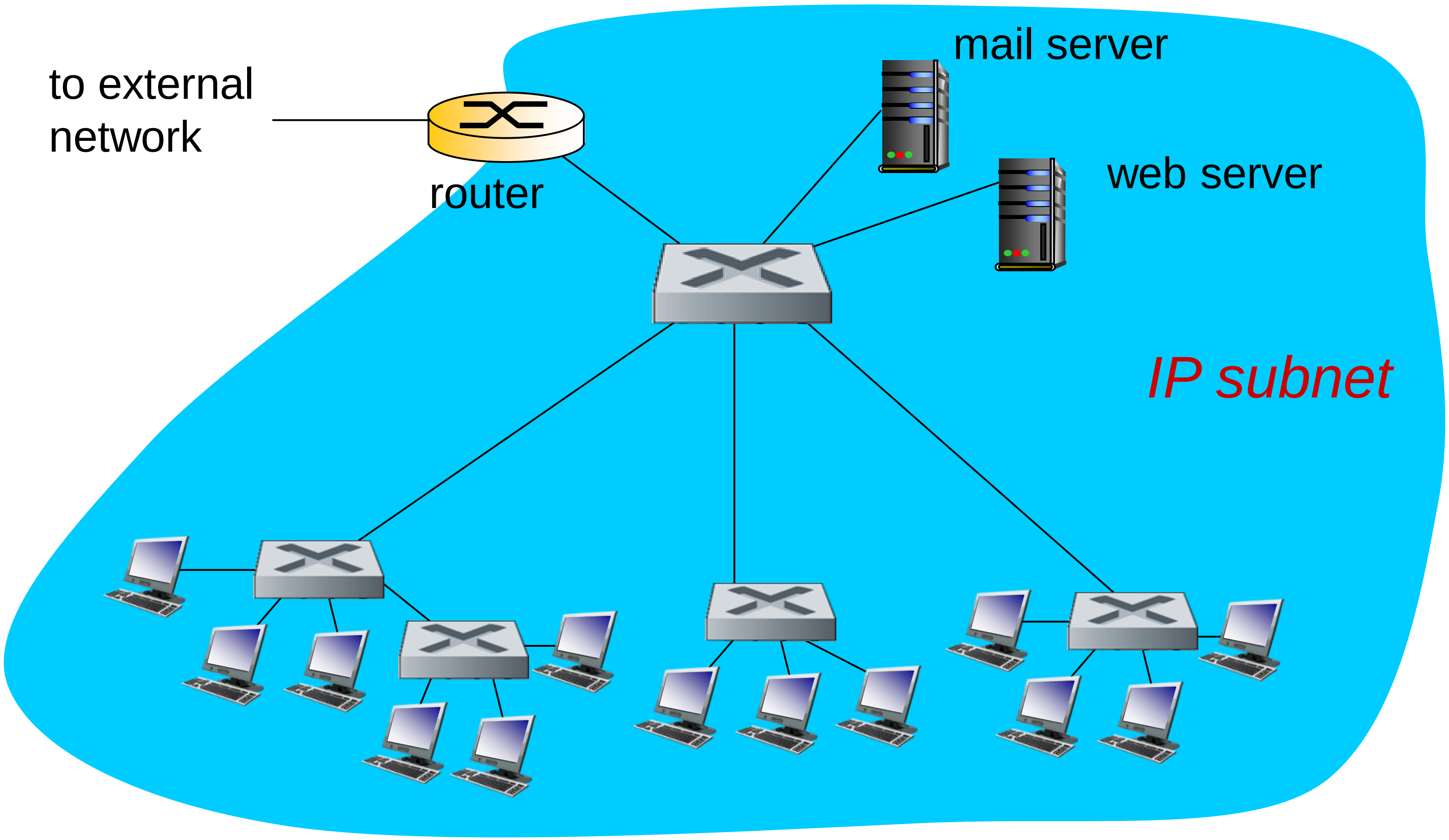
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Suppose C sends frame to I, I responds to C



- Q: show switch tables and packet forwarding in S_1, S_2, S_3, S_4

Institutional network



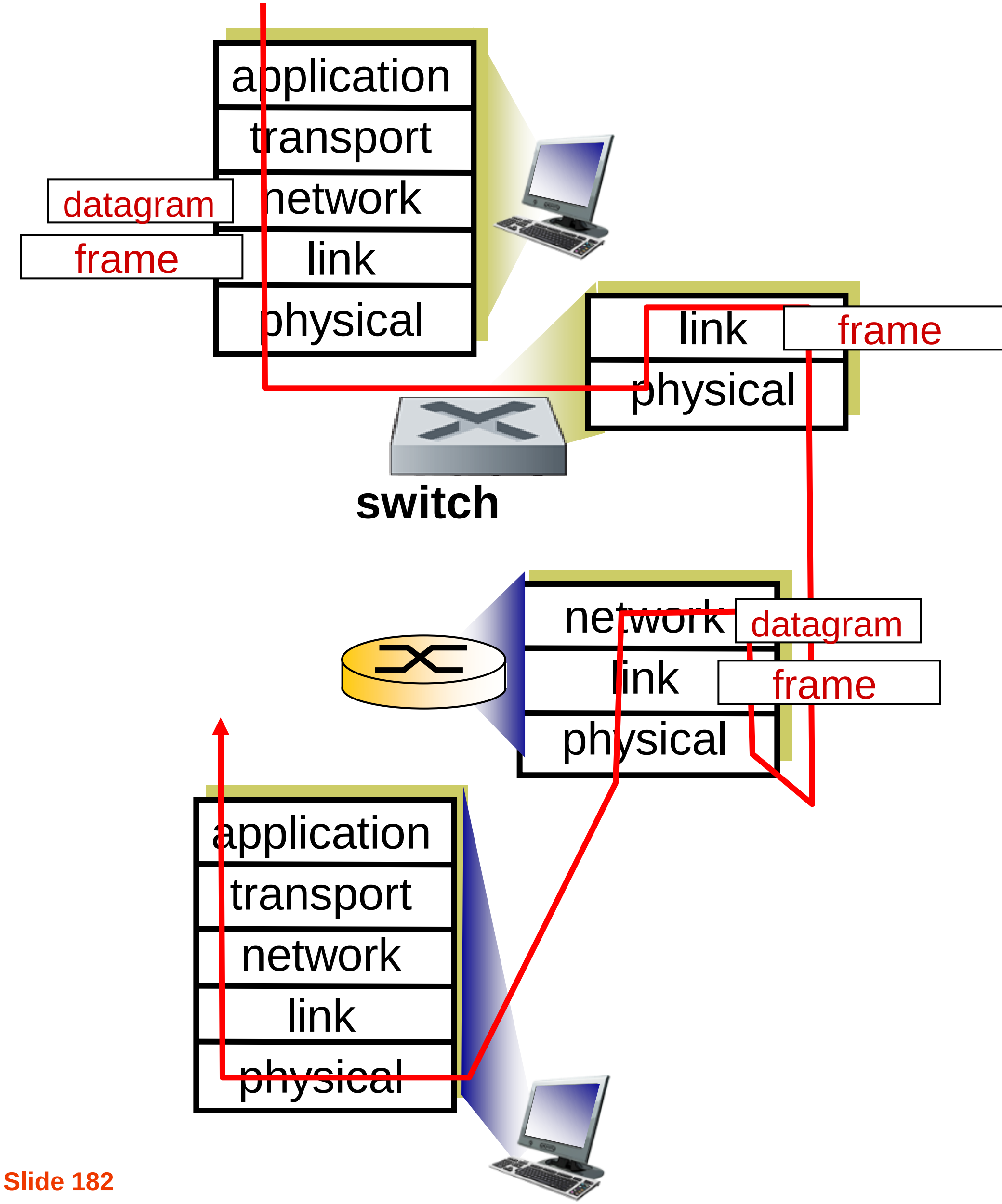
Switches vs. routers

both are store-and-forward:

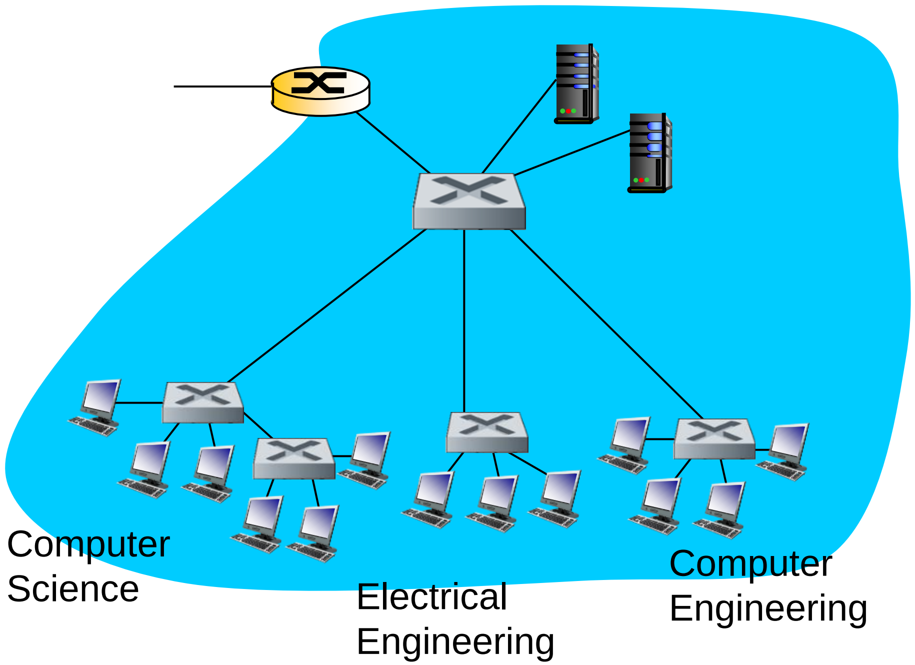
- *routers:* network-layer devices (examine network-layer headers)
- *switches:* link-layer devices (examine link-layer headers)

both have forwarding tables:

- *routers:* compute tables using routing algorithms, IP addresses
- *switches:* learn forwarding table using flooding, learning, MAC addresses



VLANs: motivation



consider:

- CS user moves office to EE, but wants connect to CS switch?
- single broadcast domain:
  - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
  - security/privacy, efficiency issues

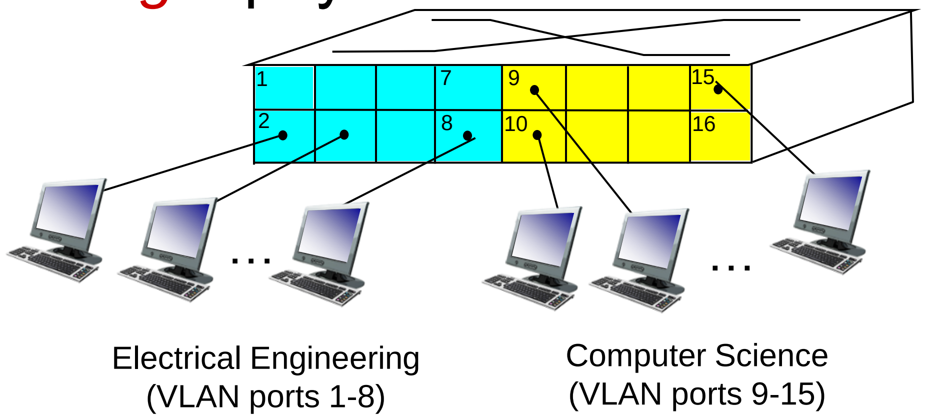
VLANs

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

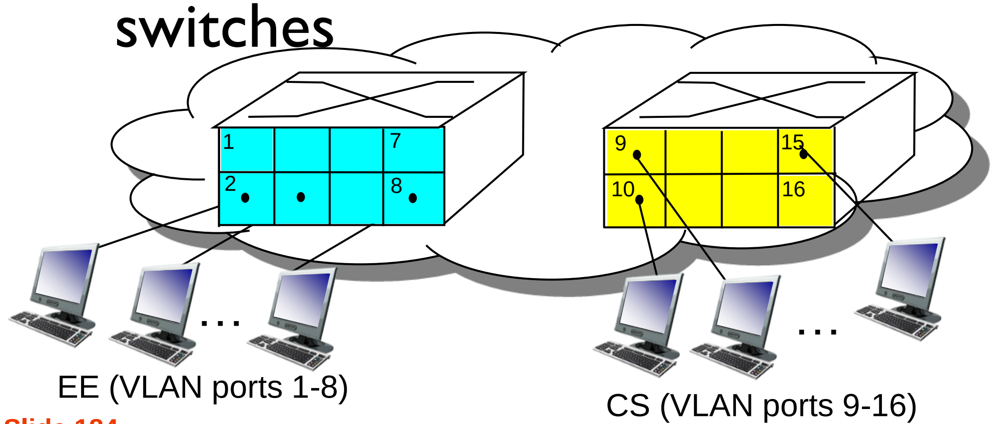
Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple ***virtual*** LANS over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch ……

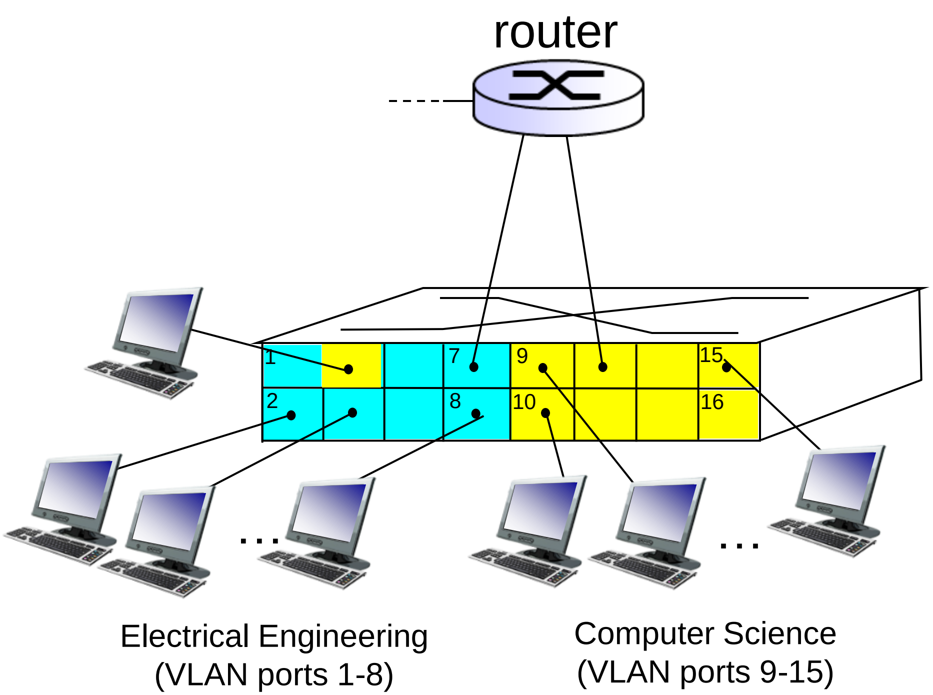


… operates as *multiple* virtual switches

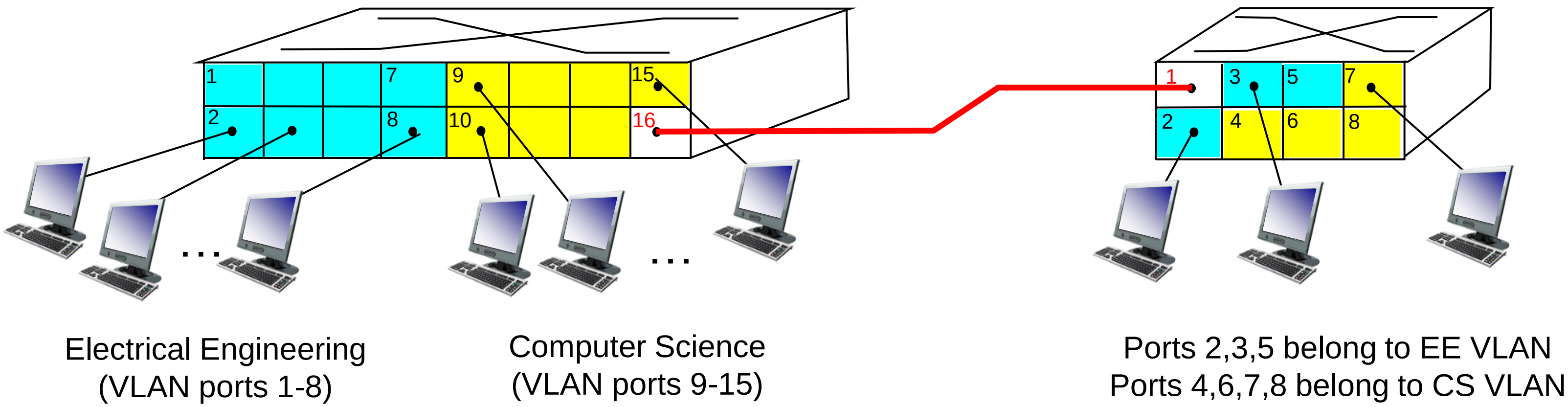


Port-based VLAN

- *traffic isolation:* frames to/from ports 1-8 can *only* reach ports 1-8
  - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- dynamic membership: ports can be dynamically assigned among VLANs
- forwarding between VLANS: done via routing (just as with separate switches)
  - in practice vendors sell combined switches plus routers



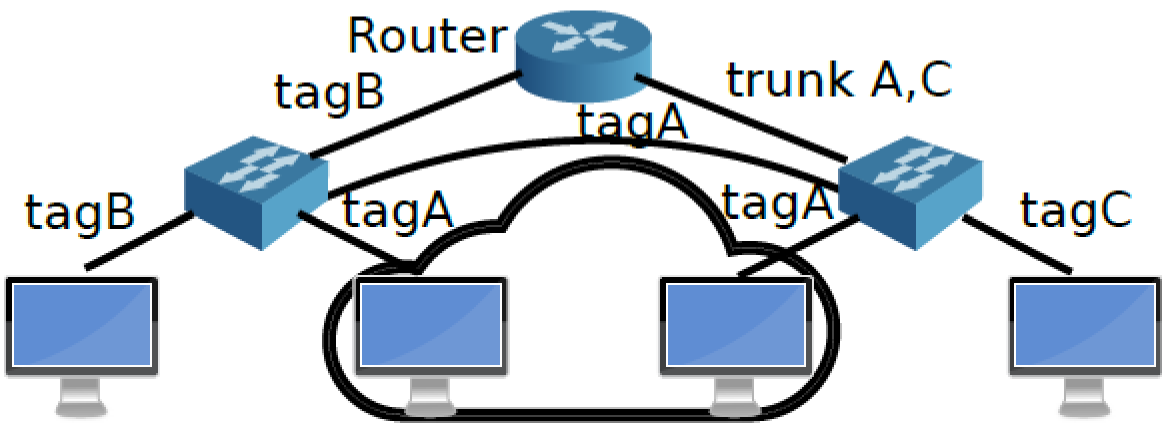
VLANS spanning multiple switches



- *trunk port:* carries frames between VLANS defined over multiple physical switches
  - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
  - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

VLAN trunks

- One cable per VLAN is clearly inefficient
- How to send traffic of several VLANs over one cable?
- A VLAN ID is added in between Link and Network layer
- Defined by 802.1Q – additional 12 bit VID field after ETH header
- The sender adds VLAN tag, receiver removes it (or passes on)
- The sender needs to expect specific VLAN IDs



Note: In this configuration, A1 + A2 reach each other and the router

Encapsulation

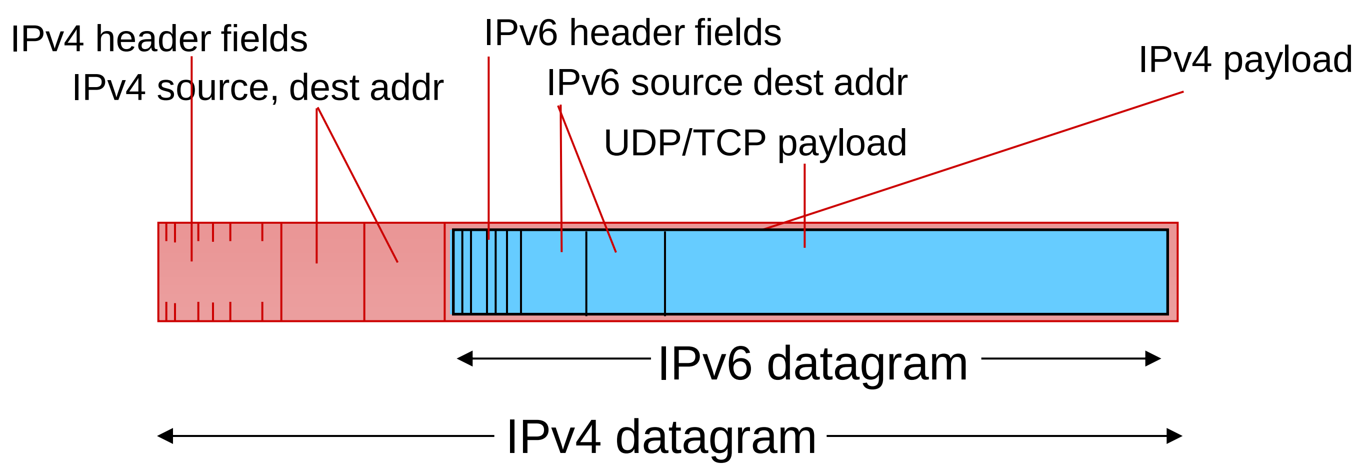
- **Encapsulation** is the process of taking data from one protocol and translating it into another protocol, so the data can continue across a network.
- Encapsulation may also be used over a **tunnel**, which is a special connection made over a network between two computers or network devices.

Encapsulation

- So far, we considered three types of protocols
  - Application protocols (e.g. HTTP, FTP, SSH)
  - Service protocols to enable networking (ARP, DHCP, DNS)
  - Foundational protocols (Ethernet, IP, TCP, UDP)
- Relatively straight forward protocol set (stack), matches to 5 layers we discussed
  - Discussion today will focus on encapsulation, i.e. nesting of partial protocol stacks as payload of other protocol stacks

Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



Bridging

- Bridging connects two interfaces of one host on link layer
- All traffic arriving on interfaceA will be forwarded to interfaceB
  - Think about the two interfaces like a 2-port switch
  - In particular, bridged interfaces normally don't have an IP
  - The bridge will be transparent to other devices on network
- A *br0* device will be created, that can have IP
- Use case examples:
  - Bridge wired and wireless network
  - Bridge two wired networks with different link layer
  - Bridge two wired networks to sniff traffic
  - Connect VM guest interface with physical host interface

Bridging remote interfaces?

- It would be useful to bridge a local and one remote interface
- This could extend link-layer broadcast domain to remote site
- Similar to VLAN
  - But VLAN 802.1Q header below IP
  - Will be removed by router and not forwarded
- How can we achieve something similar?

Bridging remote interfaces?

- It would be useful to bridge a local and one remote interface
- This could extend link-layer broadcast domain to remote site
- Similar to VLAN
  - But VLAN 802.1Q header below IP
  - Will be removed by router and not forwarded
- How can we achieve something similar?

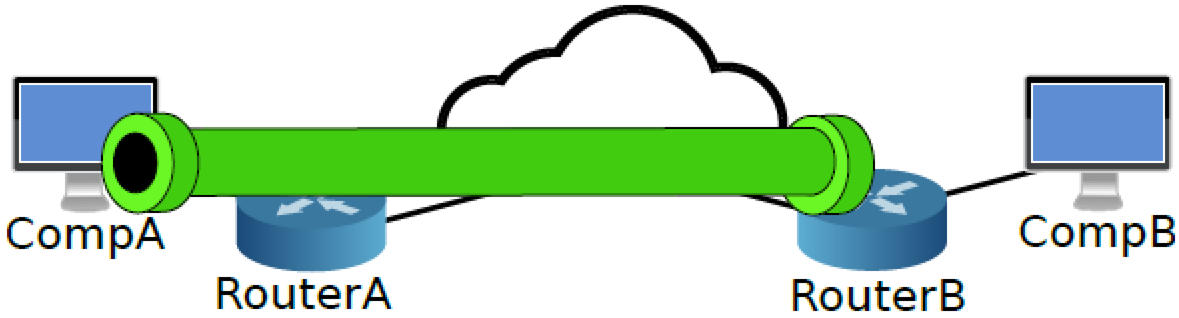
  - Layer 2 VPNs !

In a layer 2 VPN, the entire communication from the core VPN infrastructure is forwarded in a layer 2 format on a layer 3/IP network and is converted back to layer 2 mode at the receiving end.

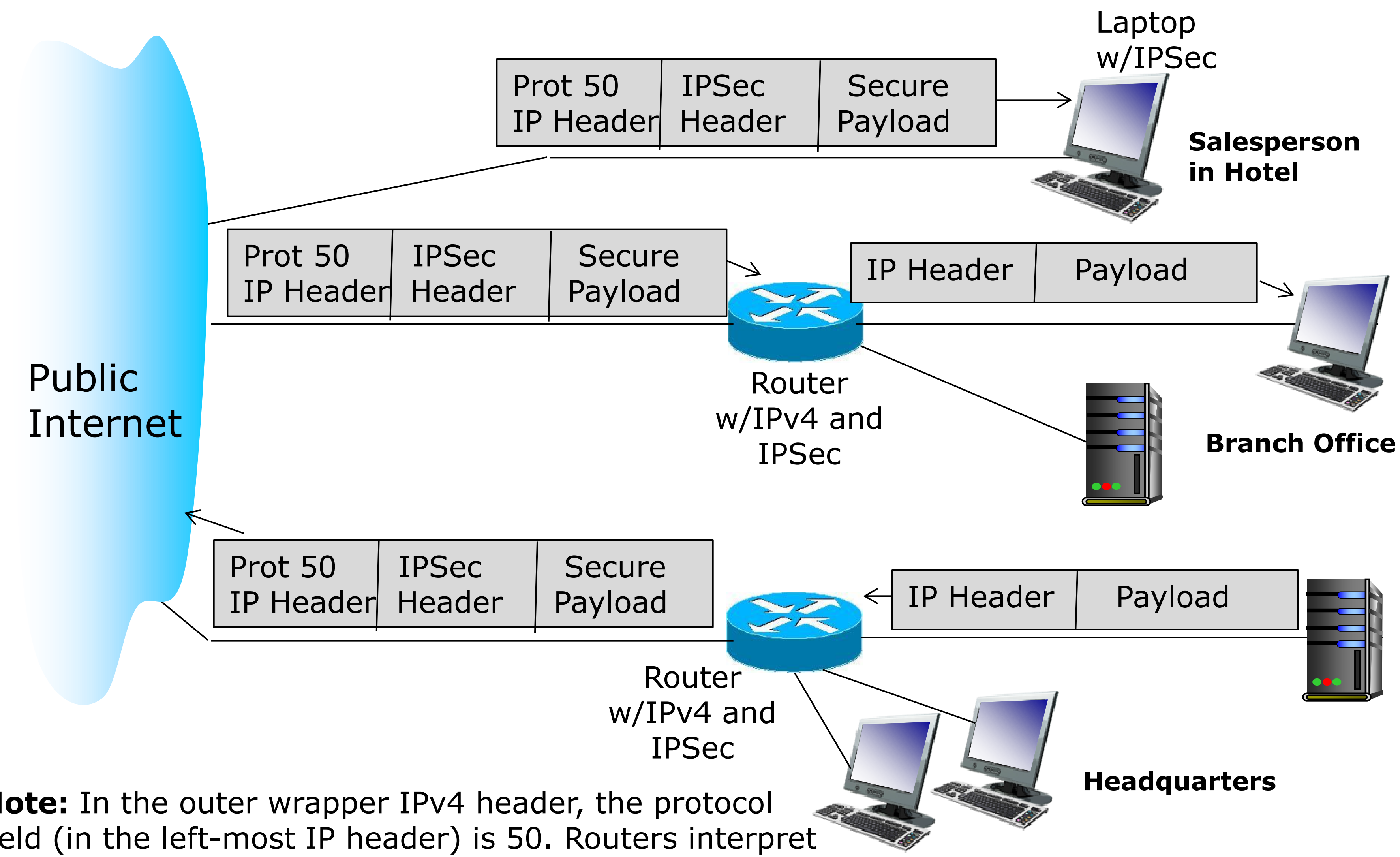
https://www.techopedia.com/definition/30756/layer-2-vpn

Virtual Private Networks (VPNs)

- A VPN allows to extend a L2 or L3 network over a public connection, for example the Internet
- In each private network, a VPN endpoint has to be created
  - Using one of the home routers, and one of the computers
  - Or directly between home routers
- Both routers will connect to Internet, get public IPs.
- ComputerA will then establish VPN tunnel to routerB
- Permanent connection with virtual interfaces at the end
- VPN Tunnel can be Layer 2 or Layer 3
  - In L2 tunnel, both interfaces are bridged
  - In L3 tunnel, traffic is routed between interfaces



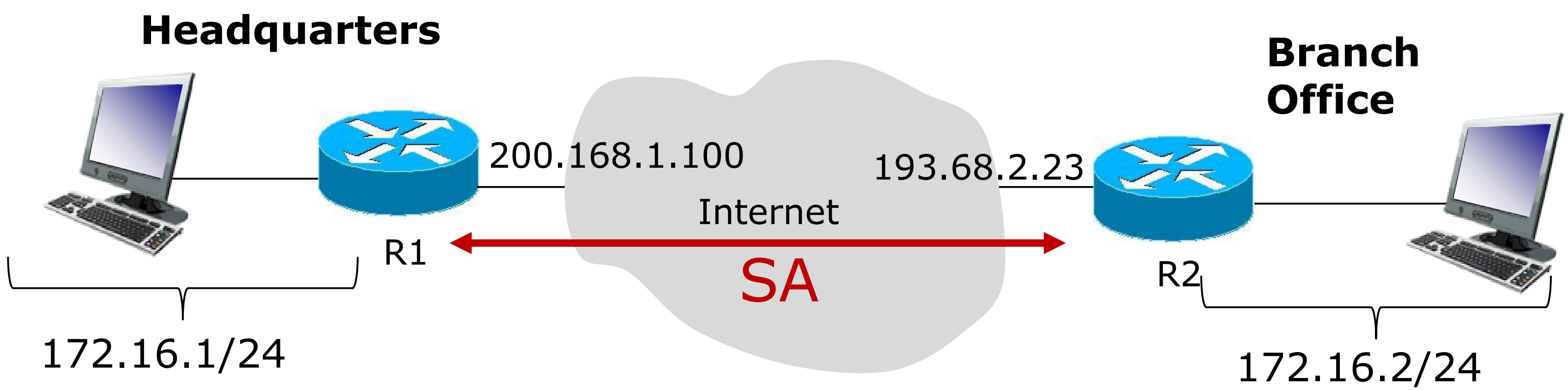
Virtual Private Network (VPN) over IPSec



Note: In the outer wrapper IPv4 header, the protocol field (in the left-most IP header) is 50. Routers interpret this to be an IPSec datagram and process it accordingly.

Security associations (SAs)

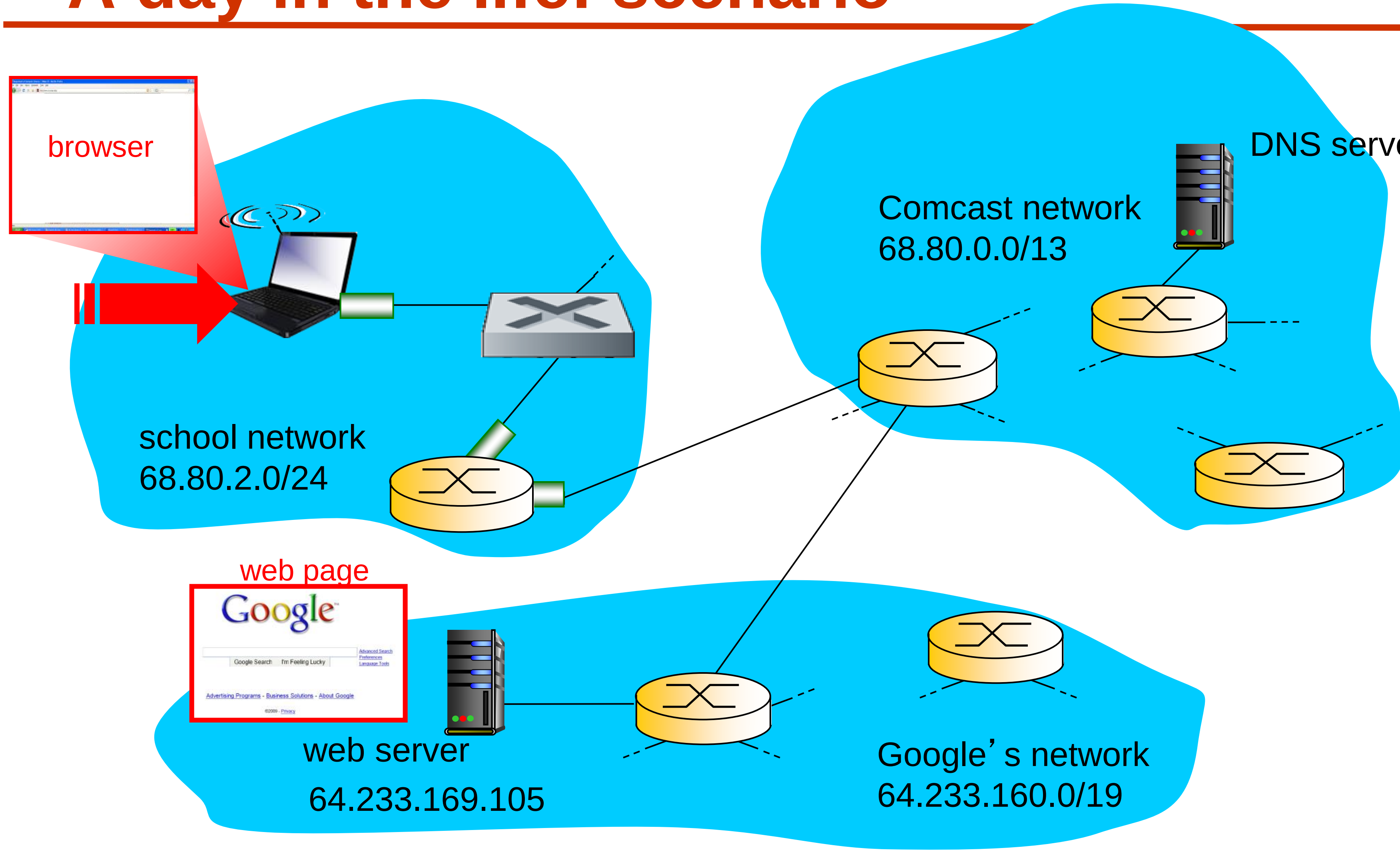
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



- before sending data, "security association (SA)" established from sending to receiving entity
  - SAs are simplex: for only one direction
- ending, receiving entitles maintain *state information* about SA
  - recall: TCP endpoints also maintain state info
  - IP is connectionless; IPsec is connection-oriented!

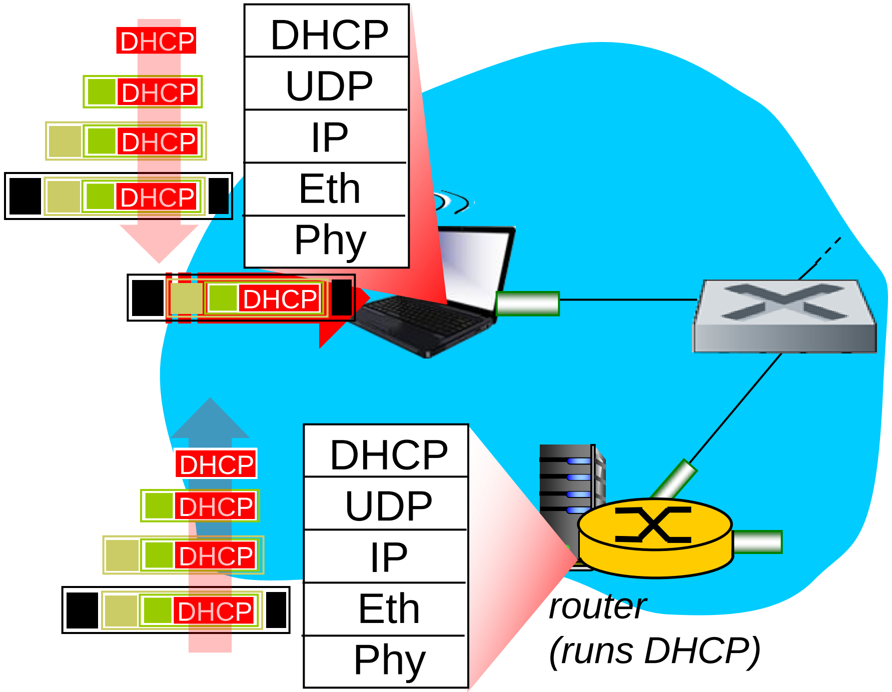
A Day in the life of a web page

A day in the life: scenario



Source: K&R Slide set – Chapter 5

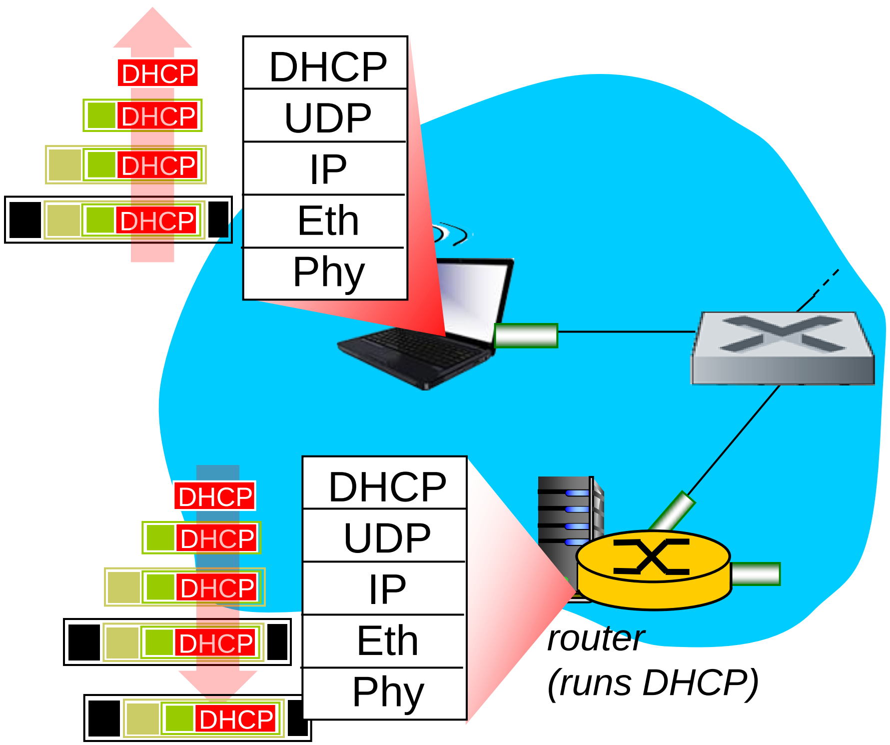
A day in the life… connecting to the Internet



- 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

Source: K&R Slide set – Chapter 5

A day in the life… connecting to the Internet

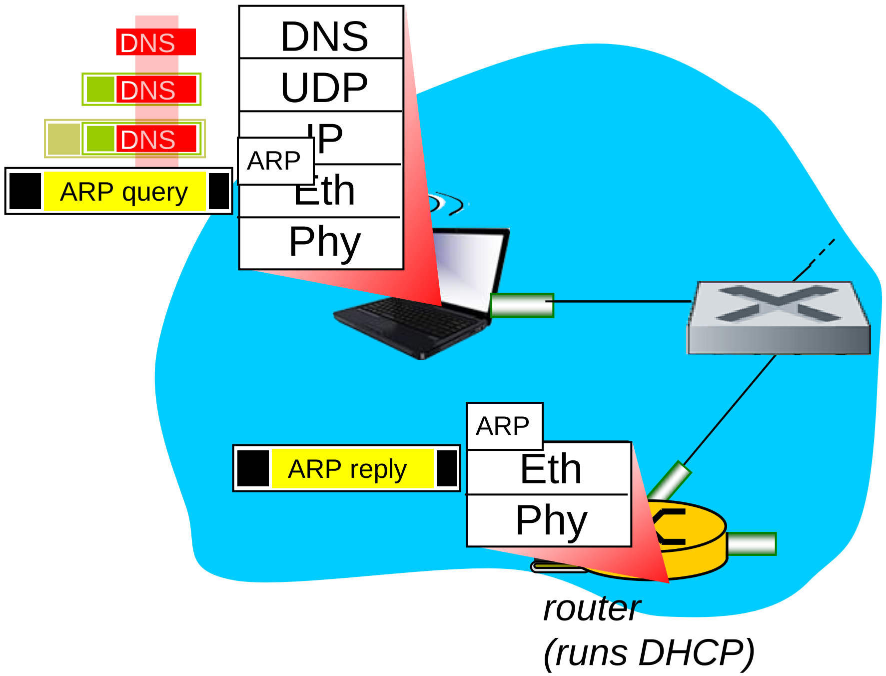


- DHCP server formulates *DHCP ACK* containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
  - encapsulation at DHCP server, frame forwarded (switch learning) through LAN, demultiplexing at client
  - DHCP client receives DHCP ACK reply

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

Source: K&R Slide set – Chapter 5

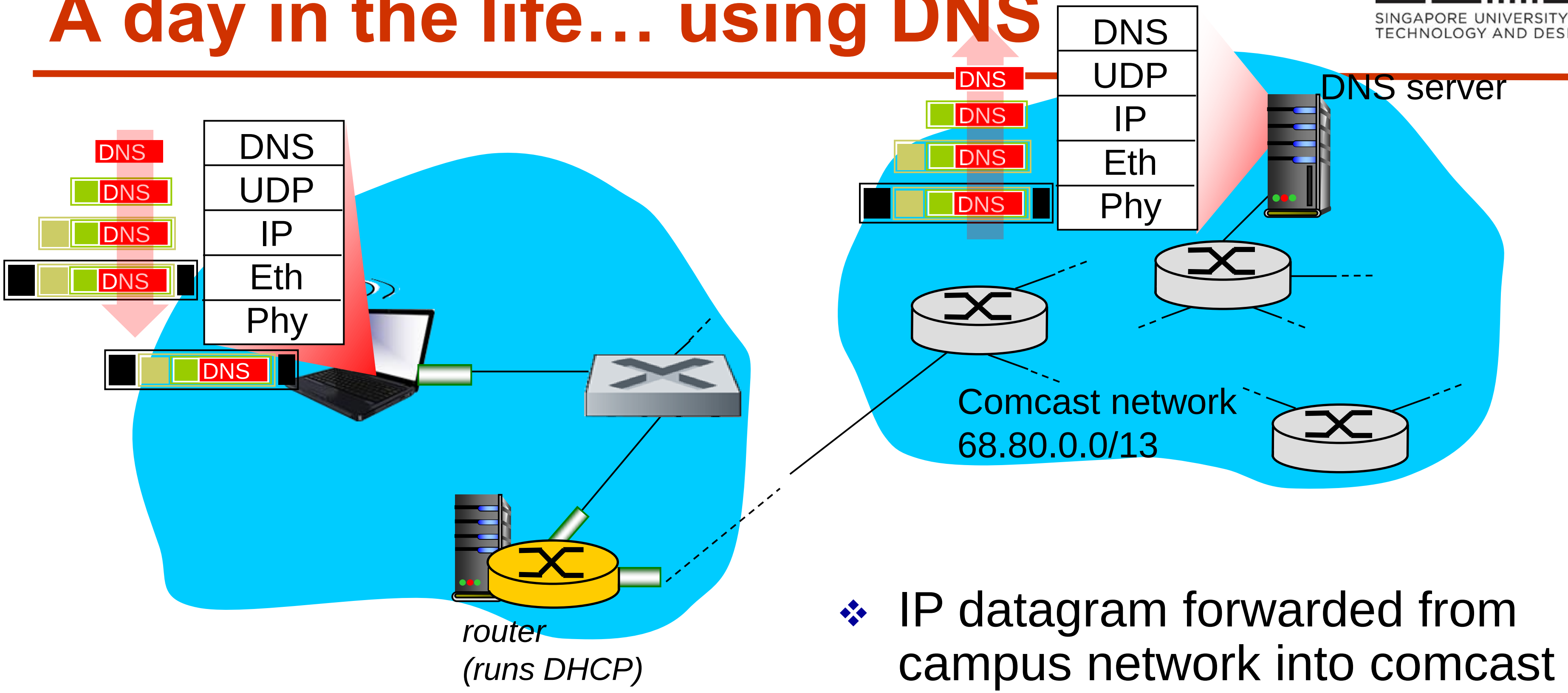
A day in the life… ARP (before DNS, before HTTP)



- 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

Source: K&R Slide set – Chapter 5

A day in the life… using DNS

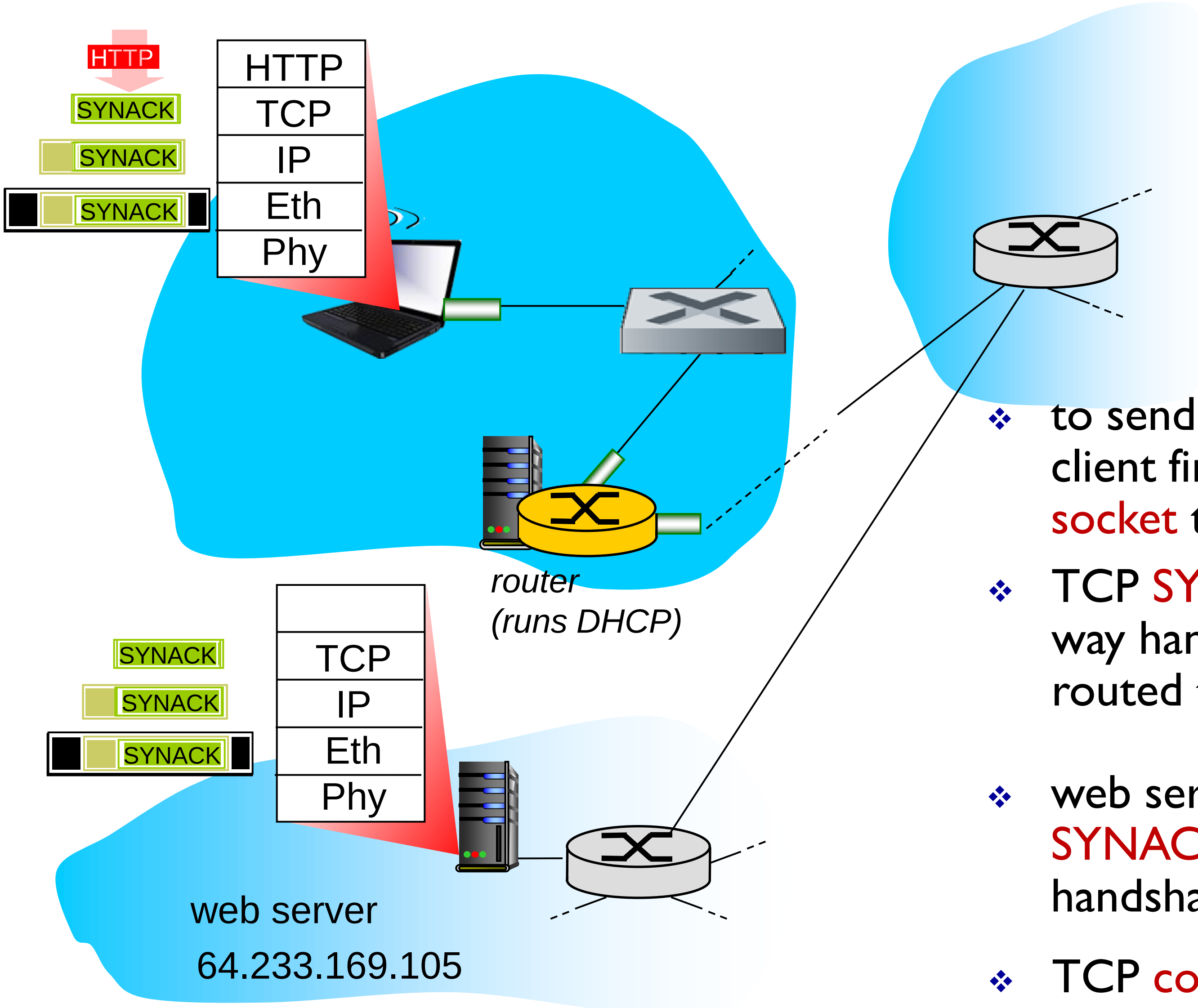


- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

- 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,
- demux'ed to DNS server
- DNS server replies to client with IP address of www.google.com

Source: K&R Slide set – Chapter 5

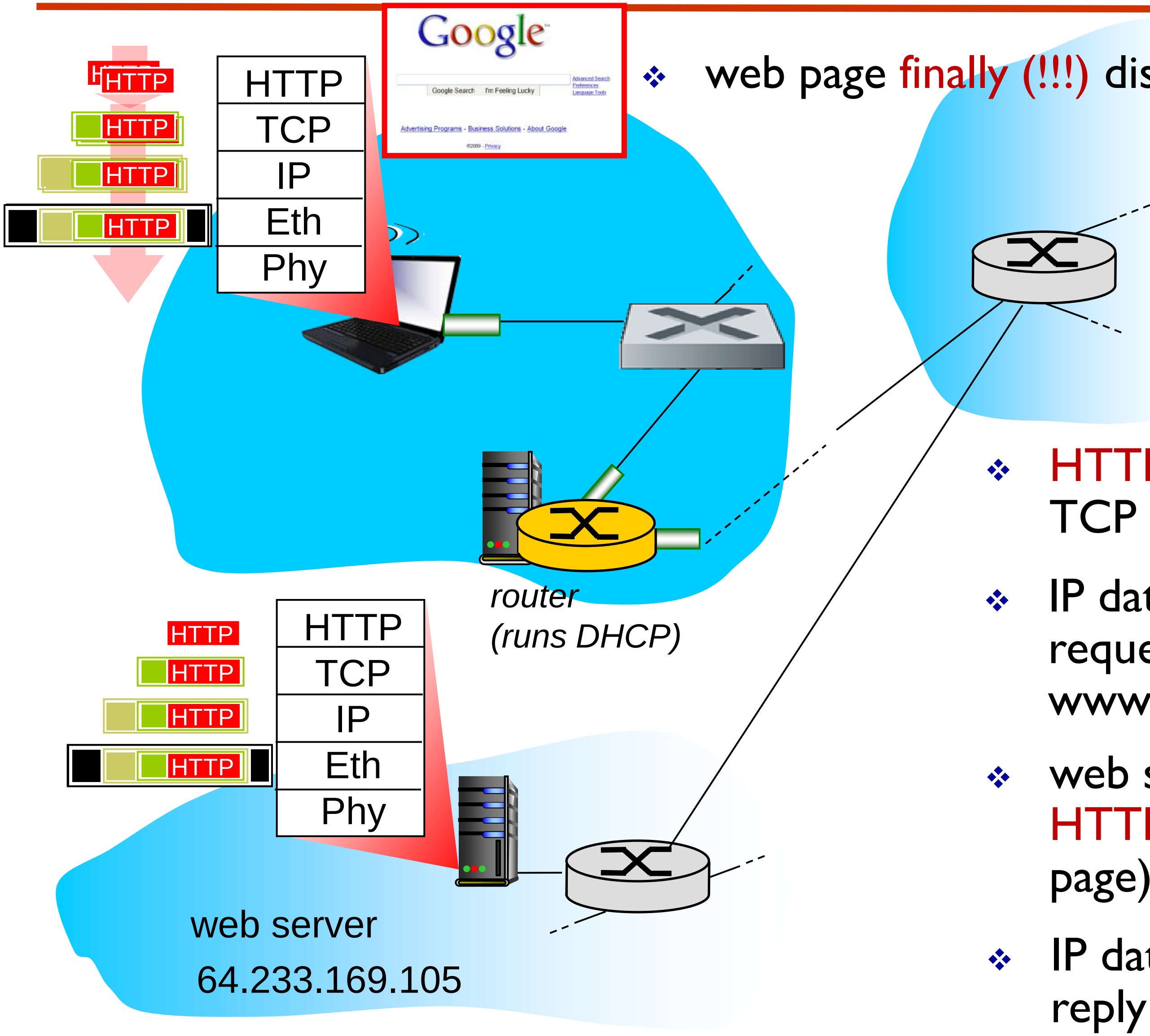
A day in the life…TCP connection carrying HTTP



- 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!

Source: K&R Slide set – Chapter 5

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

Source: K&R Slide set – Chapter 5