CSE 3231
Computer Networks

Chapter 5
The Network Layer
part 3

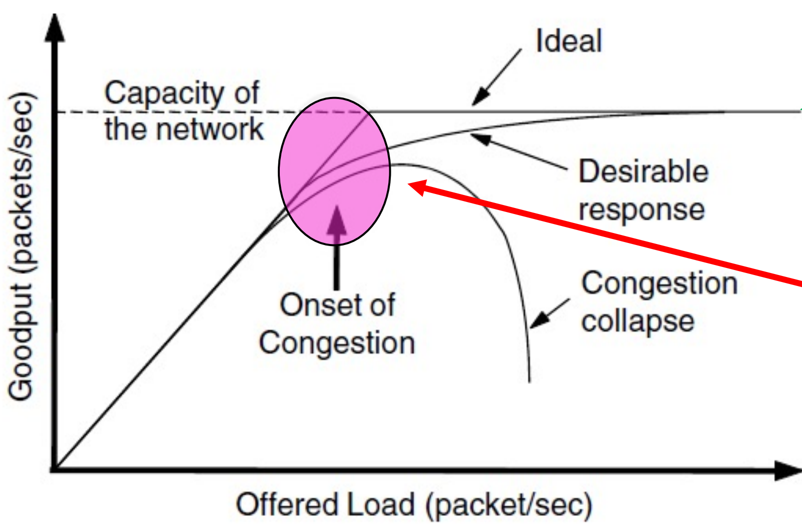
William Allen, PhD

Spring 2022

Congestion Control

Congestion results when too much traffic is entered into the network

- network performance is degraded due to packet losses and retransmissions
- results in *goodput* (number of valid packets delivered) falling far below the number of packets transmitted

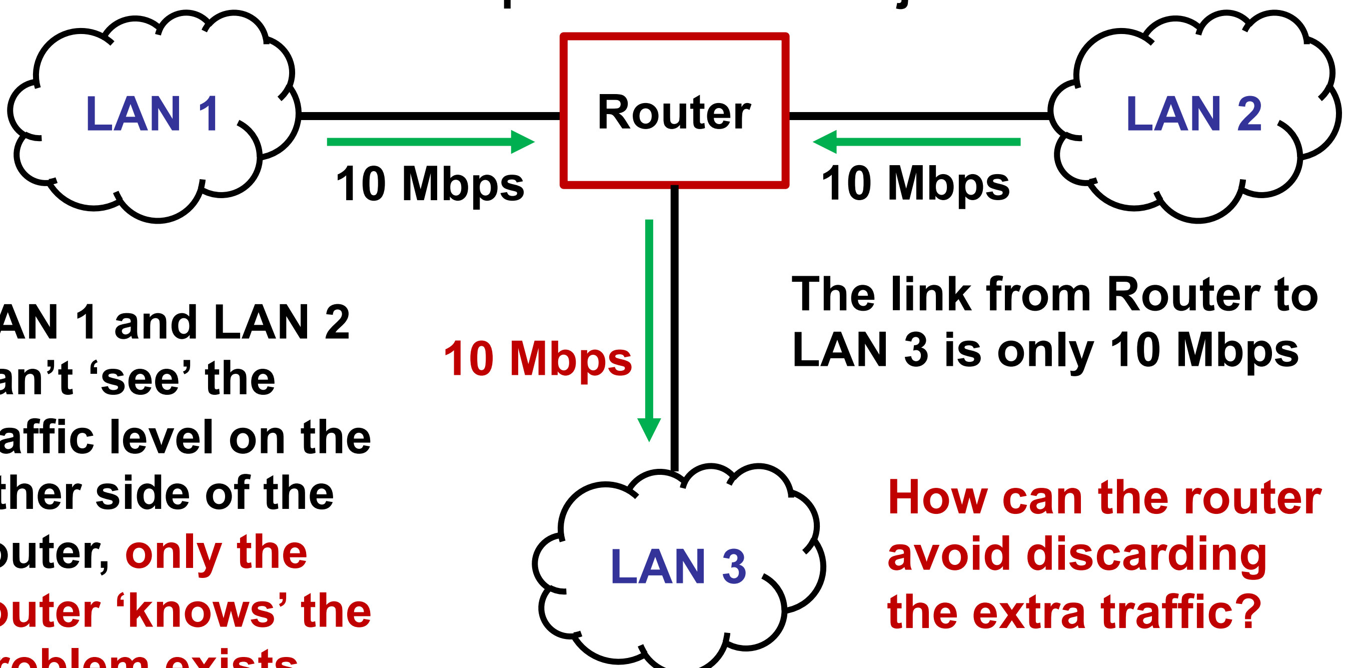


Ideally, *goodput* should increase until the network reaches its capacity

However, *congestion* causes the loss of many packets and a reduction of successful deliveries

Congestion Example

Router is receiving up to 20 Mbps of traffic it needs to send to LAN 3, but it can only forward 10 Mbps. The router's buffer will quickly fill and new packets will be rejected.



LAN 1 and LAN 2 can't 'see' the traffic level on the other side of the router, only the router 'knows' the problem exists.

The link from Router to LAN 3 is only 10 Mbps

How can the router avoid discarding the extra traffic?

Congestion Control

The network and transport layers must work together to handle *congestion*

- We will look at the network layer's part now and the transport layer's part later

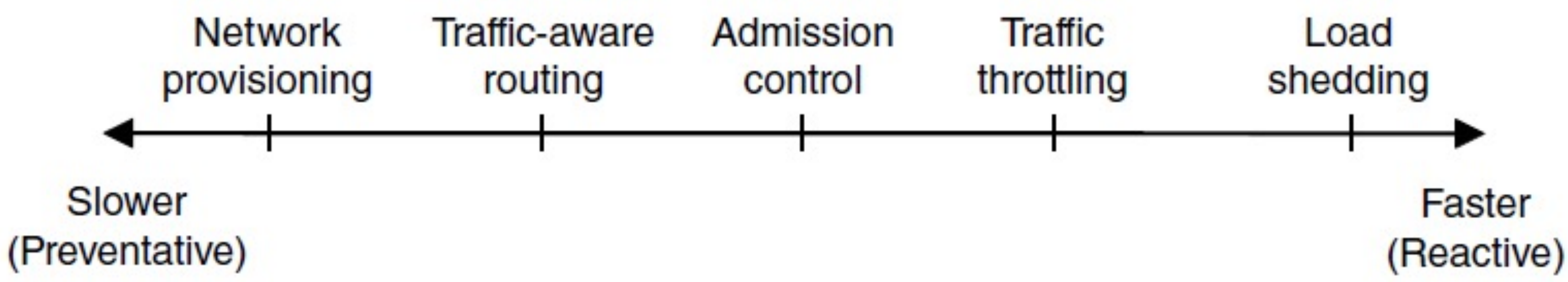
There are a number of possible approaches:

- Traffic-aware routing
- Admission control
- Traffic throttling
- Load shedding

Congestion Control – Approaches

The network must do its best with the traffic load that is transmitted by nodes

- Nodes could reduce the load they present to the network (handled by the transport layer)
- There are several approaches that work at a range of different timescales



Network Provisioning

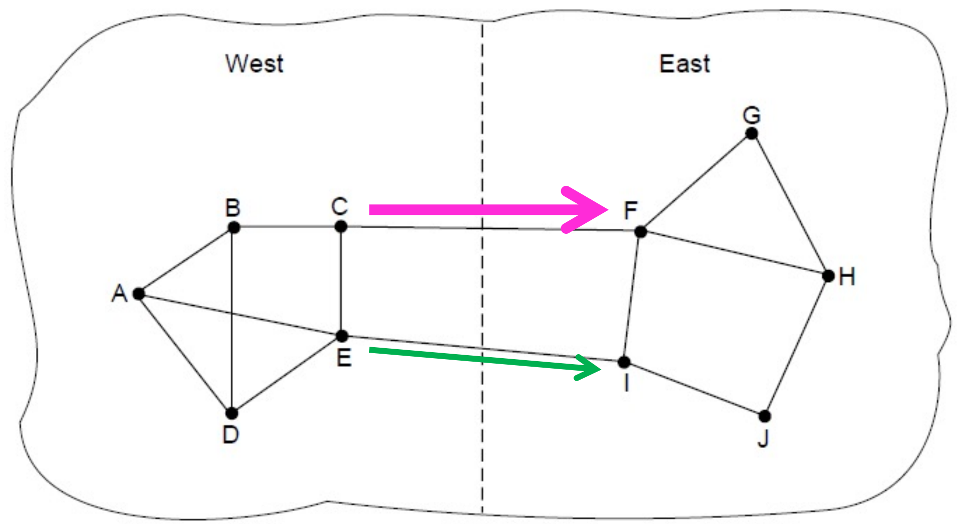
One approach is to provide users with a level of network bandwidth based on how they will use the network, this is called *network provisioning* -- allocating different resources to users based on their needs

- users may receive a certain maximum bandwidth based on the task they are doing
  - some tasks are allocated a limited bandwidth and other tasks are given a larger amount
  - this limits the amount of data a user can send into the network, avoiding overloading & congestion

Traffic-Aware Routing

Choose routes depending on the volume of traffic, not just on the network topology

If the *C to F* path is overloaded, could use the *E to I* path for traffic from the West-to-East zones



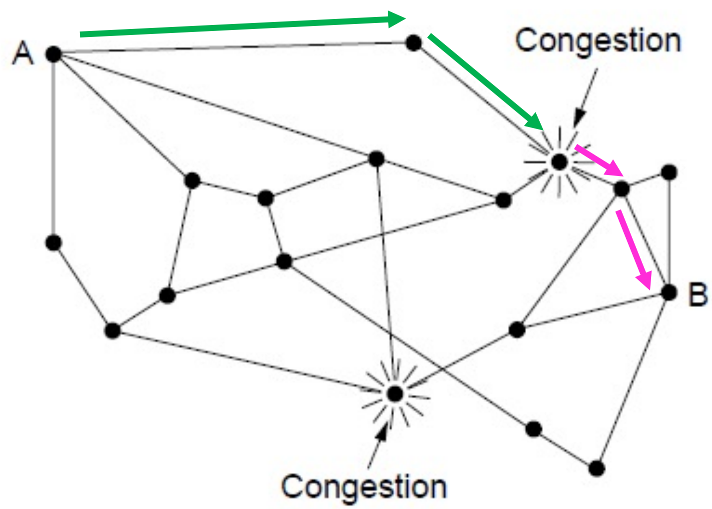
But take care to avoid *oscillations*

- i.e., switching between the two paths too frequently

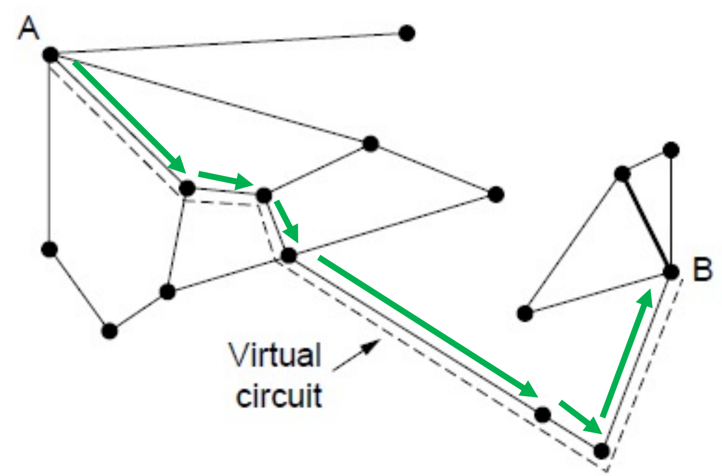
Admission Control

Virtual Circuits can use admission control to only add new traffic if the network has enough capacity

- However, can be difficult to predict future traffic needs
- Can look for an uncongested route for a new circuit



Network with some congested nodes which impacts traffic from A to B



Find an uncongested path and route traffic from *A to B* around congestion

Congestion Avoidance

One way to avoid congestion is to slow the rate of transmission of traffic when congestion is likely, this is referred to as *load shedding*

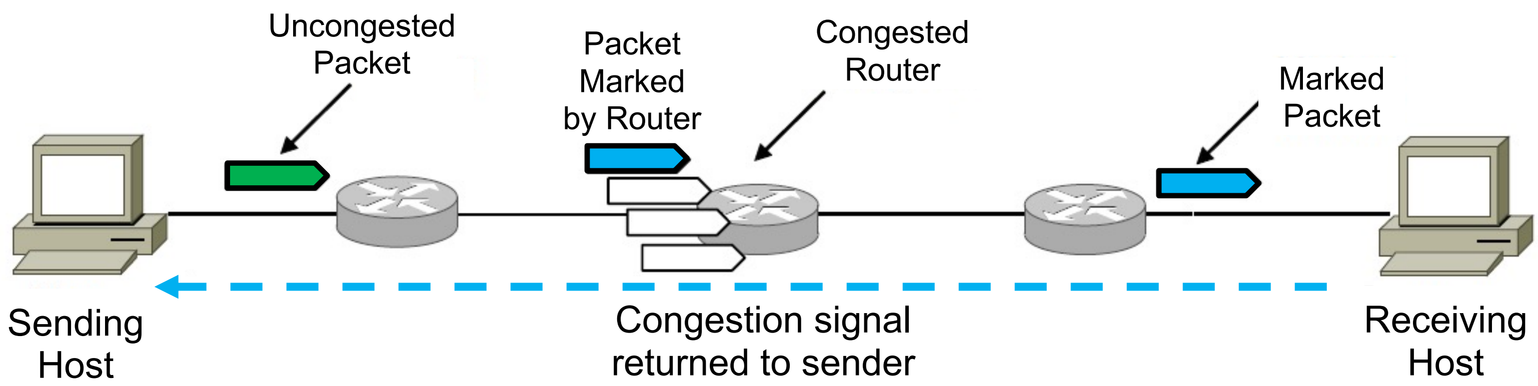
- A good predictor of congestion is the length of time packets sit in a router's buffer (i.e., *queuing delay*)
- Bursts of traffic activity can affect the length of delay so we use an *average of the delay* over time
- An *Exponentially Weighted Moving Average* reduces the impact of bursts on the average delay
  - the equation below is "*tuned*" by adjusting the weight **α** to create a balance between the current delay and past delay

$$delay_{new} = \alpha \cdot delay_{old} + (1-\alpha) \cdot (\text{current queue length})$$

Traffic Throttling

Congested routers can signal hosts to slow the rate of transmission of traffic

- ECN (Explicit Congestion Notification) marks packets suffering from congestion
  - the packet is delivered & receiving host notifies the sender to slow the rate of transmission



Load Shedding

- Another method is for routers to *drop packets* (shedding traffic load) to reduce congestion
  - Senders can track the frequency of dropped packets as an indicator of congestion in the network
  - Unlike traffic throttling, packets do not continue in the network and the receiver does not signal the sender to slow down
  - However, the router could send an ICMP packet to the sender to notify of the dropped packet
  - We will see later that TCP will also react to dropped packets and slow the rate of transmission

Random Early Detection (RED)

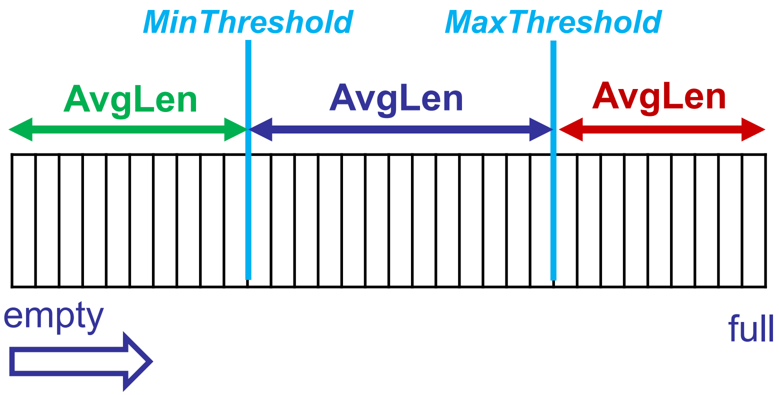
- One approach, *Random Early Detection* (RED), is similar to traffic throttling in that each router monitors its own queue length
- However, RED's goal is for the router to drop packets *before* its buffer space is full to get the source to slow down so that it will not have to drop even more packets later on
  - Therefore, RED's buffer length monitoring process is more complex than what traffic throttling uses
  - RED uses that information to decide when to drop a packet and which packet should be dropped

Random Early Detection (RED)

- To understand the basic idea, consider a simple FIFO queue. Rather than wait for the queue to become completely full and then be forced to drop each arriving packet, we decide whether to drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*
- This idea is called *early random drop*. The RED algorithm defines how to monitor the queue length and when to drop a packet

Random Early Detection (RED)

- First, RED computes an average queue length (*AvgLen*) using a weighted average similar to the traffic throttling method
- Second, RED has two queue length thresholds that trigger certain activity: *MinThreshold* and *MaxThreshold*
- When a packet arrives at the gateway, RED compares the current *AvgLen* with these two thresholds, according to the following rules:



if *AvgLen* ≤ *MinThreshold*

→ add the packet to the queue

if *MinThreshold* < *AvgLen* < *MaxThreshold*

→ calculate probability *P*

→ drop the arriving packet with probability *P*

if *MaxThreshold* ≤ *AvgLen*

→ drop the arriving packet

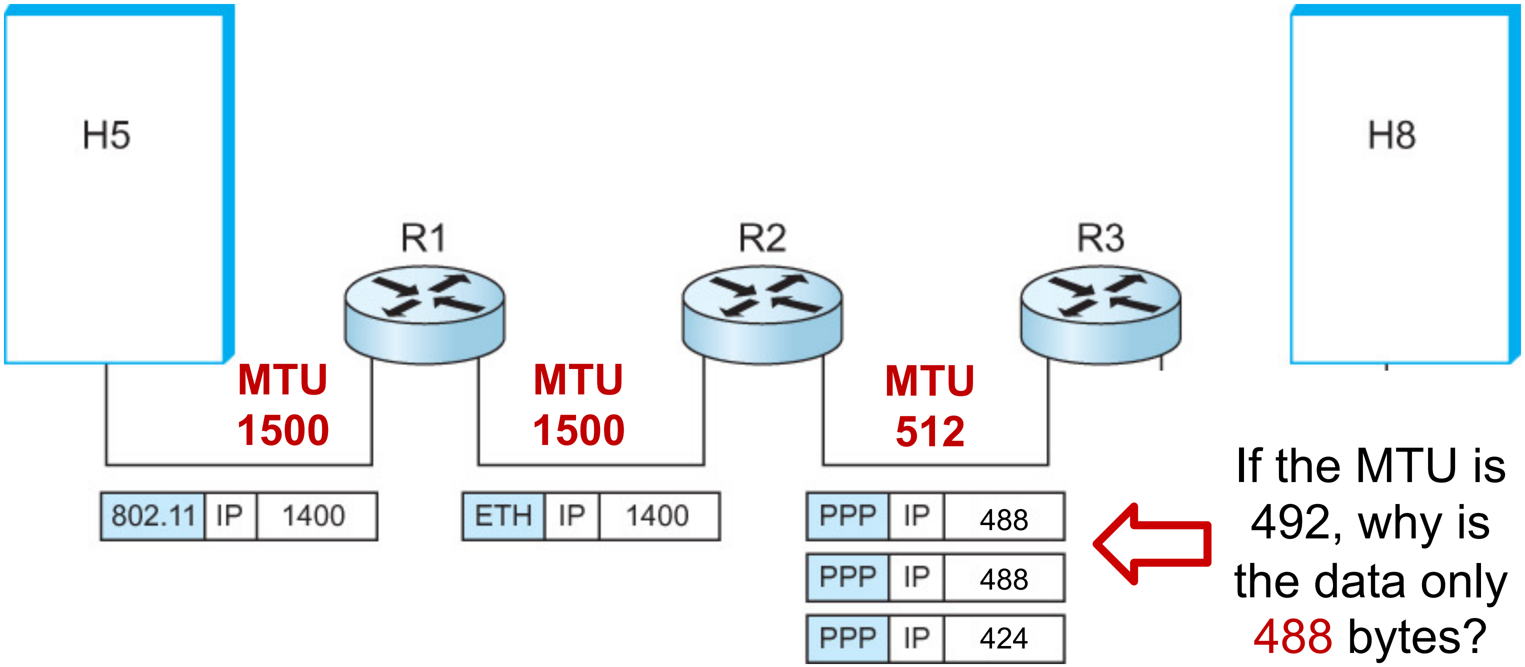
Random Early Detection (RED)

- The more a host transmits, the more likely it is that its packets are dropped
- The value of the probability *P* is based on the “burstiness” of recent traffic
  - If recent traffic often increased quickly, more room is needed in the queue to deal with it and *P* is higher to cause more packet drops
  - If recent traffic increased slowly, a lower *P* will allow the queue to get closer to full with less risk of it becoming completely full

IP Fragmentation and Reassembly

- Links have an MTU (Maximum Transmission Unit), i.e., the largest frame size that link can carry
  - Ethernet (1500 bytes), FDDI (4500 bytes)
  - *What if the next link to use has a smaller MTU?*
- Solved by *fragmentation*
  - Fragmentation occurs at a router when it receives a packet which is larger than the MTU of the *next link*
  - All fragments of a packet have their own header and carry the same identifier number in the *Ident* field
  - IP does not recover from missing fragments, if any fragments are lost, the packet is discarded
  - If a packet is fragmented, it stays fragmented until reassembly is done by the receiving host

IP Fragmentation and Reassembly



If the MTU is 492, why is the data only 488 bytes?

IP datagrams traversing a sequence of physical networks

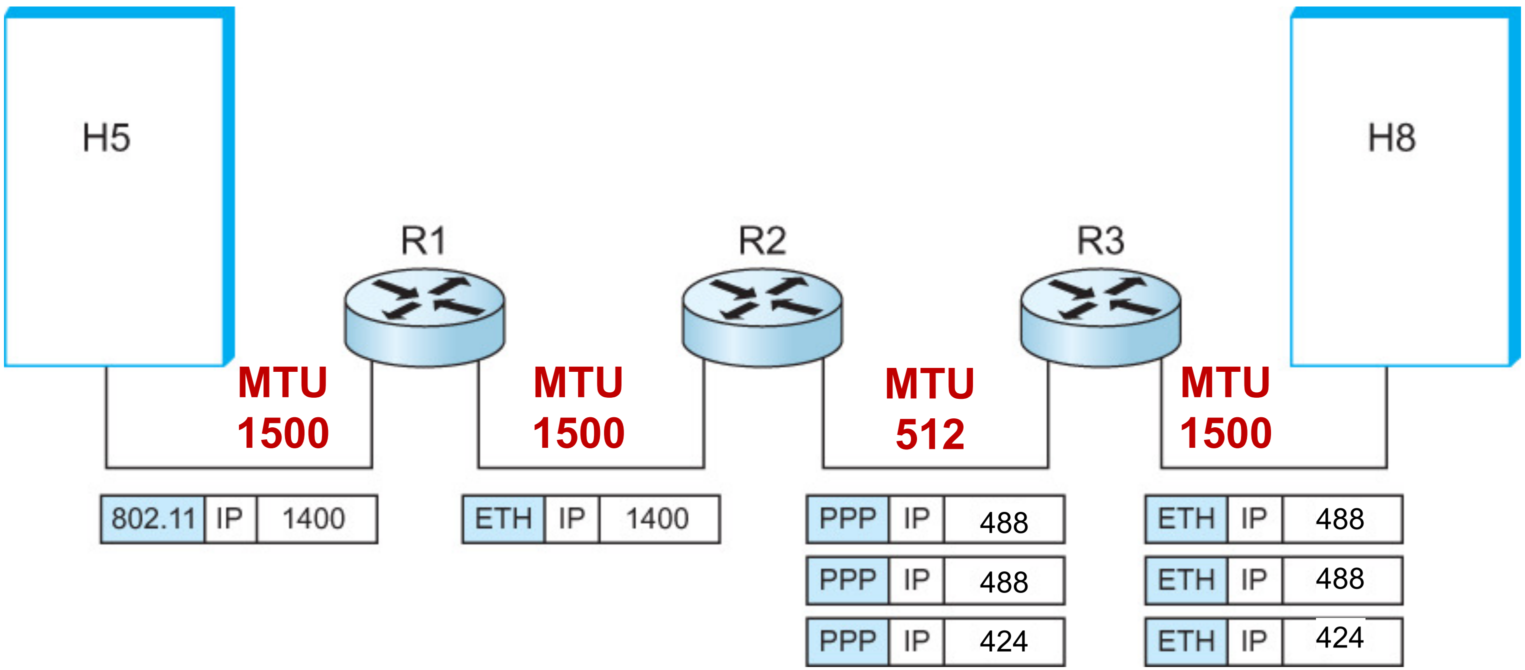
If the MTU < packet size, packets must be fragmented

– What will be the size of the fragment's payload?
  - IP Header is 20 bytes, leaving ≤ 492 for data

IP Fragmentation Offset

- The IP packet length field is 16 bits
  - that allows packets of up to 65,536 bytes
  - the *offset* value is used to indicate where each fragment was located in the original packet
- However, the offset field is only 13 bits in length and this imposes a restriction
  - the size of fragmented packets must be a multiple of 8 bytes (8, 16, etc.)
  - thus, the fragmentation offset field contains the number of 8-byte blocks in the packet
    - in the example, the MTU is 512 bytes, leaving 492 bytes for the payload of fragment 1: 61 → 61*8 = 488 bytes while 62*8 = 496 which is > 492

IP Fragmentation and Reassembly



IP datagrams traversing a sequence of physical networks

If the MTU < packet size, packets must be fragmented

– What will be the size of the fragment's payload?
  - IP Header is 20 bytes, leaving ≤ 492 for data
  - Even if the rest of the links have an MTU ≥ packet size, they are *not* reassembled until they reach the destination host.

IP Fragmentation and Reassembly

Header fields used in IP fragmentation:

- *Ident* (the same for all fragments)
- *Offset* (number of 8-byte blocks)
- One Flags bit is labeled "*MF* "
  - means "More Fragments"
  - set to 1 for all *but* the last fragment

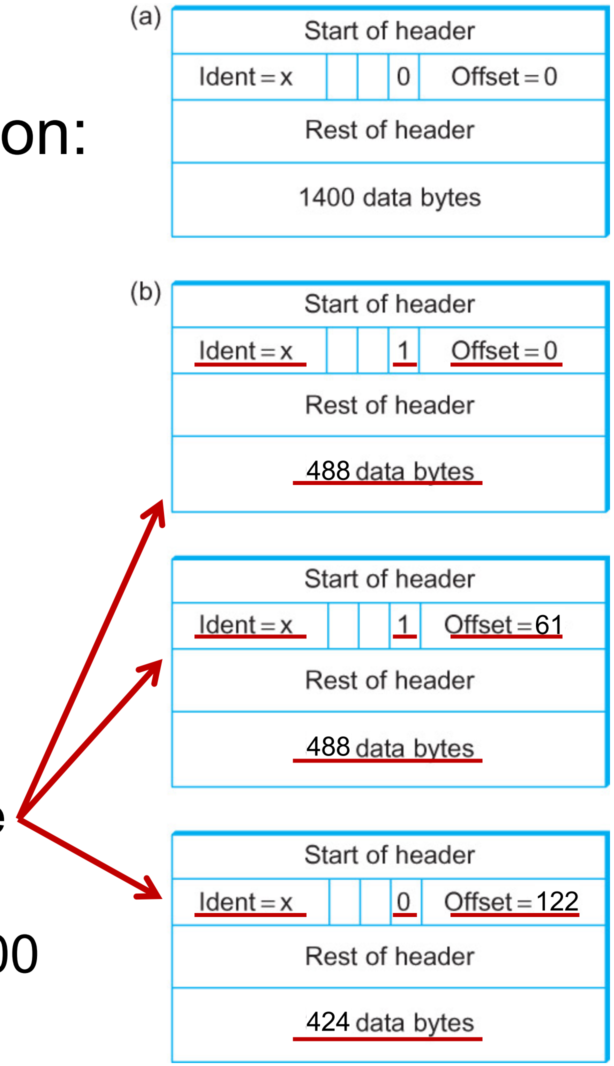
Example:

(a) Un-fragmented packet

(b) Packet after fragmentation into three smaller packets

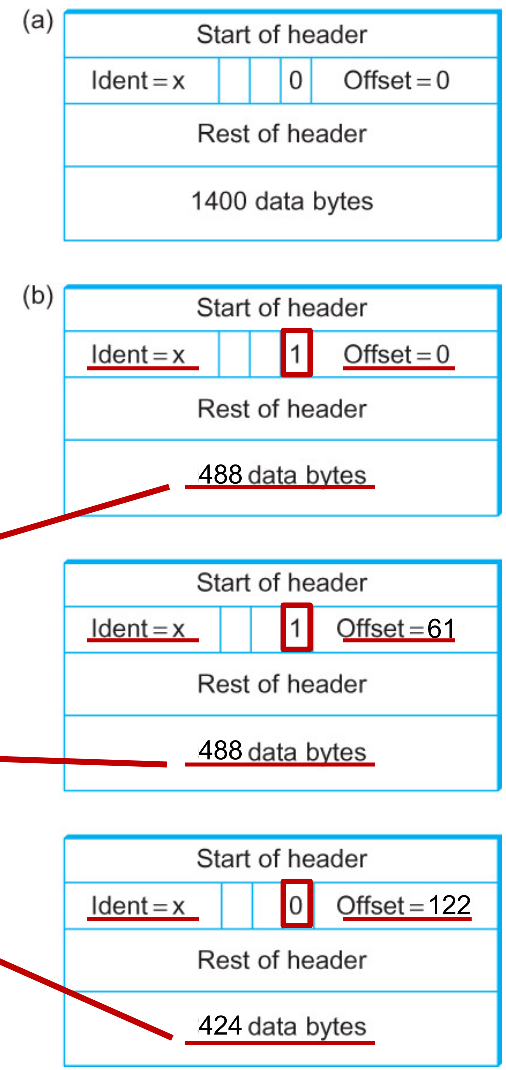
total data size: 488+488+424 = 1400

(488 => 61 blocks * 8 bytes)



IP Fragmentation and Reassembly

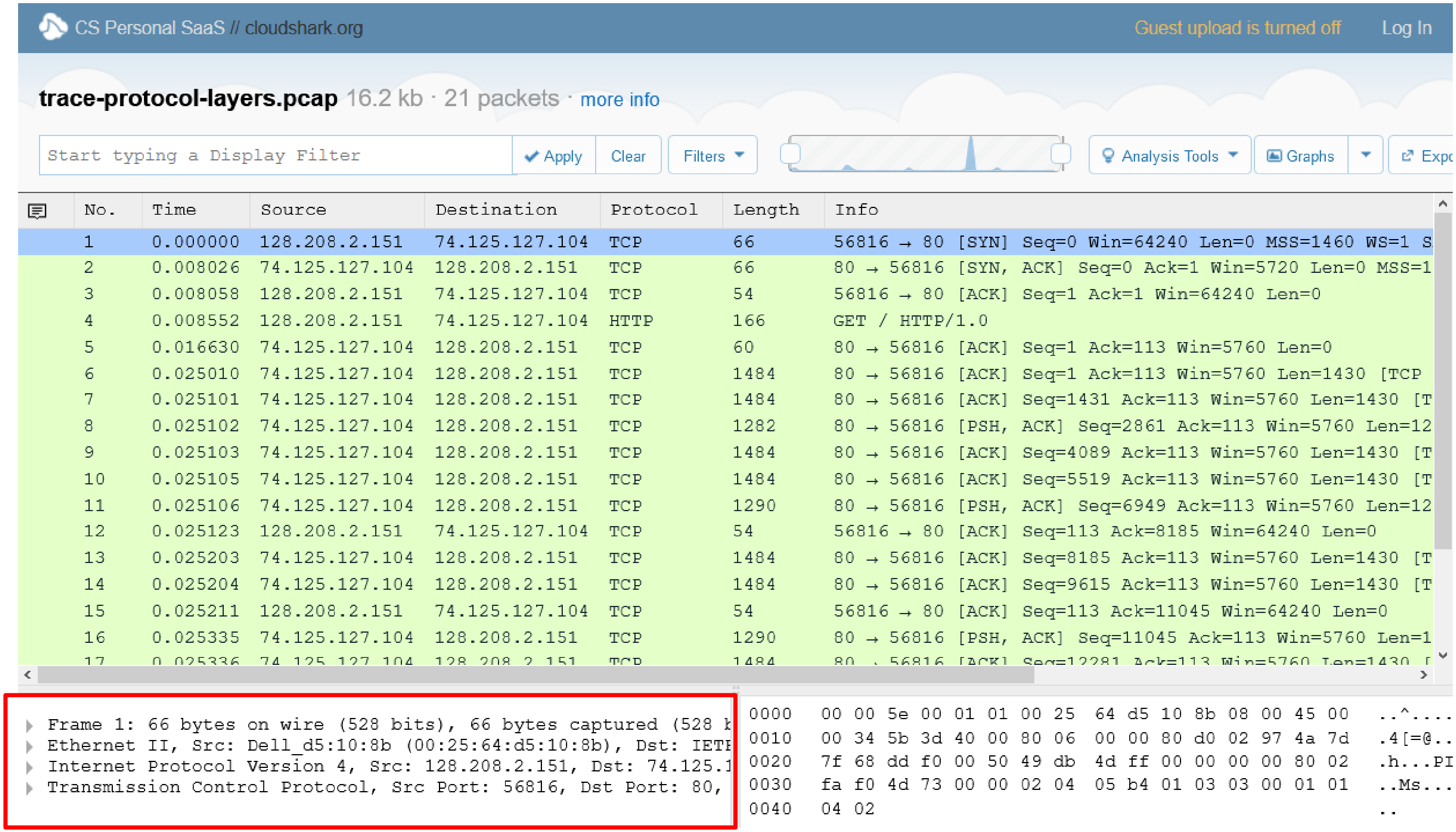
When the fragments reach the final destination, the data (payload) from each packet is stored in a memory buffer until all fragments are received and the complete packet can be delivered to the application



Size:	488	488	424
Offset:	0	8*61	8*122
	0	488	976

Total Size: 976+424 = 1400 bytes

IP Packet Header



IP Fragmentation

```
Internet Protocol Version 4, Src: 192.168.0.114, Dst: 192.168.0.193
  Identification: 0x61d1 (25041)
  Flags: 0x2000, More fragments
      0... .... .... .... = Reserved bit: Not set
      .0.. .... .... .... = Don't fragment: Not set
      ..1. .... .... .... = More fragments: Set
  ...0 0000 0000 0000 = Fragment offset: 0
```

```
Internet Protocol Version 4, Src: 192.168.0.114, Dst: 192.168.0.193
  Identification: 0x61d1 (25041)
  Flags: 0x20b9, More fragments
      0... .... .... .... = Reserved bit: Not set
      .0.. .... .... .... = Don't fragment: Not set
      ..1. .... .... .... = More fragments: Set
  ...0 0101 1100 1000 = Fragment offset: 1480
```

Offset is 0x0B9
=185 decimal
185*8=1480

```
Internet Protocol Version 4, Src: 192.168.0.114, Dst: 192.168.0.193
  Identification: 0x61d1 (25041)
  Flags: 0x0172
      0... .... .... .... = Reserved bit: Not set
      .0.. .... .... .... = Don't fragment: Not set
      ..0. .... .... .... = More fragments: Not set
  ...0 1011 1001 0000 = Fragment offset: 2960
```

Offset is 0x172
=370 decimal
370*8=2960

Last Fragment

Avoiding Fragmentation

- The fragmentation Flag field contains 3 bits:

bit 0	**bit 1**	**bit 2**
not used	DF: don’t fragment	MF: more fragments coming

  – If the DF flag = 1, routers won’t fragment the packet, but if it doesn’t fit in the next link it will be discarded

- The sending node can limit the maximum packet size to the smallest MTU in the packet’s path to avoid fragmentation occurring in routers along that path
  – however, it must determine that minimum MTU before sending packets

IP Packet Header

```
Internet Protocol
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x5b3d (23357)
  ▾ Flags: 0x4000, Don't fragment
        0... .... .... .... = Reserved bit: Not set
        .1.. .... .... .... = Don't fragment: Set
        ..0. .... .... .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
    Time to live: 128
    Protocol: TCP (6)
    Header checksum: 0x0000 [validation disabled]
    Header checksum status: Unverified
    Source: 128.208.2.151
    Destination: 74.125.127.104
```

Infrastructure Services

- Domain Name Service (DNS)
  - We have been using IP addresses to identify hosts
  - While perfectly suited for processing by routers, IP addresses are not exactly user-friendly
  - It is for this reason that a unique ***host name*** is typically assigned to each host in a network

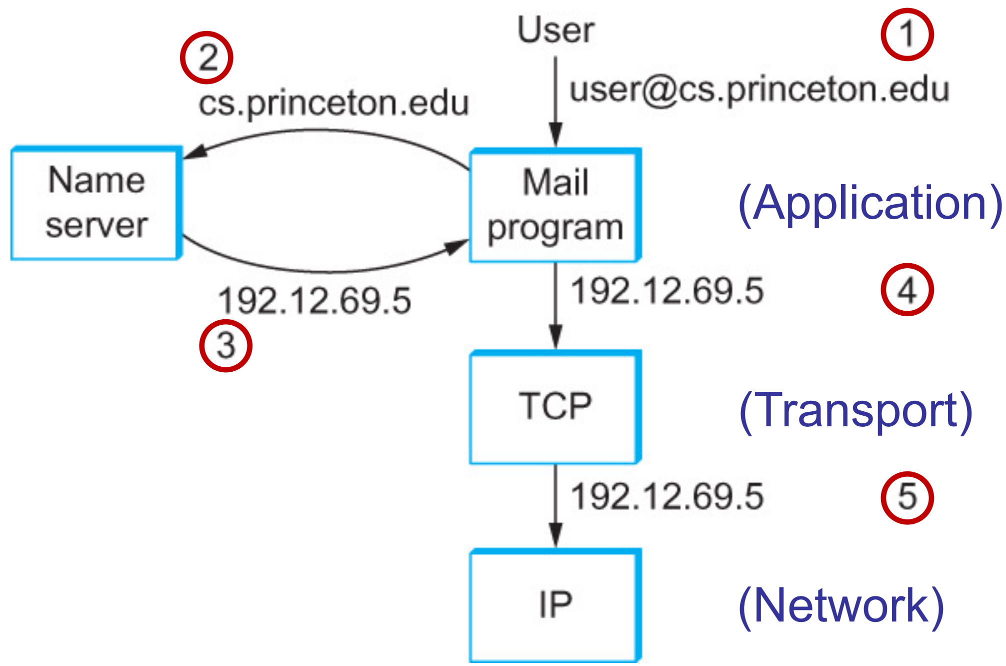
    domain name: www.fit.edu IP address: 18.215.123.220
  - Host names differ from host IP addresses in two important ways
    - First, they are usually text rather than numbers, thereby making them easier for humans to remember
    - Second, names typically contain no information that helps the network locate, or route packets toward, the host

Infrastructure Services

- Domain Name Service terminology:
  - A *name space defines the set of possible names*
    - A name space can be either *flat (names are not divisible), or it can* be *hierarchical (with ways to group names)*
  - The DNS *naming system* maintains a collection of bindings of names to values
    - the term *name* refers to the text name of the resource
    - the *value* can be anything, but in this case it is an IP address
  - A *resolution mechanism* is a procedure that, when given a name, returns the corresponding value
    - A *name server* is *a* resolution mechanism that is available on a network and that can be queried by sending it a message

Infrastructure Services

Domain Name Service (DNS)



Names translated into addresses, where the numbers 1–5 show the sequence of steps in the process

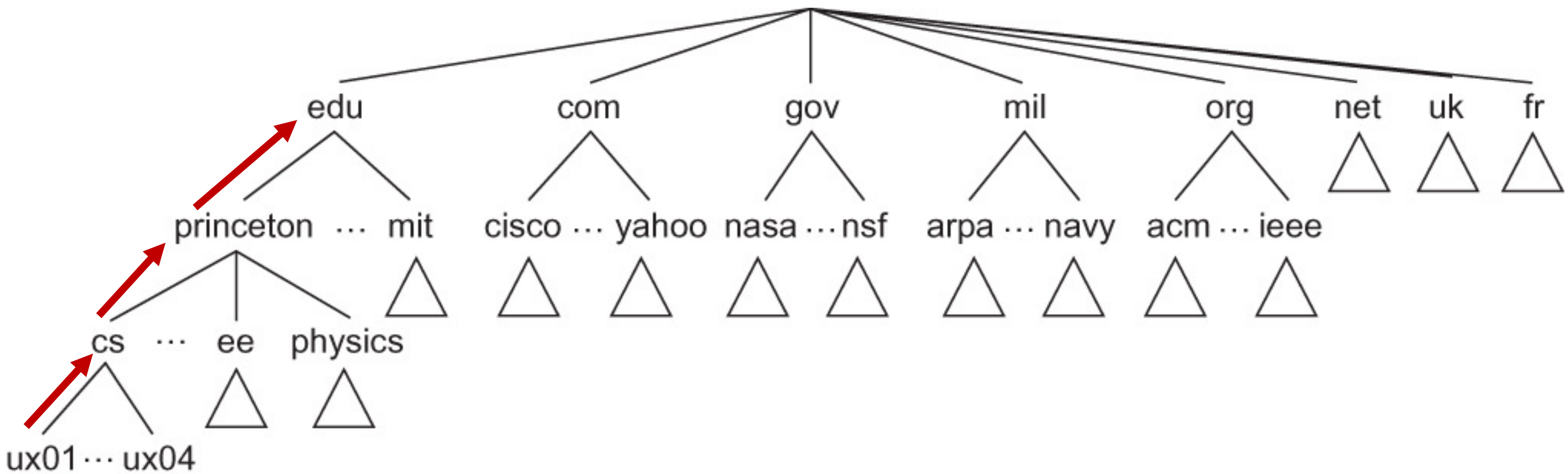
Infrastructure Services

Domain Hierarchy

- DNS implements a hierarchical name space for Internet objects. Unlike Unix/Linux file names, which are processed from left to right with the naming components separated with slashes, DNS names are *processed from right to left* and use periods as the separator
- Like a file hierarchy, the DNS hierarchy can be visualized as a tree, where each node in the tree corresponds to a *domain*, and the leaves in the tree correspond to the hosts being named

Infrastructure Services

Domain Hierarchy



Example of a domain hierarchy

host ux01 is part of the cs domain, which is at Princeton, an edu cational institution. name: **ux01.cs.princeton.edu**

Infrastructure Services

Name Servers

- The complete domain name hierarchy is an abstract concept; it is implemented by dividing the hierarchy into subtrees called *zones*
  - Each zone can be thought of as some administrative authority that is responsible for a portion of the hierarchy
- For example, the top level of the hierarchy forms a zone that is managed by the Internet Corporation for Assigned Names and Numbers (ICANN)

Infrastructure Services

Name Servers

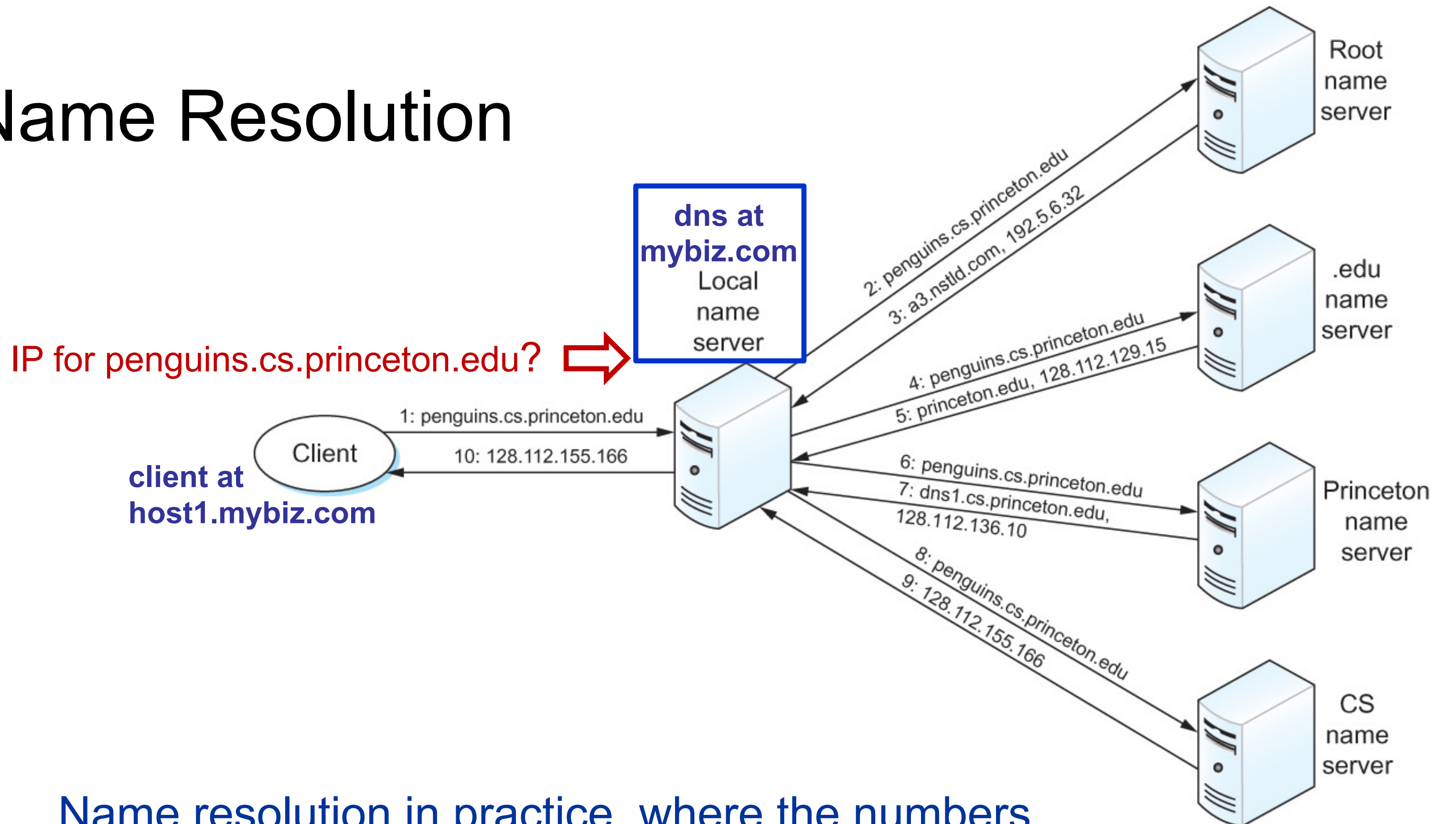
- Each name server implements the zone information as a collection of *resource records* stored in a memory cache
- In essence, a resource record is a *name-to-value binding*, or more specifically a 5-tuple that contains the following fields:

  `<Name, Value, Type, Class, TTL>`

  - Name and Value fields have been discussed, the Type field tells how to use the Value, Class describes the kind of network, TTL is time to live

Infrastructure Services

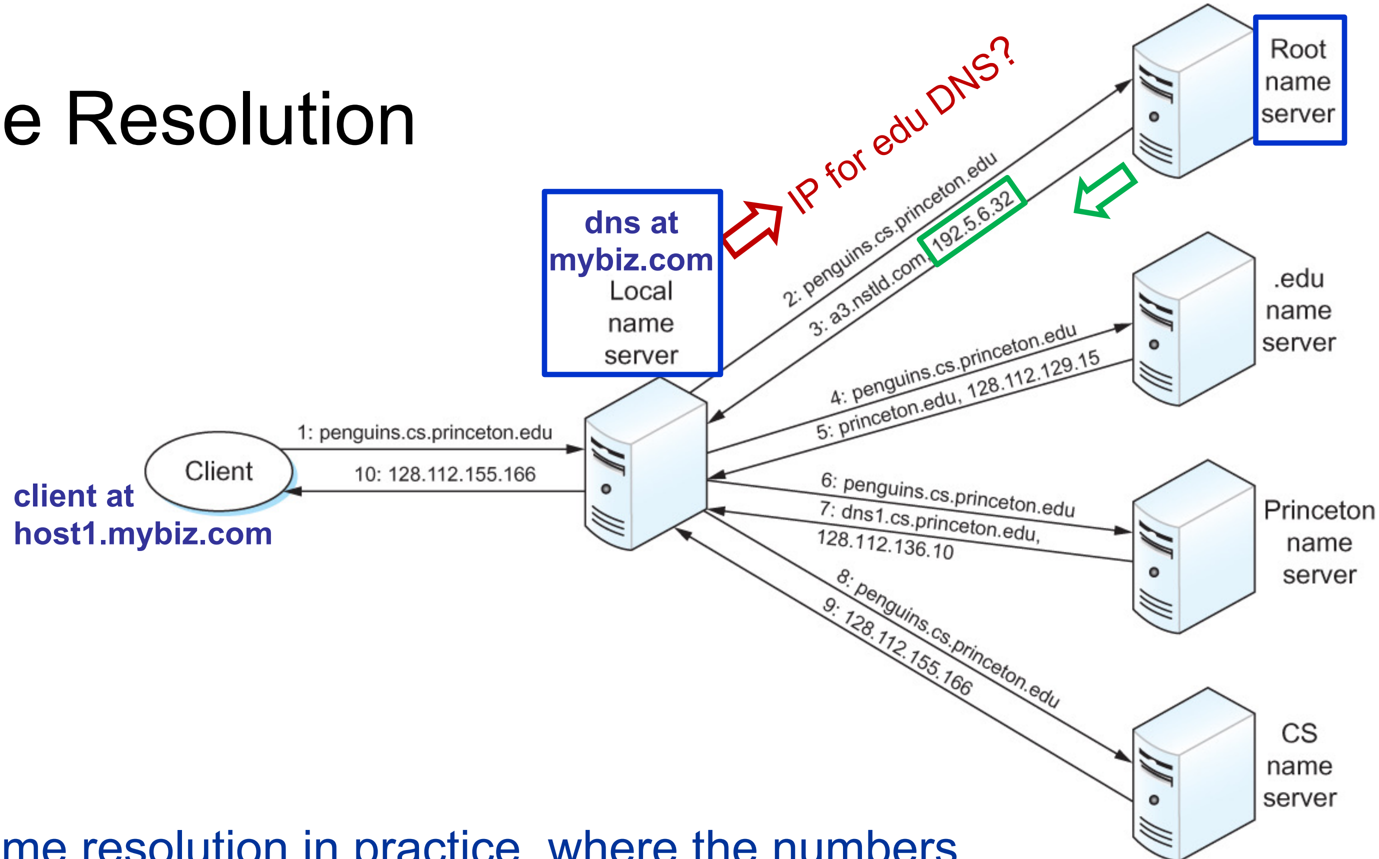
Name Resolution



Name resolution in practice, where the numbers 1–10 show the sequence of steps in the process

Infrastructure Services

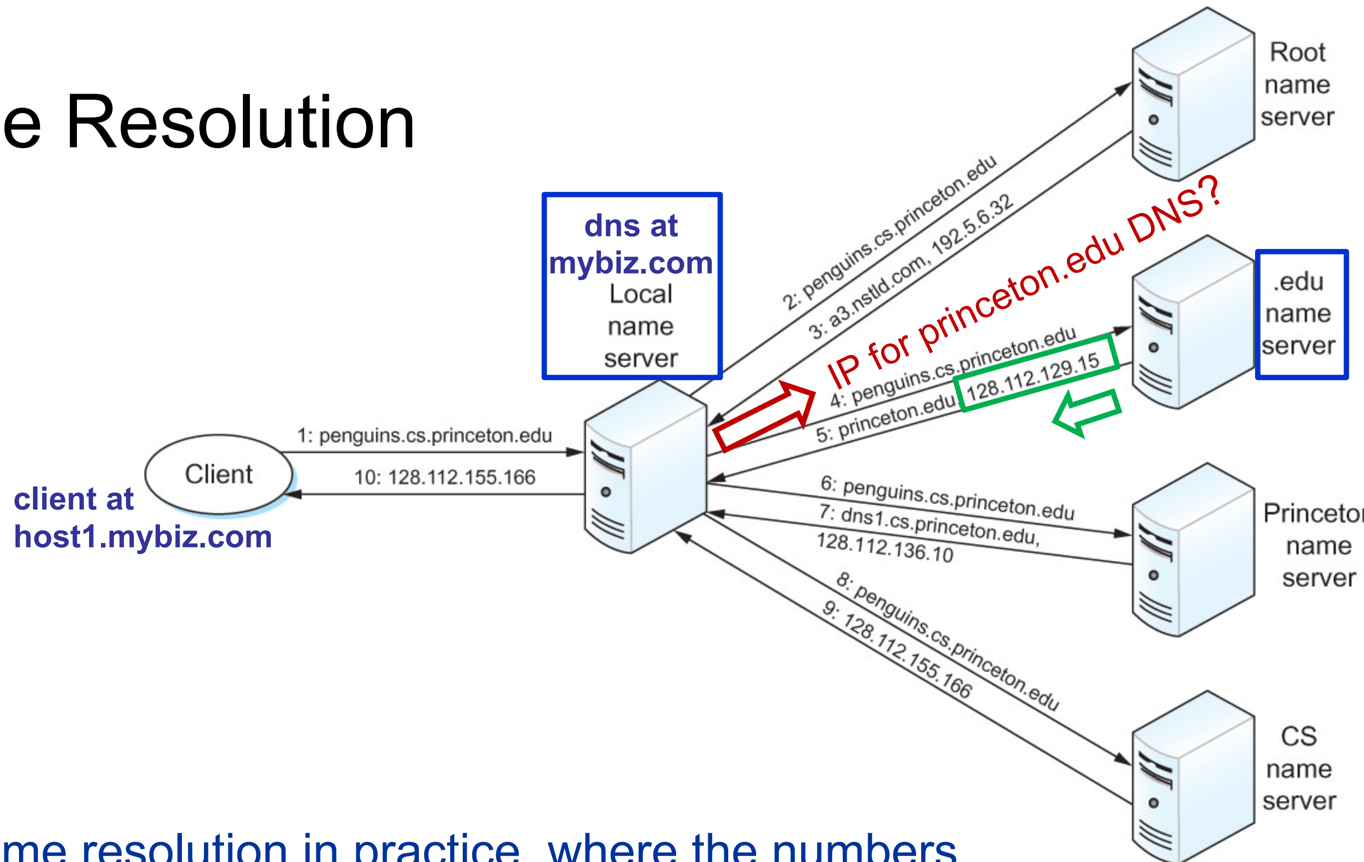
Name Resolution



Name resolution in practice, where the numbers 1–10 show the sequence of steps in the process

Infrastructure Services

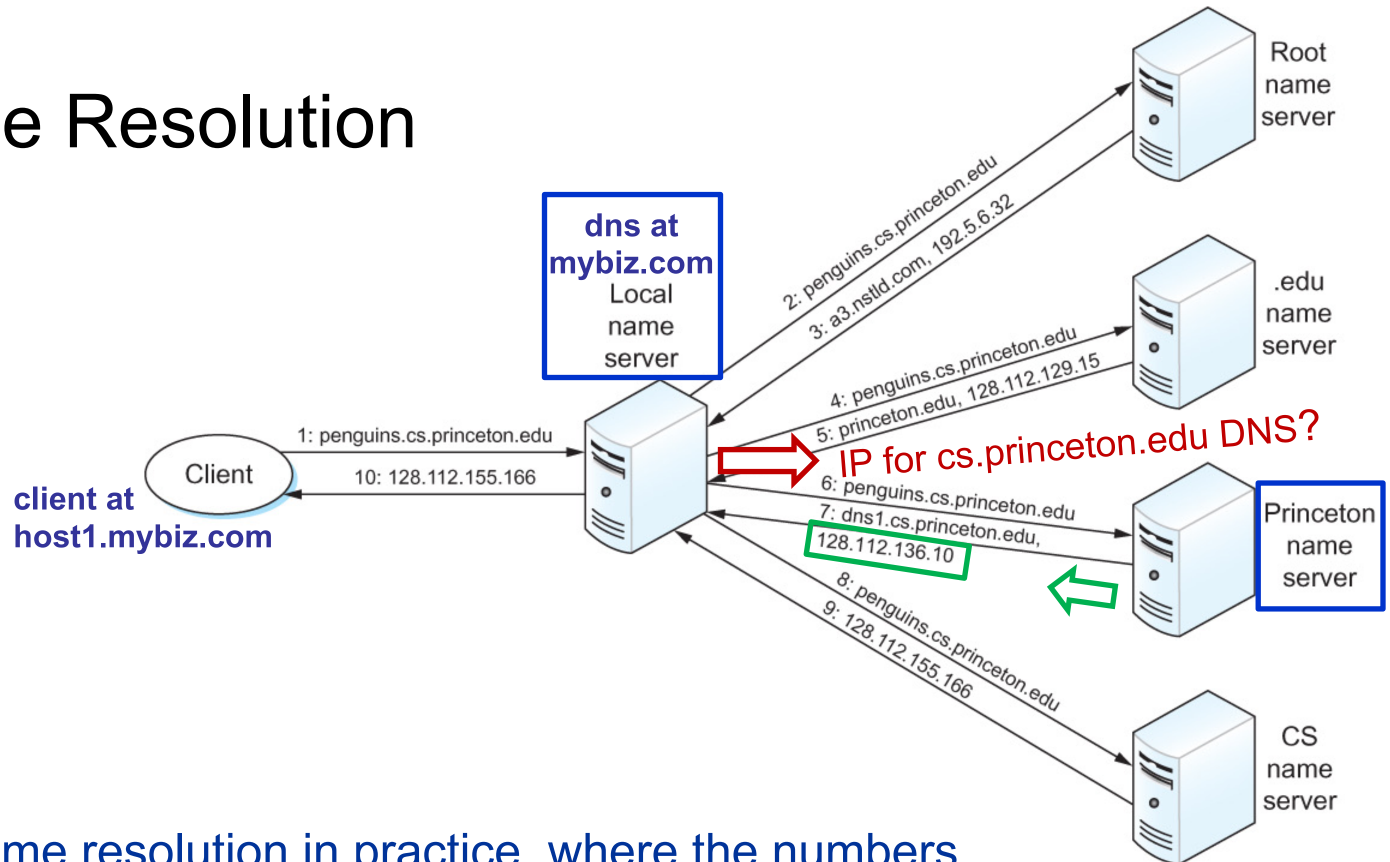
Name Resolution



Name resolution in practice, where the numbers 1–10 show the sequence of steps in the process

Infrastructure Services

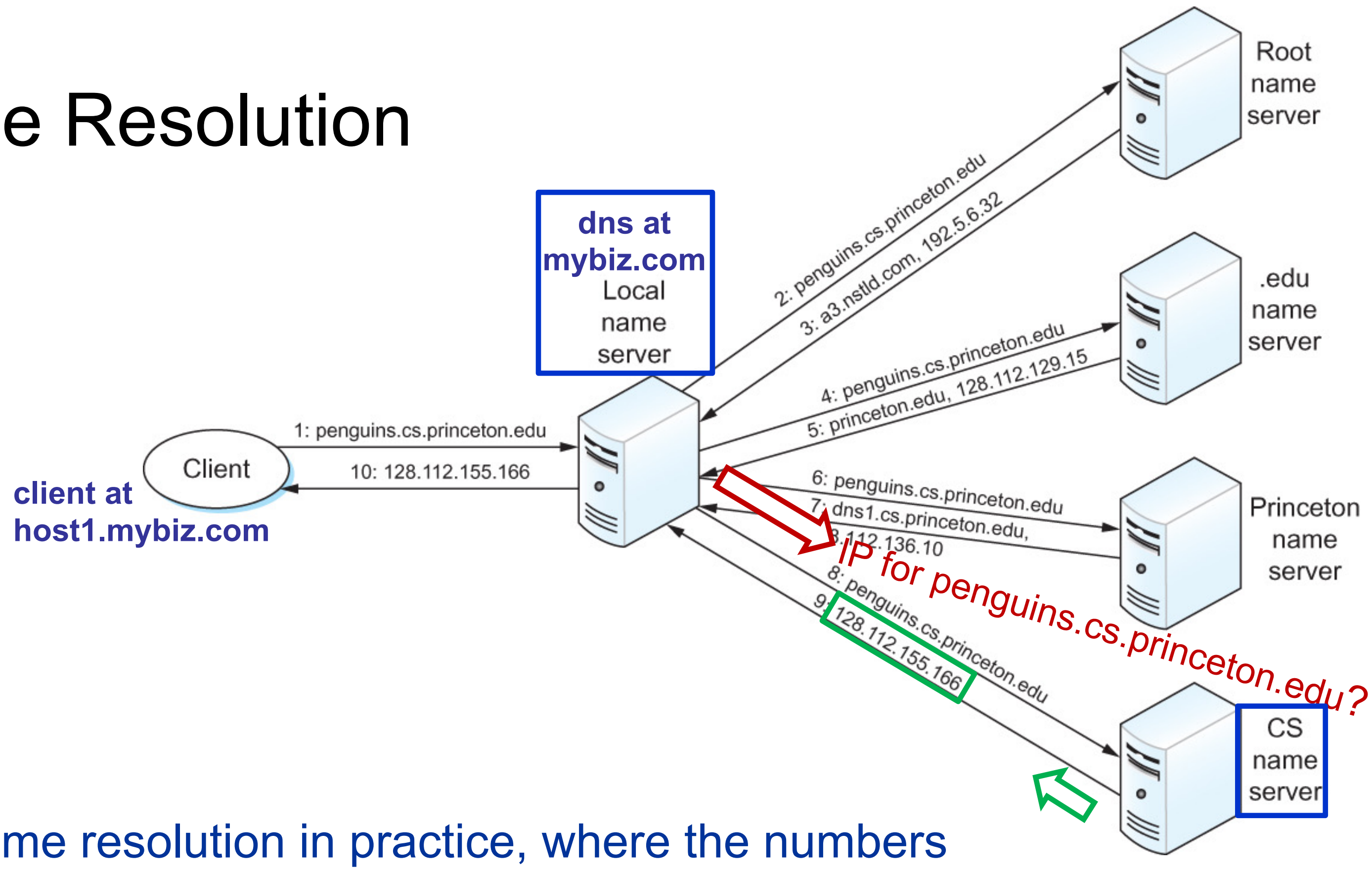
Name Resolution



Name resolution in practice, where the numbers 1–10 show the sequence of steps in the process

Infrastructure Services

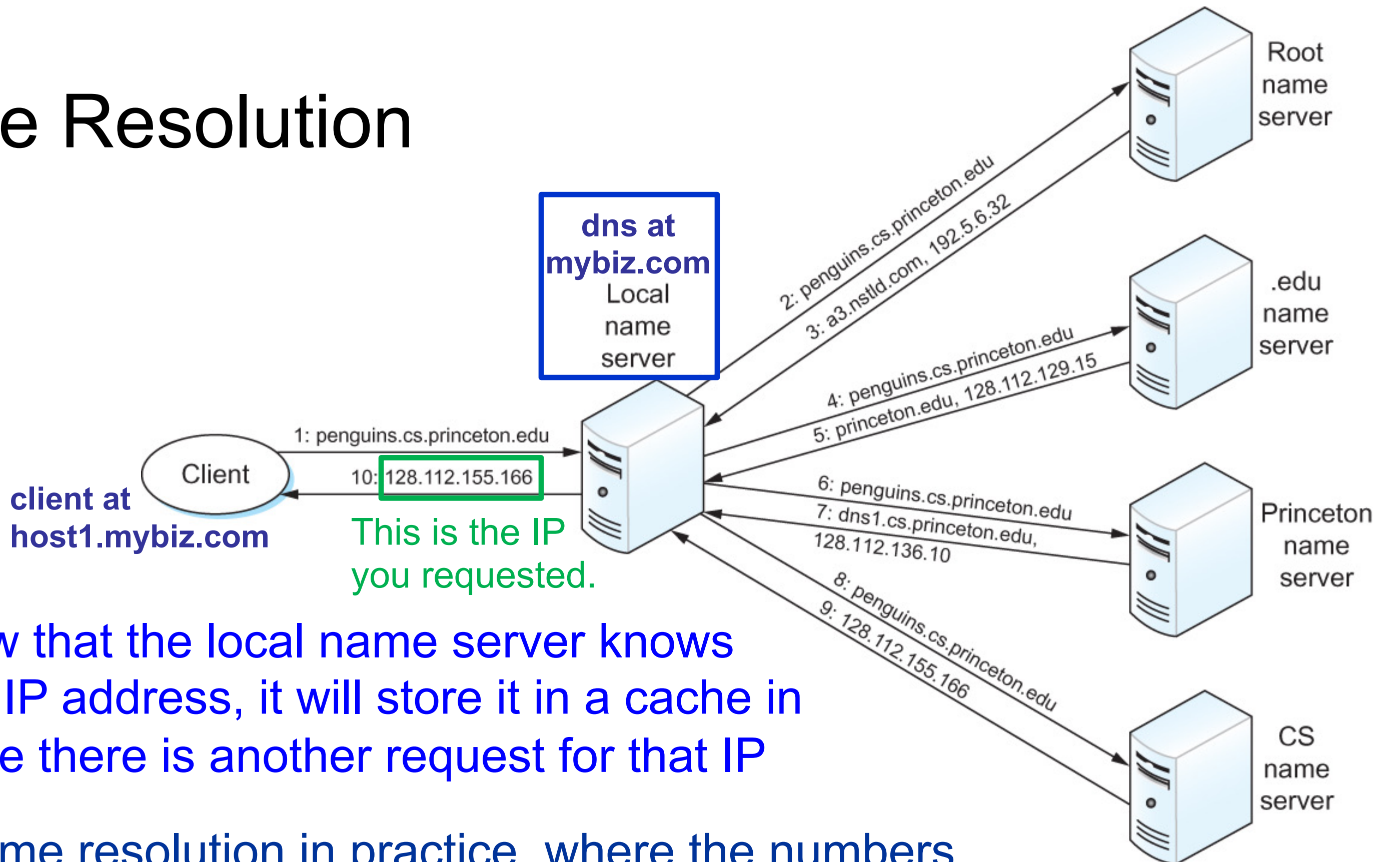
Name Resolution



Name resolution in practice, where the numbers 1–10 show the sequence of steps in the process

Infrastructure Services

Name Resolution



Now that the local name server knows the IP address, it will store it in a cache in case there is another request for that IP

Name resolution in practice, where the numbers 1–10 show the sequence of steps in the process