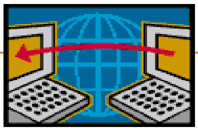


Midterm Review

EE122 Fall 2012

Scott Shenker

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson
and other colleagues at Princeton and UC Berkeley

Midterm Logistics

- Test is in this classroom starting at 5:40 exactly. Tests will be handed out before then.
- Closed book, closed notes, etc.
- Single two-sided “cheat sheet”, 8pt minimum
- No calculators, electronic devices, etc.
  - If I see them, you’ll be penalized
  - Test requires exactly one division, which you can do in your head (if not, ask us)

The test is long….(~20 pages)

- But most of the early questions are simple
  - Just to see if you’ve been listening
- And nothing is very difficult or deep
- No one will get a perfect score

Today

- Available after class
- I hate these review lectures….
- And I’m missing the A’s game.

Midterm Review

My General Philosophy on Tests

- I am not a sadist (although my kids disagree)
- I am not a masochist (except in some areas)
- For those of you who only read the slides at home:
  - If you don’t attend lectures, then it is your own damn fault if you missed something….
- I believe in testing your understanding of the basics, not tripping you up on tiny details or making you calculate pi to 15 decimal places

General Guidelines

- Know the basics well, don't focus on tiny details
  - Study lecture notes and problem sets
- Read text only for general context and to nail down certain details
  - like DNS resource records, header fields, etc.
  - Wikipedia is fine too
- Just because I didn't cover it in review doesn't mean you don't need to know it!
  - But if I covered it today, you should know it.

Things You Don't Need to Know

- The exact layout of packet headers
  - Know what the fields do, not where they are located
- Details of HTTP, CDNs, caching
  - Those are for the final
- Mathematics of M/M/1 queues

Homework #2

Scores are high except on….

- Routing validity:
  - Nodes don't *need* consistent state to be valid
  - Least cost paths are *sufficient*, but not necessary
- Reliability correctness:
  - A design where packets are resent forever is inefficient, but still reliable
- Routing: see solution sheet

One Positive Aspect of Reviews

Can focus on "putting it all together"

Putting It All Together

Headers

Packet Headers

- What does a packet on the wire look like?
- In what order to the headers occur?

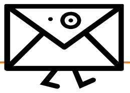
What headers are present?

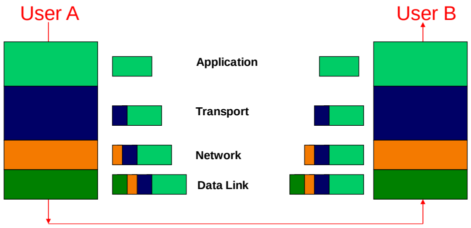
- Consider the case of a DNS request from a laptop connected to an ethernet
- Which headers are present in the packet as it hits the wires?
- Take a few minutes to discuss this…

Headers from outermost inwards

- Data-link (e.g., Ethernet, ATM, etc.)
- IP
- Transport (e.g., UDP, TCP)
- Application (e.g., DNS, DHCP, HTTP, etc.)
  - Not strictly a "header", but close enough

Layer Encapsulation





Common case: 20 bytes TCP header + 20 bytes IP header + 14 bytes Ethernet header = *54 bytes overhead*

Putting It All Together

Accessing a web page

Opening laptop, making Web request

- What steps are involved?

What messages do you need?

- Take five minutes to figure this out
- I'll take some volunteers to give their answer
- If no one volunteers, then I won't cover this….

At a high level….

- Getting an address for your laptop
- Getting the address of the server
- Contacting the server
- Fetching the data
- Shutting connection down

What protocols are used?

- Getting an address for your laptop
  - **DHCP**
- Getting the address of the server
  - **DNS**
- Contacting the server
  - **TCP**
- Fetching the data
  - **HTTP**

Working our way through answer…

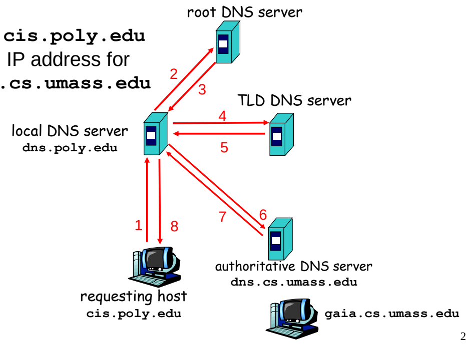
- DHCP:
  - Laptop: discovery
  - DHCP server: offer
  - Laptop: request (accepting offer)
  - DHCP server: ACK
- Which of these are broadcast?

Continuing

- DNS:
  - Laptop: request to local DNS server
  - (magic happens, discussed on next slide)
  - DNS server: response

How Does Resolution Happen?

Host at **`cis.poly.edu`**
wants IP address for
`gaia.cs.umass.edu`



DNS Resource Records

DNS: distributed DB storing resource records (RR)

RR format: `(name, value, type, ttl)`

- Type=A
  - `name` is hostname
  - `value` is IP address
- Type=NS
  - `name` is domain (e.g. foo.com)
  - `value` is hostname of authoritative name server for this domain
- Type=PTR
  - `name` is reversed IP quads
    - E.g. 78.56.34.12.in-addr.arpa
  - `value` is corresponding hostname
- Type=CNAME
  - `name` is alias name for some "canonical" name
    E.g., `www.cs.mit.edu` is really `eecsweb.mit.edu`
  - `value` is canonical name
- Type=MX
  - `value` is name of mailserver associated with `name`
  - Also includes a weight/preference

Continuing

- TCP:
  - Laptop: SYN
  - Server: SYN-ACK
  - Laptop: ACK
- HTTP: (assume single packet for each)
  - Laptop: HTTP request
  - Server: HTTP response (ACK piggybacked)
  - Laptop: TCP ACK to server resp. ***(missing in 2011MT)***
- TCP:
  - Laptop: FIN
  - Server: FIN-ACK
  - Laptop: ACK

How Did We Get to the Internet Design?

First Step: Basic Decisions

- Packet Switching winner over circuit switching
- Best-effort service model

Second Step: Architectural Principles

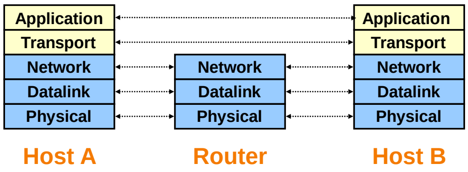
- Layering
- End-to-End Principle
- Fate-Sharing

These principles drove the design…

- How to break system into modules
  - **Dictated by Layering**
- Where modules are implemented
  - **Dictated by End-to-End Principle**
- Where state is stored
  - **Dictated by Fate-Sharing**

Who Does What?

- Five layers
  - Lower three layers implemented everywhere
  - Top two layers implemented only at hosts



What about switches?

Third Step: Design Challenges

- Consider each of the layers:
  - Physical
  - Datalink
  - Network
  - Transport
  - Application

- What function does each layer need to implement?

- And which of them are both general and hard?

Two Layers We Don’t Worry About

- Physical: Technology-dependent

- Application: Application-dependent

Datalink and Network Layers

- Both support best-effort delivery
  - Datalink over local scope: **MAC addresses**
  - Network over global scope: **IP addresses**

- Key challenge: scalable, robust **routing**
  - How to direct packets to destination

Transport Layer

- Provide reliable delivery over unreliable network

We Only Have Two Design Challenges

- **Routing**:

- **Reliable delivery**:

Routing and Reliability

- Reliable Transport:

A transport mechanism is "reliable" if and only if it resends all dropped or corrupted packets

- Routing:

Global routing state is valid if and only if there are no dead ends (easy) and there are no loops (hard)

Missing Pieces

- Sharing addresses: NAT, DHCP
- Forwarding based on addresses: LPM
- Translating names to addresses: DNS
- ….

Some General Themes

General Rules of System Design

- System not scalable?
  - Add hierarchy
  - DNS, IP addressing
- System not flexible?
  - Add layer of indirection
  - DNS names (rather than using IP addresses as names)
- System not performing well?
  - Add caches
  - Web and DNS caching

The Paradox of Internet Traffic

- The majority of flows are short
  - A few packets
- The majority of bytes are in long flows
  - MB or more
- And this trend is accelerating…

A Common Pattern…..

- Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
  - Large fraction of total metric in the top 10%
  - Sizable fraction (~10%) of total fraction in low values
- Not an exponential distribution
  - Large fraction is in top 10%
  - But low values have very little of overall total
- Lesson: have to pay attention to both ends of dist.

Little’s Law (1961)

$$L = A \times W$$

- L is average number of packets in queue
- A is average arrival rate
- W is average waiting time for each packet

- Why do you care?
  - Easy to compute L, harder to compute W

Routing

How Can You Avoid Loops?

- Restrict topology to spanning tree
  - If the topology has no loops, packets can’t loop!
- Computation over entire graph
  - Can make sure no loops
  - Link-State
- Minimizing metric in distributed computation
  - Loops are never the solution to a minimization problem
  - Distance vector

- Won’t review LS/DV, but will review learning switch

Easiest Way to Avoid Loops

- Use a topology where loops are impossible!

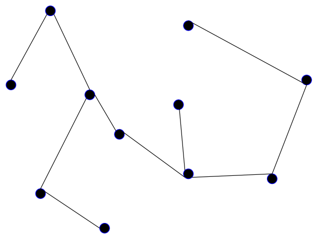
- Take arbitrary topology

- Build spanning tree (algorithm covered later)
  - Ignore all other links (as before)

- Only one path to destinations on spanning trees

- Use “learning switches” to discover these paths
  - No need to compute routes, just observe them

A Spanning Tree



Clarification

- General comments in lecture were about learning applied to case where switches were never the destination

- The examples given referred only to switches because it made the graphs simpler, but it did raise the possibility that floods didn’t reach everywhere

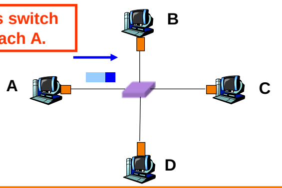
- My apologies for the confusion

Self-Learning Switch

When a packet arrives

- Inspect *source* ID, associate with *incoming* port
- Store mapping in the switch table
- Use time-to-live field to eventually forget mapping

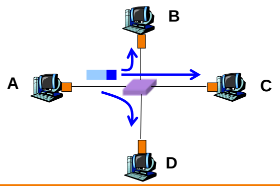
Packet tells switch how to reach A.



Self Learning: Handling Misses

When packet arrives with unfamiliar destination

- Forward packet out **all** other ports
- Response will teach switch about that destination



General Rule

When switch receives a packet:

```
index the switch table using destination ID
if entry found for destination {
    if dest on port from which packet arrived
        then drop packet
        else forward packet on port indicated
}
   else flood
```

forward on all but the interface on which the frame arrived

Core of Real Architecture

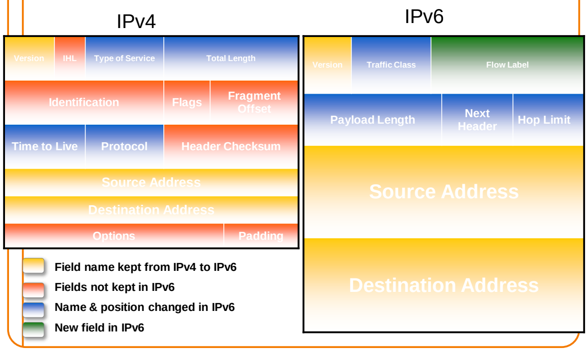
Addressing, Forwarding, TCP, DNS, Web

IP Packet Header

IP Packet Structure

<table>
<tr><td>4-bit Version</td><td>4-bit Header Length</td><td>8-bit Type of Service (TOS)</td><td colspan="2">16-bit Total Length (Bytes)</td></tr>
<tr><td colspan="3">16-bit Identification</td><td>3-bit Flags</td><td>13-bit Fragment Offset</td></tr>
<tr><td colspan="2">8-bit Time to Live (TTL)</td><td>8-bit Protocol</td><td colspan="2">16-bit Header Checksum</td></tr>
<tr><td colspan="5">32-bit Source IP Address</td></tr>
<tr><td colspan="5">32-bit Destination IP Address</td></tr>
<tr><td colspan="5">Options (if any)</td></tr>
<tr><td colspan="5">Payload</td></tr>
</table>

IPv4 and IPv6 Header Comparison



🟨 Field name kept from IPv4 to IPv6
🟥 Fields not kept in IPv6
🟦 Name & position changed in IPv6
🟩 New field in IPv6

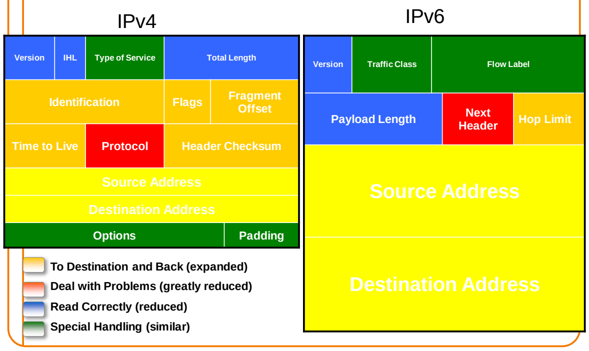
Summary of Changes

- Eliminated fragmentation ***(why?)***
- Eliminated header length ***(why?)***
- Eliminated header checksum ***(why?)***
- New options mechanism (next header) ***(why?)***
- Expanded addresses ***(why?)***
- Added Flow Label ***(why?)***

Philosophy of Changes

- Don’t deal with problems: leave to ends
  - Eliminated fragmentation
  - Eliminated checksum
  - ***Why retain TTL?***
- Simplify handling:
  - New options mechanism (uses next header approach)
  - Eliminated header length
    - ***Why couldn’t IPv4 do this?***
- Provide general flow label for packet
  - Not tied to semantics
  - Provides great flexibility

Comparison of Design Philosophy



🟨 To Destination and Back (expanded)
🟥 Deal with Problems (greatly reduced)
🟦 Read Correctly (reduced)
🟩 Special Handling (similar)

Addressing

Original Internet Addresses

- First eight bits: network address (/8)
- Last 24 bits: host address

Assumed 256 networks were more than enough!

Next Design: Classful Addressing

- Class A: if first byte in [0..127] ⇒ assume /8 **(top bit = 0)**

  0******* | ******** | ******** | ********
  - o Very large blocks (e.g., MIT has 18.0.0.0/8)
- Class B: first byte in [128..191] ⇒ assume /16 **(top bits = 10)**

  10****** | ******** | ******** | ********
  - o Large blocks (e.g,. UCB has 128.32.0.0/16)
- Class C: [192..223] ⇒ assume /24 **(top bits = 110)**

  110***** | ******** | ******** | ********
  - o Small blocks (e.g., ICIR has 192.150.187.0/24)

Classful Addressing (cont’d)

- Class D: [224..239] **(top bits 1110)**

  1110**** | ******** | ******** | ********
  - o Multicast groups
- Class E: [240..255] **(top bits 1111)**

  11110*** | ******** | ******** | ********
  - o Reserved for future use

- What problems can classful addressing lead to?
  - Only comes in 3 sizes
  - Routers can end up knowing about ***many*** class C’s (/24s)
  - Wasted address space

Today’s Addressing: CIDR

- CIDR = Classless Interdomain Routing

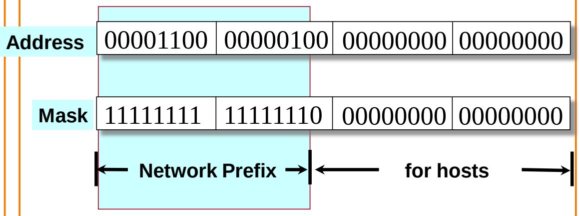
- Flexible division between network and host addresses

- ***Must specify both address and mask***
  - Clarifies where boundary between addresses lies
  - Classful addressing communicate this with first few bits
  - CIDR requires explicit mask

CIDR Addressing

Use two 32-bit numbers to represent a network.
Network number = IP address + Mask

IP Address : 12.4.0.0 IP Mask: 255.254.0.0

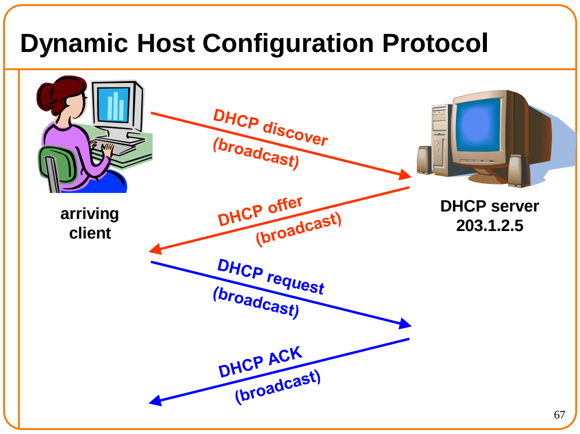


Written as 12.4.0.0/15 or 12.4/15

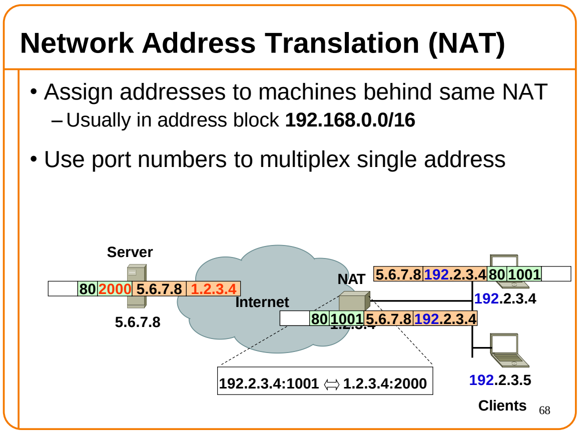
Obtaining a Block of Addresses

- Allocation is also hierarchical
  - Prefix: assigned ***to*** an institution
  - Addresses: assigned ***by*** the institution to their nodes
- Who assigns prefixes?
  - Internet Corporation for Assigned Names and Numbers
    - o Allocates large address blocks to *Regional Internet Registries*
    - o ICANN is **politically charged**
  - Regional Internet Registries (RIRs)
    - o E.g., **ARIN** (American Registry for Internet Numbers)
    - o Allocates address blocks within their regions
    - o Allocated to Internet Service Providers and large institutions ($$)
  - Internet Service Providers (ISPs)
    - o Allocate address blocks to their customers (could be recursive)
      - Often w/o charge

DHCP and NAT

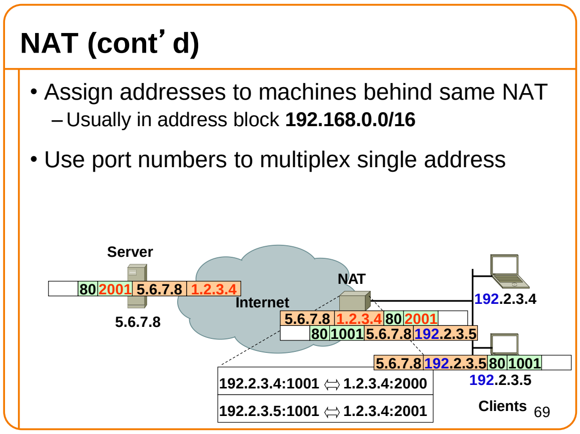


Dynamic Host Configuration Protocol



Network Address Translation (NAT)

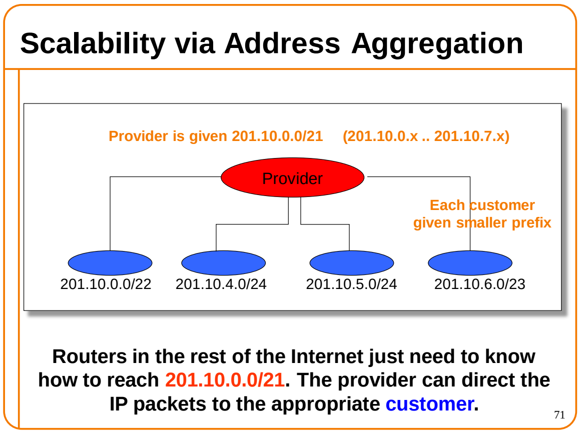
- Assign addresses to machines behind same NAT
  - Usually in address block **192.168.0.0/16**
- Use port numbers to multiplex single address



NAT (cont' d)

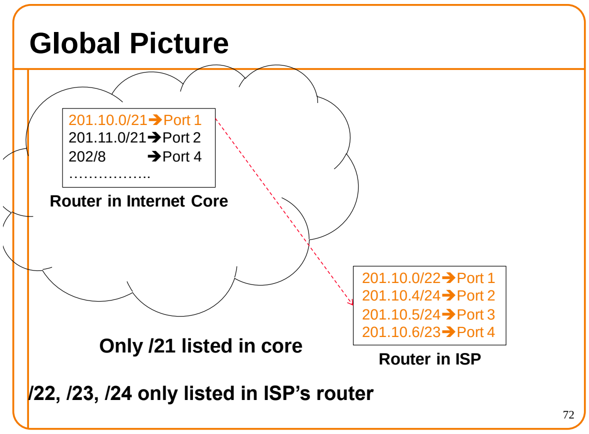
- Assign addresses to machines behind same NAT
  - Usually in address block **192.168.0.0/16**
- Use port numbers to multiplex single address

Forwarding



Scalability via Address Aggregation

Routers in the rest of the Internet just need to know how to reach 201.10.0.0/21. The provider can direct the IP packets to the appropriate customer.

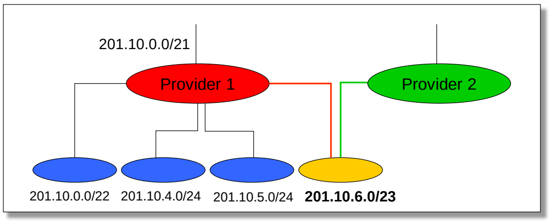


Global Picture

Only /21 listed in core

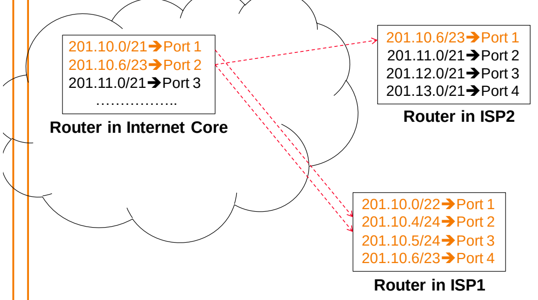
/22, /23, /24 only listed in ISP's router

Aggregation Not Always Possible



Multi-homed **customer with 201.10.6.0/23 has two providers. Other parts of the Internet need to know how to reach these destinations through** ***both*** **providers.**
⇒ /23 route must be globally visible

Multihoming Global Picture



201.10.0/21→Port 1
201.10.6/23→Port 2
201.11.0/21→Port 3
……………..

Router in Internet Core

201.10.6/23→Port 1
201.11.0/21→Port 2
201.12.0/21→Port 3
201.13.0/21→Port 4

Router in ISP2

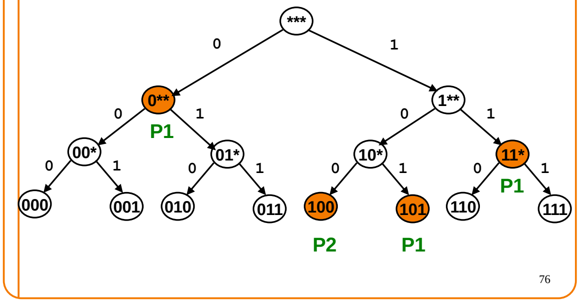
201.10.0/22→Port 1
201.10.4/24→Port 2
201.10.5/24→Port 3
201.10.6/23→Port 4

Router in ISP1

Simple Example

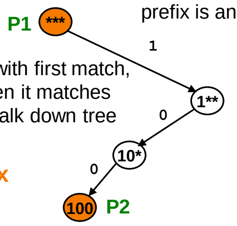
- 0**→ Port 1
- 100 → Port 2
- 101 → Port 1
- 11* → Port 1

Prefix Tree



More Compact Representation

If you ever leave path, you are done, last matched prefix is answer



Record port associated with first match, and only over-ride when it matches another prefix during walk down tree

This is longest prefix match (LPM)

Longest Prefix Match Representation

- *** → Port 1
- 100 → Port 2

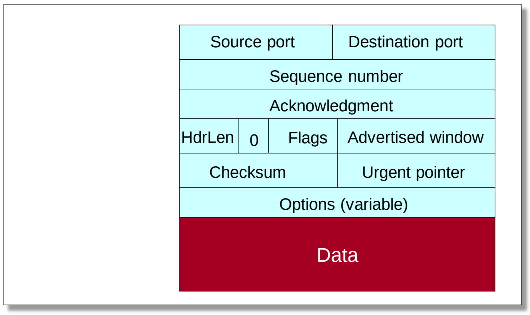
- If address matches both, then take longest match

Transport

Role of Transport Layer

- Provide common end-to-end services for app layer
  - **Deal with network on behalf of applications**
  - **Deal with applications on behalf of networks**
- Could have been built into apps, but want common implementations to make app development easier
  - Since TCP runs on end host, this is about software modularity, not overall network architecture

TCP Header



Source port | Destination port

Sequence number

Acknowledgment

HdrLen | 0 | Flags | Advertised window

Checksum | Urgent pointer

Options (variable)

Data

Example

- Packet arrives:
  - Seq: 2323
  - Ack: 4001
  - W=3000
  - [no payload]
- Appropriate response?
  - Seq: 4001, payload: 4001-8000
  - Seq: 2001, payload: 2001-5000
  - Seq: 4001, payload: 4001-5000
  - Seq: 5001, payload: 5001-6000
  - Seq: 8001, payload: 8001-9000

Advertised Window Limits Rate

- Sender can send no faster than W/RTT bytes/sec
- In ideal case, throughput = MIN [W/RTT, B]
  - Where B is bottleneck on path

Good Luck on Thursday!