The IP Data Plane: Packets and Routers

EE 122, Fall 2013
Sylvia Ratnasamy
http://inst.eecs.berkeley.edu/~ee122/

Material thanks to Ion Stoica, Scott Shenker, Jennifer Rexford, Nick McKeown, and many other colleagues
The IP layer

- So far, we’ve focused mostly on routing protocols
  - how routers discover and select end-to-end paths
  - part of a network’s control plane

- Today: the data plane
  - what data packets look like at the IP layer (the IP header)
  - how routers forward these IP packets
Recall from Lecture#3: Layer Encapsulation
What is Designing IP?

- Syntax: format of packet
  - Nontrivial part: packet “header”
  - Rest is opaque payload (why opaque?)

- Semantics: meaning of header fields
  - Required processing
Packet Headers

- Think of packet header as interface
  - Only way of passing information from packet to switch

- Designing interfaces
  - What task are you trying to perform?
  - What information do you need to accomplish it?

- Header reflects information needed for basic tasks
What Tasks Do We Need to Do?

- Read packet correctly
- Get packet to the destination; responses back to the source
- Carry data
- Tell host what to do with packet once arrived
- Specify any special network handling of the packet
- Deal with problems that arise along the path
## IP Packet Structure

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Length</th>
<th>8-bit Type of Service (TOS)</th>
<th>16-bit Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>16-bit Identification</td>
<td>3-bit Flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-bit Time to Live (TTL)</td>
<td>13-bit Fragment Offset</td>
</tr>
<tr>
<td></td>
<td>8-bit Protocol</td>
<td></td>
<td>16-bit Header Checksum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32-bit Source IP Address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>32-bit Destination IP Address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
20 Bytes of Standard Header, then Options

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Length</th>
<th>8-bit Type of Service (TOS)</th>
<th>16-bit Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16-bit Identification</th>
<th>3-bit Flags</th>
<th>13-bit Fragment Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8-bit Time to Live (TTL)</th>
<th>8-bit Protocol</th>
<th>16-bit Header Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32-bit Source IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32-bit Destination IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Options (if any)

Payload
Fields for Reading Packet Correctly

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit Version</td>
<td>4-bit</td>
<td>Version of the IP protocol.</td>
</tr>
<tr>
<td>4-bit Header Length</td>
<td>4-bit</td>
<td>Length of the header.</td>
</tr>
<tr>
<td>8-bit Type of Service (TOS)</td>
<td>8-bit</td>
<td>Type of service flags.</td>
</tr>
<tr>
<td>16-bit Total Length (Bytes)</td>
<td>16-bit</td>
<td>Total length of the packet in bytes.</td>
</tr>
<tr>
<td>16-bit Identification</td>
<td>16-bit</td>
<td>Identification of the packet.</td>
</tr>
<tr>
<td>3-bit Flags</td>
<td>3-bit</td>
<td>Flags for the packet.</td>
</tr>
<tr>
<td>13-bit Fragment Offset</td>
<td>13-bit</td>
<td>Offset of the fragment.</td>
</tr>
<tr>
<td>8-bit Time to Live (TTL)</td>
<td>8-bit</td>
<td>Time to live, the time the packet is valid.</td>
</tr>
<tr>
<td>16-bit Header Checksum</td>
<td>16-bit</td>
<td>Header checksum for validation.</td>
</tr>
<tr>
<td>32-bit Source IP Address</td>
<td>32-bit</td>
<td>Source IP address of the packet.</td>
</tr>
<tr>
<td>32-bit Destination IP Address</td>
<td>32-bit</td>
<td>Destination IP address of the packet.</td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td>Options if available.</td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td>Payload of the packet.</td>
</tr>
</tbody>
</table>
Reading Packet Correctly

- **Version number (4 bits)**
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)

- **Header length (4 bits)**
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be more when IP options are used

- **Total length (16 bits)**
  - Number of bytes in the packet
  - Maximum size is 65,535 bytes \(2^{16} - 1\)
  - … though underlying links may impose smaller limits
### Fields for Reaching Destination and Back

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4-bit</td>
<td></td>
</tr>
<tr>
<td>Header Length</td>
<td>4-bit</td>
<td></td>
</tr>
<tr>
<td>Type of Service (TOS)</td>
<td>8-bit</td>
<td></td>
</tr>
<tr>
<td>Total Length (Bytes)</td>
<td>16-bit</td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td>16-bit</td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td>3-bit</td>
<td></td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>13-bit</td>
<td></td>
</tr>
<tr>
<td>Time to Live (TTL)</td>
<td>8-bit</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td>8-bit</td>
<td></td>
</tr>
<tr>
<td>Header Checksum</td>
<td>16-bit</td>
<td></td>
</tr>
<tr>
<td>Source IP Address</td>
<td>32-bit</td>
<td></td>
</tr>
<tr>
<td>Destination IP Address</td>
<td>32-bit</td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Telling End-Host How to Handle Packet

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit Version</td>
<td></td>
</tr>
<tr>
<td>4-bit Header Length</td>
<td></td>
</tr>
<tr>
<td>8-bit Type of Service (TOS)</td>
<td></td>
</tr>
<tr>
<td>16-bit Total Length (Bytes)</td>
<td></td>
</tr>
<tr>
<td>16-bit Identification</td>
<td></td>
</tr>
<tr>
<td>3-bit Flags</td>
<td></td>
</tr>
<tr>
<td>13-bit Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>8-bit Protocol</td>
<td></td>
</tr>
<tr>
<td>8-bit Time to Live (TTL)</td>
<td></td>
</tr>
<tr>
<td>16-bit Header Checksum</td>
<td></td>
</tr>
<tr>
<td>32-bit Source IP Address</td>
<td></td>
</tr>
<tr>
<td>32-bit Destination IP Address</td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
Telling End-Host How to Handle Packet

- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host

```
L7  Application
     |    SMTP  HTTP  DNS  NTP
L4  Transport
     |    TCP   UDP
L3  Network
     |    IP
L2  Data link
     |    Ethernet FDDI  PPP
L1  Physical
     |    optical  copper  radio  PSTN
```
Telling End-Host How to Handle Packet

- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host
- Most common examples
  - E.g., “6” for the Transmission Control Protocol (TCP)
  - E.g., “17” for the User Datagram Protocol (UDP)
Potential Problems

- Header Corrupted: **Checksum**
- Loop: **TTL**
- Packet too large: **Fragmentation**
### Checksum, TTL and Fragmentation Fields

<table>
<thead>
<tr>
<th>4-bit Version</th>
<th>4-bit Header Length</th>
<th>8-bit Type of Service (TOS)</th>
<th>16-bit Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 16-bit Identification
- 3-bit Flags
- 13-bit Fragment Offset

- 8-bit Time to Live (TTL)
- 8-bit Protocol
- 16-bit Header Checksum

- 32-bit Source IP Address
- 32-bit Destination IP Address

- Options (if any)

- Payload
Header Corruption (Checksum)

- Checksum (16 bits)
  - Particular form of checksum over packet header

- If not correct, router discards packets
  - So it doesn’t act on bogus information

- Checksum recalculated at every router
  - Why?
  - Why include TTL?
  - Why only header?
Preventing Loops (TTL)

- Forwarding loops cause packets to cycle for a long time
  - As these accumulate, eventually consume all capacity

- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - …and “time exceeded” message is sent to the source
Fragmentation

- Fragmentation: when forwarding a packet, an Internet router can split it into multiple pieces ("fragments") if the packet is too big for next hop link
  - too big $\rightarrow$ exceeds the link’s “Max Transmission Unit” (MTU)

- Must reassemble to recover original packet
  - Need fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset

- Details in Section
### Fields for Special Handling

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

- “Type of Service”, or “Differentiated Services Code Point (DSCP)” (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer
  - Has been redefined several times, will cover later in class

- Options (not often used)
  - details in Section
Examples of Options

- Record Route
- Strict Source Route
- Loose Source Route
- Timestamp
- Traceroute
- Router Alert
- .....
Let’s take a quick look at the IPv6 header…
IPv6

- Motivated (prematurely) by address exhaustion
  - Addresses *four* times as big

- Steve Deering focused on simplifying IP
  - Got rid of all fields that were not absolutely necessary
  - “Spring Cleaning” for IP
Summary of Changes

- Eliminated fragmentation (why?)
- Eliminated checksum (why?)
- New options mechanism (next header) (why?)
- Eliminated header length (why?)
- Expanded addresses (why?)
- Added Flow Label (why?)
## IPv4 and IPv6 Header Comparison

<table>
<thead>
<tr>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td><strong>Version</strong></td>
</tr>
<tr>
<td><strong>IHL</strong></td>
<td><strong>Traffic Class</strong></td>
</tr>
<tr>
<td><strong>Type of Service</strong></td>
<td><strong>Flow Label</strong></td>
</tr>
<tr>
<td><strong>Total Length</strong></td>
<td><strong>Payload Length</strong></td>
</tr>
<tr>
<td><strong>Identification</strong></td>
<td><strong>Next Header</strong></td>
</tr>
<tr>
<td><strong>Flags</strong></td>
<td><strong>Hop Limit</strong></td>
</tr>
<tr>
<td><strong>Fragment Offset</strong></td>
<td><strong>Source Address</strong></td>
</tr>
<tr>
<td><strong>Time to Live</strong></td>
<td><strong>Destination Address</strong></td>
</tr>
<tr>
<td><strong>Protocol</strong></td>
<td><strong>Options</strong></td>
</tr>
<tr>
<td><strong>Header Checksum</strong></td>
<td><strong>Padding</strong></td>
</tr>
</tbody>
</table>

- **Field name kept from IPv4 to IPv6**
- **Fields not kept in IPv6**
- **Name & position changed in IPv6**
- **New field in IPv6**
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

IPv4

Version  IHL  Type of Service  Total Length
Identification  Flags  Fragment Offset
Time to Live  Protocol  Header Checksum
Source Address  
Destination Address
Options  Padding

IPv6

Version  Traffic Class  Flow Label
Payload Length  Next Header  Hop Limit
Source Address
Destination Address

Legend:
- Yellow: To Destination and Back (expanded)
- Orange: Deal with Problems (greatly reduced)
- Blue: Read Correctly (reduced)
- Green: Special Handling (similar)
IP Routers

- Core building block of the Internet infrastructure
- $120B+ industry
- Vendors: Cisco, Huawei, Juniper, Alcatel-Lucent (account for >90%)
Lecture#2: Routers Forward Packets

Forwarding Table

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCB</td>
<td>4</td>
</tr>
<tr>
<td>UW</td>
<td>5</td>
</tr>
<tr>
<td>MIT</td>
<td>2</td>
</tr>
<tr>
<td>NYU</td>
<td>3</td>
</tr>
</tbody>
</table>

Diagram showing the network with routers and switches connected to different destinations, including UCB, UW, MIT, and NYU.
Router definitions

- $N =$ number of external router “ports”
- $R =$ speed (“line rate”) of a port
- Router capacity = $N \times R$
Networks and routers

UCB

home, small business

AT&T

core

BBN

dge (enterprise)

dge (ISP)

NYU

core
Examples of routers (core)

**Juniper T4000**
- R = 10/40 Gbps
- NR = 4 Tbps

**Cisco CRS**
- R = 10/40/100 Gbps
- NR = 322 Tbps
Examples of routers (edge)

Cisco ASR 1006
- R=1/10 Gbps
- NR = 40 Gbps

Juniper M120
- R= 2.5/10 Gbps
- NR = 120 Gbps
Examples of routers (small business)

Cisco 3945E

- \( R = 10/100/1000 \text{ Mbps} \)
- \( \text{NR} < 10 \text{ Gbps} \)
What’s inside a router?

- Processes packets on their way in
- Processes packets before they leave
- Transfers packets from input to output ports

Route/Control Processor

Input and Output for the same port are on one physical linecard.
What’s inside a router?

(1) Implement IGP and BGP protocols; compute routing tables

(2) Push forwarding tables to the line cards
What’s inside a router?

Constitutes the control plane

Constitutes the data plane

Route/Control Processor

Interconnect Fabric

Linecards (input)

1

2

N

Linecards (output)

1

2

N
Input Linecards

- Tasks
  - Receive incoming packets (physical layer stuff)
  - Update the IP header

```
<table>
<thead>
<tr>
<th>Version</th>
<th>Header Length</th>
<th>Type of Service (TOS)</th>
<th>Total Length (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit Identification</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>Time to Live (TTL)</td>
<td>Protocol</td>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP Address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Input Linecards

● Tasks
  ● Receive incoming packets (physical layer stuff)
  ● Update the IP header
    ● TTL, Checksum, Options (maybe), Fragment (maybe)
  ● Lookup the output port for the destination IP address
  ● Queue the packet at the switch fabric

● Challenge: speed!
  ● 100B packets @ 40Gbps \(\rightarrow\) new packet every 20 nano secs!

● Typically implemented with specialized hardware
  ● ASICs, specialized “network processors”
  ● “exception” processing often done at control processor
Recall IP addressing and BGP routing
- For scalability, multiple IP addresses are aggregated
- BGP operates on IP address prefixes (recall “/n” notation)
- IP routing tables maintain a mapping from IP prefixes to output interfaces

Route lookup → find the longest prefix in the table that matches the packet destination address
- Longest Prefix Match (LPM) lookup
Longest Prefix Match Lookup

- Packet with destination address 12.82.100.101 is sent to interface 2, as 12.82.100.xxx is the longest prefix matching packet’s destination address.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.16.120.xxx</td>
<td>1</td>
</tr>
<tr>
<td>12.82.xxx.xxx</td>
<td>3</td>
</tr>
<tr>
<td>12.82.100.xxx</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Longest Prefix Match is **NOT**…

- Check an address against all destination prefixes and select the prefix it matches with on the most bits

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>101.xx.xxx.xxx (/3)</td>
<td>1</td>
</tr>
<tr>
<td>1**.***.xxx.xxx(/1)</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>..</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- To which port should we send a packet with destination address 100.5.6.7?
Example #1: 4 Prefixes, 4 Ports

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>201.143.0.0/22</td>
<td>Port 1</td>
</tr>
<tr>
<td>201.143.4.0.0/24</td>
<td>Port 2</td>
</tr>
<tr>
<td>201.143.5.0.0/24</td>
<td>Port 3</td>
</tr>
<tr>
<td>201.143.6.0/23</td>
<td>Port 4</td>
</tr>
</tbody>
</table>
Finding the Match

- Consider 11001001100011110000010111010010
  - First 21 bits match 4 partial prefixes
  - First 22 bits match 3 partial prefixes
  - First 23 bits match 2 partial prefixes
  - First 24 bits match exactly one full prefix
Finding Match Efficiently

- Testing each entry to find a match scales poorly
  - On average: $O(\text{number of entries})$

- Leverage tree structure of binary strings
  - Set up tree-like data structure

- Return to example:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100100110001111000000000100**</td>
<td>1</td>
</tr>
<tr>
<td>1100100110001111000000100100**</td>
<td>2</td>
</tr>
<tr>
<td>11001001100011110000001010101**</td>
<td>3</td>
</tr>
<tr>
<td>1100100110001111000000111011111111000000011**</td>
<td>4</td>
</tr>
</tbody>
</table>
Consider four three-bit prefixes

- Just focusing on the bits where all the action is....

- 0** ➔ Port 1
- 100 ➔ Port 2
- 101 ➔ Port 3
- 11* ➔ Port 4
Tree Structure

0** ➔ Port 1
100 ➔ Port 2
101 ➔ Port 3
11* ➔ Port 4
Walk Tree: Stop at Prefix Entries

0** ➔ Port 1
100 ➔ Port 2
101 ➔ Port 3
11* ➔ Port 4
Walk Tree: Stop at Prefix Entries

0** → Port 1
100 → Port 2
101 → Port 3
11* → Port 4
Slightly Different Example

- Several of the unique prefixes go to same port
  - 0** → Port 1
  - 100 → Port 2
  - 101 → Port 1
  - 11* → Port 1
Prefix Tree

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

00* → P1
01* → P1
10* → Port 1
11* → Port 1

000 → 0
001 → 1
010 → 0
011 → 1
100 → 0
101 → 1
110 → 0
111 → 1
More Compact Representation

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

Record port associated with latest match, and only over-ride when it matches another prefix during walk down tree
LPM in real routers

- Real routers use far more advanced/complex solutions than the approaches I just described
  - but what we discussed is their starting point

- With many heuristics and optimizations that leverage real-world patterns
  - Some destinations more popular than others
  - Some ports lead to more destinations
  - Typical prefix granularities
Recap: Input linecards

- Main challenge is processing speeds

- Tasks involved:
  - Update packet header (easy)
  - LPM lookup on destination address (harder)

- Mostly implemented with specialized hardware
Output Linecard

- **Packet classification**: map each packet to a “flow”
  - Flow (for now): set of packets between two particular endpoints
- **Buffer management**: decide when and which packet to drop
- **Scheduler**: decide when and which packet to transmit
Output Linecard

- **Packet classification**: map each packet to a “flow”
  - Flow (for now): set of packets between two particular endpoints
- **Buffer management**: decide when and which packet to drop
- **Scheduler**: decide when and which packet to transmit

- Used to implement various forms of policy
  - Deny all e-mail traffic from ISP-X to Y (access control)
  - Route IP telephony traffic from X to Y via PHY_CIRCUIT (policy)
  - Ensure that no more than 50 Mbps are injected from ISP-X (QoS)
Simplest: FIFO Router

- No classification
- Drop-tail buffer management: when buffer is full drop the incoming packet
- First-In-First-Out (FIFO) Scheduling: schedule packets in the same order they arrive
Packet Classification

- Classify an IP packet based on a number of fields in the packet header, e.g.,
  - source/destination IP address (32 bits)
  - source/destination TCP port number (16 bits)
  - Type of service (TOS) byte (8 bits)
  - Type of protocol (8 bits)

- In general fields are specified by range
  - classification requires a multi-dimensional range search!
Scheduler

- One queue per “flow”
- Scheduler decides when and from which queue to send a packet
- Goals of a scheduling algorithm:
  - Fast!
  - Depends on the policy being implemented (fairness, priority, etc.)
Example: Priority Scheduler

- Priority scheduler: packets in the highest priority queue are always served **before** the packets in lower priority queues.
Example: Round Robin Scheduler

- Round robin: packets are served from each queue in turn
Switching

![Diagram of switching system with interconnect fabric, route/control processor, and linecards.]
Shared Memory (1st Generation)

Limited by rate of shared memory

(* Slide by Nick McKeown, Stanford Univ.)
Shared Bus (2nd Generation)

Limited by shared bus

(* Slide by Nick McKeown)
Point-to-Point Switch (3rd Generation)

Switched Backplane

(*Slide by Nick McKeown)
This is called an “output queued” switch
This is called an “input queued” switch
Two challenges with input queuing

1) Need an internal fabric scheduler!
3rd Gen. Router: Switched Interconnects

Fabric Scheduler
Two challenges with input queuing

1) Need an internal fabric scheduler!
2) Must avoid “head-of-line” blocking
Head of Line Blocking

HoL blocking limits throughput to approximately 58% of capacity
“Virtual Output Queues”
3rd Gen. Router: Switched Interconnects
Reality is more complicated

- Commercial (high-speed) routers use
  - combination of input and output queuing
  - complex multi-stage switching topologies (Clos, Benes)
  - distributed, multi-stage schedulers (for scalability)

- We’ll consider one simpler context
  - de-facto architecture for a long time and still used in lower-speed routers
Scheduling

- **Context**
  - crossbar fabric
  - centralized scheduler

- **Goals**
  - *Work conserving (100% throughput)*
  - Fast
Scheduling

- **Context**
  - crossbar fabric
  - centralized scheduler

- **Goal:** 100% throughput, fast
  - optimal solution: maximum matching on a bipartite graph
  - problem: too slow
  - **practical solution:** a good *maximal* matching
  - multiple fast algorithms exist for computing a good and fair maximal matching [PIM, iSlip]
In Summary

- IP header formats
  - not as boring as one might imagine

- IP routers
  - core building block of the infrastructure
  - needs simple, fast implementations for longest-prefix matching, multi-dimensional search, switch and output scheduling
Administrivia

- All but one of you should now have an instructional account
  - email apanda@cs and sylvia@cs if you haven’t!
  - don’t wait until 5 minutes before HW1 is due

- HW1 is due Oct 4, 5pm
- Check the course website for other adjustments

- Next lecture will start with a 5minute Q&A with Gautam/Kaifei/Radhika on Project#1