CS4450

Computer Networks:
Architecture and Protocols

Lecture 16
BGP—unsolved problems
THE Internet Protocol
Switch Architecture

Rachit Agarwal
Announcements

• Problem set 4 released

• You may not realize this but ....
  • We have learnt a lot of material!!!!!!

• Next lecture is very very very ....
  • very very very very ....
  • important
  • Please attend

• I will discuss how everything we have covered so far FITS TOGETHER ...
  • ... into an end-to-end design
Goals for Today’s Lecture

• Wrap up BGP

• Understand IP (the Internet Protocol)
  • Packet Header as a network “interface”

• Understand switch architecture
ASes provide “transit” between their customers

Peers do not provide transit between other peers
Recap: BGP Inspired by Distance Vector

- Per-destination route advertisements
- No global sharing of network topology
- Iterative and distributed convergence on paths

But, **four key differences**

- **BGP does not pick shortest paths**
- **Path-vector rather than distance vector**
  - Each announcement contains the path for each destination
- **Selective route advertisement**
  - If I select a path, I don’t *have to* advertise it to others
- **Route aggregation**
  - Rather than storing a.b.*.*/16 and a.c.*.*/16, store a.*.*.*/*8
Recap: Policy on how routes are selected and exported

**Selection:** Which path to use
- Controls whether / how traffic leaves the network
- Typical policy:
  1. Make or save money (send to customer > peer > provider)
  2. Maximize performance (smallest AS path length)
  3. Minimize use of my network bandwidth (“hot potato”)
  4. ...

**Export:** Which path to advertise
- Controls whether / how traffic enters the network

<table>
<thead>
<tr>
<th>Destination prefix advertised by...</th>
<th>Export route to...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Everyone (providers, peers, other customers)</td>
</tr>
<tr>
<td>Peer</td>
<td>Customers</td>
</tr>
<tr>
<td>Provider</td>
<td>Customers</td>
</tr>
</tbody>
</table>
Recap: Putting the Pieces Together

1. Provide internal reachability (IGP)
2. Learn routes to external destinations (eBGP)
3. Distribute externally learned routes internally (iBGP)
4. Travel shortest path to egress (IGP)
Recap: Route Updates

- Format: `<IP prefix: route attributes>`
- Two kinds of updates:
  - Announcements: new routes or changes to existing routes
  - Withdrawals: remove routes that no longer exist
- Route Attributes
  - Describe routes, used in `selection/export` decisions
    - ASPATH
    - Local PREF
    - MED
    - IGP Cost
Questions?
BGP Details

- BGP Policy
  - Typical policies and implementation

- BGP protocol details

- Why am I not blindly in love with BGP?

- Why do I love BGP?
Why am I not blindly in love with BGP?

- Reachability
- Convergence
- Performance
- Anomalies
- Security
Reachability

- In shortest-path routing, if graph is connected then reachability is assured
- With policy routing, this doesn’t always hold
Convergence

- If all AS policies follow Gao-Rexford rules,
  - Then BGP is guaranteed to converge (safety)

- For arbitrary policies, BGP may fail to converge!

- Do you appreciate the beauty of the result (and Gao-Rexford policies?)
“1” prefers “1 3 0” over “1 0” to reach “0”
Initially: nodes 1, 2, 3 know only shortest path to 0
Step-by-step Policy Oscillation

1 advertises its path 1 0 to 2
Step-by-step Policy Oscillation
3 advertises its path 3 0 to 1
Step-by-step Policy Oscillation
Step-by-step Policy Oscillation

1 withdraws its path 1 0 from 2
Step-by-step Policy Oscillation
Step-by-step Policy Oscillation

2 advertises its path 2 0 to 3
Step-by-step Policy Oscillation
Step-by-step Policy Oscillation

3 withdraws its path 3 0 from 1
Step-by-step Policy Oscillation

![Diagram showing a network with nodes labeled 0, 1, 2, and 3, and connections between them.]
1 advertises its path 1 0 to 2
Step-by-step Policy Oscillation
Step-by-step Policy Oscillation

2 withdraws its path 2 0 from 3
Step-by-step Policy Oscillation

We are back to where we started!
The stable path assignment implicitly defines a tree rooted at the origin. Note, however, that this is not always a spanning tree. For example, Fig. 2(a) presents a stable paths for a node that is not shortest path. The simplest instance, called a shortest path tree as depicted in Fig. 3(b). An alternative solution for node 3, yet it has the same unique solution as Fig. 2.

So far, our examples each has had at most one solution. This is not always the case. The simplest instance, called a shortest path tree as depicted in Fig. 3(b). An alternative solution for node 3, yet it has the same unique solution as Fig. 2.

Therefore, any stable path adds one permitted path and its two solutions. However, as is explained in Section IV, the protocol SPVP can diverge and its two solutions.

If no such assignment exists, then the solutions are shortest paths. For example, Fig. 2(a) presents a shortest path tree as depicted in Fig. 3(b). An alternative solution for node 3, yet it has the same unique solution as Fig. 2.

Therefore, any stable path adds one permitted path and its two solutions. However, as is explained in Section IV, the protocol SPVP can diverge and its two solutions.

If no such assignment exists, then the solutions are shortest paths. For example, Fig. 2(a) presents a shortest path tree as depicted in Fig. 3(b). An alternative solution for node 3, yet it has the same unique solution as Fig. 2.
BGP Example (Bad bad bad bad)

NAUGHTY GADGET

The ranking function for each nonzero node is depicted as a node in the vertical list next to the node, with the highest ranked path at the top going down to the lowest ranked nonempty path at the bottom. The ranking of paths is not required to prefer shorter paths however, that this is not always a spanning tree.

A stable path assignment is defined as an assignment implicitly defines a tree rooted at the origin. Note, therefore, any stable path assignment having more than one solution is illustrated in Fig. 3(a). The solution for this problem.
Performance Non-Issues

- Internal Routing
  - Domains typically use "hot potato" routing
  - Not always optimal, but economically expedient

- Policy not about performance
  - So policy-chosen paths aren’t shortest

- AS path length can be misleading
  - 20% of paths inflated by at least 5 router hops
Performance (example)

- AS path length can be misleading
  - An AS may have many router-level hops

BGP says that path 4 1 is better than path 3 2 1
Performance: Real Issue

Slow Convergence

- BGP outages are biggest source of Internet problems

- Labovitz et al. *SIGCOMM’97*
  - 10% of routes available less than 95% of the time
  - Less than 35% of routes available 99.99% of the time

- Labovitz et al. *SIGCOMM 2000*
  - 40% of path outages take 30+ minutes to repair

- But most popular paths are very stable
Security

- An AS can claim to serve a prefix that they actually don’t have a route to (blackholing traffic)
  - Problem not specific to policy or path vector
  - Important because of AS autonomy
  - **Fixable:** make ASes prove they have a path

- But...
  - AS may forward packets along a route different from what is advertised
    - Tell customers about a fictitious short path...
    - Much harder to fix!
BGP Outline

- BGP Policy
  - Typical policies and implementation

- BGP protocol details

- Why am I not blindly in love with BGP?

- Why do I love BGP?
Think about what BGP is able to achieve....

• Every ISP acts completely independently and selfishly....
  • Policies, path selection, path advertisement, complete privacy
  • Creates relationships independently
  • Cares only about its own interest—money earned
  • Operates in a distributed manner
  • And yet ....

• How frequently are we disconnected from the Internet?
  • Almost rare ....
  • It just works .... And ...
BGP is one of those examples where ....

• Real-world protocol that was born in an organic manner ...
  • ... has led to advances in computing
  • A recent result:
    • BGP is an example of the “hardest” Nash equilibrium instance
    • Many recent beautiful results in theoretical computer science
  • But ...

• My real love?
  • It just works .... And nobody knows why!!

• Has been working since 1991(5)
  • With almost no changes ...
  • Perfect in inception, but perhaps poor in execution
What do we know so far [1] ...

- Network performance metrics
  - Transmission delay, propagation delay, queueing delay, bandwidth

- Sharing networks
  - Circuit switching, packet switching, and associated tradeoffs
  - Why is Internet packet switched?

- Architectural principles and design goals
  - Layering principle, End-to-end principle, Fate sharing principle
  - Many important design goals from David Clark’s paper
    - And many important missing goals

- Addressing
  - Link layer MAC names, and scalability challenges at the Internet
  - Network layer IP addresses: three requirements, aggregation, CIDR
What do we know so far [2] ...

• Link Layer
  • Sharing a Broadcast medium, associated challenges, CSMA/CD
  • Link layer addressing: MAC names
  • Why Frames? Why Switched Ethernet?
  • The Spanning Tree Protocol (STP)

• Network Layer
  • Why Network Layer? Why not just use STP across the Internet?
  • Routing Tables: A collection of spanning trees, one per destination
  • Generating Valid Routing tables (within a domain):
    • Global view (Link-State Protocol), and limitations
    • Local view (Distance-vector Protocol)
  • Generating Valid Routing tables (across domains):
    • Border Gateway Protocol, Internet structure, routing policies
One last piece—Packet headers
Network Layer

• THE functionality: delivering the data

• THE protocol: Internet Protocol (IP)

• Achieves its functionality (delivering the data), using three ideas:
  • Addressing (IP addressing)
  • Routing (using a variety of protocols)
  • Packet header as an interface (Encapsulating data into packets)
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 Header as Interface

• 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 the packet to the destination
• Get responses to the packet back to source
• Carry data
• Tell host what to do with the packet once arrived
• Specify any special network handling of the packet
• Deal with problems that arise along the path
Reading Packet Correctly

• Where does the header end?
• Where the the packet end?
• What protocol are we using?
  • Why is this so important?
Getting to the Destination

• Provide destination address

• Should this be location or identifier (name)?
  • And what’s the difference?

• If a host moves should its address change?
  • If not, how can you build scalable Internet?
  • If so, then what good is an address for identification?
Getting Response Back to Source

• Source address

• Necessary for routers to respond to source
  • When would they need to respond back?
    • Failures!
  • Do they really need to respond back?
    • How would the source know if the packet has reached the destination?
Carry Data

• Payload!
Questions?
List of Tasks

• Read packet correctly
• Get the packet to the destination
• Get responses to the packet back to 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
Telling Destination How to Process Packet

• Indicate which protocols should handle packet
• What layers should this protocol be in?
• What are some options for this today?
• How does the source know what to enter here?
Special Handling

• Type of service, priority, etc.

• Options: discuss later
Dealing With Problems

• Is packet caught in loop?
  • TTL

• Header corrupted:
  • Detect with Checksum
  • What about payload checksum?

• Packet too large?
  • Deal with fragmentation
  • Split packet apart
  • Keep track of how to put together
Are We Missing Anything?

• Read packet correctly
• Get the packet to the destination
• Get responses to the packet back to 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
From Semantics to Syntax

• The past few slides discussed the information the header must provide
• Will now show the syntax (layout) of IPv4 header, and discuss the semantics in more detail
# 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></td>
<td></td>
</tr>
<tr>
<td>16-bit Identification</td>
<td>3-bit Flags</td>
<td>13-bit Fragment Offset</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>
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>

<table>
<thead>
<tr>
<th>Options (if any)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Next Set of Slides

• Mapping between tasks and header fields
• Each of these fields is devoted to a task
• Let’s find out which ones and why...
Go Through Tasks One-by-One

• Read packet correctly
• Get the packet to the destination
• Get responses to the packet back to 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
Read 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 Reading Packet Correctly

- 4-bit Version
- 4-bit Header Length
- 8-bit Type of Service (TOS)
- 16-bit Total Length (Bytes)
- 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
Getting Packet to Destination and Back

• Two IP addresses
  • Source IP address (32 bits)
  • Destination IP address (32 bits)

• Destination Address
  • Unique locator for the receiving host
  • Allows each node to make forwarding decisions

• Source Address
  • Unique locator for the sending host
  • Recipient can decide whether to accept packet
  • Enables recipient to send a reply back to the source
# Fields for Reading Packet Correctly

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit Version</td>
<td>4-bits</td>
</tr>
<tr>
<td>4-bit Header Length</td>
<td>4-bits</td>
</tr>
<tr>
<td>8-bit Type of Service (TOS)</td>
<td>1-byte</td>
</tr>
<tr>
<td>16-bit Total Length (Bytes)</td>
<td>2-bytes</td>
</tr>
<tr>
<td>16-bit Identification</td>
<td>2-bytes</td>
</tr>
<tr>
<td>3-bit Flags</td>
<td>1-byte</td>
</tr>
<tr>
<td>13-bit Fragment Offset</td>
<td>1-byte</td>
</tr>
<tr>
<td>8-bit Time to Live (TTL)</td>
<td>1-byte</td>
</tr>
<tr>
<td>8-bit Protocol</td>
<td>1-byte</td>
</tr>
<tr>
<td>16-bit Header Checksum</td>
<td>2-bytes</td>
</tr>
<tr>
<td>32-bit Source IP Address</td>
<td>4-bytes</td>
</tr>
<tr>
<td>32-bit Destination IP Address</td>
<td>4-bytes</td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
Questions?
List of Tasks

• Read packet correctly
• Get the packet to the destination
• Get responses to the packet back to 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
**Telling 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

<table>
<thead>
<tr>
<th>Protocol = 6</th>
<th>Protocol = 17</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Header</td>
<td>IP Header</td>
</tr>
<tr>
<td>TCP Header</td>
<td>TCP Header</td>
</tr>
</tbody>
</table>
Fields for Reading Packet Correctly

- 4-bit Version
- 4-bit Header Length
- 8-bit Type of Service (TOS)
- 16-bit Total Length (Bytes)
- 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
Special Handling

- **Type-of-Service (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, no general use

- **Options**
  - Ability to specify other functionality
  - Extensible format
Examples of Options

• Record Route
• Strict Source Route
• Loose Source Route
• Timestamp
• Traceroute
• Router Alert
• ...

Potential Problems

• Header Corrupted: **Checksum**

• Loop: **TTL**

• Packet too large: **Fragmentation**
Preventing Loops

• Forwarding loops cause packets to cycle forever
  • 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
    • Using “ICMP” control message; basis for traceroute
<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit Version</td>
<td>4-bit</td>
</tr>
<tr>
<td>4-bit Header Length</td>
<td>4-bit</td>
</tr>
<tr>
<td>8-bit Type of Service (TOS)</td>
<td>8-bit</td>
</tr>
<tr>
<td>16-bit Total Length (Bytes)</td>
<td>16-bit</td>
</tr>
<tr>
<td>16-bit Identification</td>
<td>16-bit</td>
</tr>
<tr>
<td>3-bit Flags</td>
<td>3-bit</td>
</tr>
<tr>
<td>13-bit Fragment Offset</td>
<td>13-bit</td>
</tr>
<tr>
<td>8-bit Time to Live (TTL)</td>
<td>8-bit</td>
</tr>
<tr>
<td>8-bit Protocol</td>
<td>8-bit</td>
</tr>
<tr>
<td>16-bit Header Checksum</td>
<td>16-bit</td>
</tr>
<tr>
<td>32-bit Source IP Address</td>
<td>32-bit</td>
</tr>
<tr>
<td>32-bit Destination IP Address</td>
<td>32-bit</td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>
Header Corruption

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

• If not correct, router discards packets
  • So it doesn’t act in bogus information

• Checksum recalculated at every router
  • Why?
  • Why include TTL?
  • Why only header?
Checksum Field

- 4-bit Version
- 4-bit Header Length
- 8-bit Type of Service (TOS)
- 16-bit Total Length (Bytes)
- 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
Packet Header as an interface

• Useless to learn the header format by heart
  • If you remember the tasks that need to be performed ...
  • Understanding *why* header format is what it is ...
  • In general: if you understand the problem, solution is easy
  • As the problem evolves, you will know where to look for a solution

• Transition from IPv4 to IPv6
  • Gradually happening ...
  • If you want to learn a bit, see backup slides
Switch/Router Architecture
IP Routers and Switches (used interchangeably today)

• Core building block of Internet infrastructure

• $120B+ industry

• Vendors: Cisco, Huawei, Juniper, Alcatel-Lucent (account for >90%)
Recap: Routers Forward Packets
Router Definitions

- \( N \) = No. of external router ports
- \( R \) = bandwidth ("line rate") of a port
- Router capacity = \( N \times R \)
Networks and Routers

- Cornell
  - home, small business
- AT&T
  - core
- edge/border (ISP)
- edge/border (enterprise)
- BBN
- MIT
Examples of Routers (core)

- **Core**: Cisco CRS
  - $R = 10/40/100$ Gbps
  - NR = 922 Tbps
  - Netflix: 0.7 GB/hr (1.5 Mb/s)
  - ~600 million concurrent Netflix users

- **Edge (ISP)**: Cisco ASR
  - $R = 1/10/40$ Gbps
  - NR = 120 Gbps

- **Edge (enterprise)**: Cisco 3945E
  - $R = 10/100/1000$ Mbps
  - NR < 10 Gbps
What’s Inside a Router?

- Processes packets on their way in
- Transfers packets from input to output ports
- Input and Output for the same port are on one physical linecard
- Processes packets before they leave

Linecards (input)

1

2

N

Route/Control Processor

Interconnect (Switching) Fabric

Linecards (output)

1

2

N

Processes packets on their way in

Transfers packets from input to output ports

Input and Output for the same port are on one physical linecard

Processes packets before they leave
What’s Inside a Router?

1. Implement IGP and BGP protocols; compute routing tables
2. Push forwarding tables to the line cards

Linecards (input) → Interconnect (Switching) Fabric → Linecards (output)
What’s Inside a Router?

- **Route/Control Processor**
  - Constitutes the control plane

- **Interconnect Fabric**
  - Constitutes the data plane

- **Linecards (input)**
  - 1
  - 2
  - N

- **Linecards (output)**
  - 1
  - 2
  - N
Input Line Cards: Tasks

- Receive incoming packets (physical layer stuff)
- Update the IP header
  - TTL, Checksum (maybe some other fields)
- Lookup the output port for the destination IP address
- Queue the packet at the switch fabric
Challenge: Speed!

- 100B packets @ 40Gbps => packet every 20 nano secs!

- Typically implemented with specialized hardware
  - ASICs, specialized “network processors”
Looking up the Output Port

- Upon receiving a packet
  - Inspect the destination IP address in the header
  - Index into the routing/forwarding table
  - If no match, select the **default route**
  - Forward packet out appropriate interface

- Default route
  - Configured to cover cases where no matches
  - Allows small tables at edge (w/o routing algorithms)
    - if it isn’t on my subnet, send it to my ISP
Scaling the Lookup

- Recall: For scalability, addresses are **aggregated**

- Longest Prefix match
  - Find the entry with matching “longest prefix” with destination address

<table>
<thead>
<tr>
<th>Address</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.16.120.xxx</td>
<td>1</td>
</tr>
<tr>
<td>128.82.xxx.xxx</td>
<td>3</td>
</tr>
<tr>
<td>128.82.100.xxx</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Finding a Match

- Incoming packet destination: 201.143.7.0

<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 a Match: Covert to Binary

- Incoming packet destination: 201.143.7.0

| 11001001 | 10001111 | 00000111 | 11010010 |

Routing Table

201.143.0.0/22

| 11001001 | 10001111 | 000000 - - | - - - - - - |

201.143.4.0/24

| 11001001 | 10001111 | 00000100 | - - - - - - |

201.143.5.0/24

| 11001001 | 10001111 | 00000101 | - - - - - - |

201.143.6.0/23

| 11001001 | 10001111 | 0000011- | - - - - - - |
### Finding a Match: Covert to Binary

- **Incoming packet destination:** 201.143.7.0

<table>
<thead>
<tr>
<th>Destination</th>
<th>Prefix Length</th>
<th>Binary Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>201.143.0.0/22</td>
<td>6</td>
<td>11001001 10001111 000000111 11010010</td>
</tr>
<tr>
<td>201.143.4.0/24</td>
<td>8</td>
<td>11001001 10001111 000000100 - - - - - -</td>
</tr>
<tr>
<td>201.143.5.0/24</td>
<td>8</td>
<td>11001001 10001111 000000101 - - - - - -</td>
</tr>
<tr>
<td>201.143.6.0/23</td>
<td>8</td>
<td>11001001 10001111 00000011- - - - - -</td>
</tr>
</tbody>
</table>
Finding a Match: Covert to Binary

- Incoming packet destination: 201.143.7.0

| 11001001 | 10001111 | 000000111 | 11010010 |

**Routing Table**

<table>
<thead>
<tr>
<th>Network</th>
<th>Prefix Length</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>201.143.0.0/22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11001001 10001111 000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201.143.4.0/24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11001001 10001111 00000100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201.143.5.0/24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11001001 10001111 00000101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>201.143.6.0/23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11001001 10001111 00000011-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Longest Prefix Match

- Incoming packet destination: 201.143.7.0

Routing Table

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Destination</th>
<th>Prefix Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>201.143.0.0/22</td>
<td>201.143.4.0/24</td>
<td>201.143.5.0/24</td>
</tr>
<tr>
<td>201.143.6.0/23</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- Testing each entry to find a match scales poorly
  - Roughly (number of entries) × (number of bits)

- Must leverage tree structure of binary strings
  - Set up tree-like data structure
  - Called a **TRIE**
    - We will briefly discuss it; more details in text
    - In case you are interested ....
Consider Four 3-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
Walk Tree: Stop at Prefix Entries

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

walking trees takes $O(#\text{bits})$
Longest Prefix Match in Real Routers

• Real routers use far more advanced/complex solutions
  • But what we discussed is the 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 fix granularities
Recap: Input Linecards

• Main challenge is processing speed
  • But what we discussed is the starting point

• Tasks involved
  • Update packet header (easy)
  • Longest prefix match lookup on destinations address (harder)

• Mostly implemented with specialized hardware
• **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 incoming packet

• First In First Out (FIFO) Scheduling: schedule packets in order of arrival
Packet Classification

• Classify an IP packet based on the number of fields in the packet header
  • 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 from which queue to send a packet

• Goals of scheduling algorithm
  • Fast!
  • Depends on the policy being implemented (fairness, priority, etc.)
Example: Priority Scheduler

- Packets in the highest priority queue are always served before the packets in the lower priority queues
Example: Round Robin Scheduler

- Packets are served from each queue in turn
Connecting Input to Output: Switch Fabric

- Priority Scheduler: packets are served from each queue in turn
Today’s Switch Fabrics: Mini Network!
What’s Hard About the Switch Fabric?

Queueing!
Third Generation Router: Switched Interconnects

This is called an “output queued” switch

© Nick McKeown 2006
Third Generation Router:Switched Interconnects

This is called an “input queued” switch
Reality is More Complicated

• Commercial high-speed routers use
  • Combination of input and output queueing
  • Complex multi-stage “topologies”
  • Distributed multi-stage schedulers (for scalability)
IP Routers Recap

• Core building block of Internet infrastructure

• Scalable Routing -> Longest Prefix Matching

• Need fast implementations for
  • Longest prefix matching
  • Switch fabric scheduling
Now you know as much as my PhD students :-)
IPv6
IPv6

• Motivated (prematurely) by address exhaustion
  • Address **four** times as big

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

• Result is an elegant, if unambitious, protocol
IPv4 and IPv6 Header Comparison

Field name kept from IPv4 to IPv6
- Version
- IHL
- Identification
- Time to Live (TTL)
- Protocol
- Source Address
- Destination Address
- Options

Fields not kept in IPv6
- Type of Service (TOS)
- Flags
- Fragment Offset
- Header Checksum

Name and position changed in IPv6
- Total Length
- Protocol

New field in IPv6
- Traffic Class
- Flow Label
- Payload Length
- Next Header
- Hop Limit
Summary of Changes

• Eliminated Fragmentation
• Eliminated header length
• Eliminated Checksum
• New options mechanism (next header)
• Expanded address
• Added Flow Label
IPv4 and IPv6 Header Comparison

Field name kept from IPv4 to IPv6
Fields not kept in IPv6
Name and 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
## IPv4 and IPv6 Header Comparison

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

### Differences:
- **To Destination and Back (expanded)**
- **Deal with Problems (greatly reduced)**
- **Read Correctly (reduced)**
- **Special Handling (Similar)**