Chapter 4
Network Layer: The Data Plane

Andrei Gurtov
Chapter 4: outline

4.1 Overview of Network layer
   - data plane
   - control plane

4.2 What’s inside a router

4.3 IP: Internet Protocol
   - datagram format
   - fragmentation
   - IPv4 addressing
   - network address translation
   - IPv6

4.4 Generalized Forward and SDN
   - match
   - action
   - OpenFlow examples of match-plus-action in action
Chapter 4: network layer

ch**apter goals:**

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - generalized forwarding

- instantiation, implementation in the Internet
Network layer

- transport segment from sending to receiving host
- on sending side, encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it
Two key network-layer functions

- **forwarding**: move packets from router’s input to appropriate router output
- **routing**: determine route taken by packets from source to destination
  - *routing algorithms*

analogy: *taking a trip*

- **forwarding**: process of getting through single interchange
- **routing**: process of planning trip from source to destination
Network layer: data plane, control plane

**Data plane**
- local, per-router function
- determines how datagram arriving on router input port is forwarded to router output port
- forwarding function

**Control plane**
- network-wide logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms*: implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane.
Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs)

values in arriving packet header
Network service model

Q: What service model for “channel” transporting datagrams from sender to receiver?

example services for individual datagrams:
- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a flow of datagrams:
- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing
### Network layer service models:

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>Service Model</th>
<th>Guarantees?</th>
<th>Congestion feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bandwidth</td>
<td>Loss</td>
</tr>
<tr>
<td>Internet</td>
<td>best effort</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>ATM</td>
<td>CBR</td>
<td>constant rate</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>VBR</td>
<td>guaranteed rate</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>ABR</td>
<td>guaranteed minimum</td>
<td>no</td>
</tr>
<tr>
<td>ATM</td>
<td>UBR</td>
<td>none</td>
<td>no</td>
</tr>
</tbody>
</table>
Chapter 4: outline

4.1 Overview of Network layer
  - data plane
  - control plane

4.2 What’s inside a router

4.3 IP: Internet Protocol
  - datagram format
  - fragmentation
  - IPv4 addressing
  - network address translation
  - IPv6

4.4 Generalized Forward and SDN
  - match
  - action
  - OpenFlow examples of match-plus-action in action
Router architecture overview

- high-level view of generic router architecture:

  - **routing, management control plane** (software) operates in millisecond time frame
  - **forwarding data plane** (hardware) operates in nanosecond timeframe

  - **routing processor**
  - **high-seed switching fabric**
  - **router input ports**
  - **router output ports**
Input port functions

- **line termination**
- **link layer protocol (receive)**
- **lookup, forwarding queueing**

**Physical layer:**
- Bit-level reception

**Data link layer:**
- e.g., Ethernet
- See chapter 5

**Decentralized switching:**
- Using header field values, lookup output port using forwarding table in input port memory ("match plus action")
- Goal: complete input port processing at ‘line speed’
- Queuing: if datagrams arrive faster than forwarding rate into switch fabric
Input port functions

- **Line termination**
- **Link layer protocol (receive)**
- **Lookup, forwarding, queueing**

**Switch fabric**

**Physical layer:** bit-level reception

**Data link layer:** e.g., Ethernet

**Decentralized switching:**
- using header field values, lookup output port using forwarding table in input port memory ("match plus action")
- **Destination-based forwarding:** forward based only on destination IP address (traditional)
- **Generalized forwarding:** forward based on any set of header field values
## Destination-based forwarding

### Forwarding Table

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>

**Q:** but what happens if ranges don’t divide up so nicely?
Longest prefix matching

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010*** *********</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 *********</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011*** *********</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>

examples:
- DA: 11001000 00010111 00010110 10101010
  - which interface?
- DA: 11001000 00010111 00011000 10101010
  - which interface?
Longest prefix matching

- we’ll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  
  - *content addressable*: present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM
Switching fabrics

- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics

Network Layer: Data Plane 4-18
Switching via memory

*first generation routers:*
- traditional computers with switching under direct control of CPU
- packet copied to system’s memory
- speed limited by memory bandwidth (2 bus crossings per datagram)
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- **bus contention**: switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers
Switching via interconnection network

- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network
Input port queuing

- Fabric slower than input ports combined -> queueing may occur at input queues
  - Queueing delay and loss due to input buffer overflow!
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

Output port contention:
- Only one red datagram can be transferred.
  - Lower red packet is blocked

One packet time later:
- Green packet experiences HOL blocking
Output ports

- **buffering** required from fabric faster rate
- **scheduling** datagrams

Datagram (packets) can be lost due to congestion, lack of buffers

Priority scheduling – who gets best performance, network neutrality
Output port queueing

- buffering when arrival rate via switch exceeds output line speed
- queueing (delay) and loss due to output port buffer overflow!
How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gpbs link: 2.5 Gbit buffer
- recent recommendation: with N flows, buffering equal to

\[
\frac{RTT \cdot C}{\sqrt{N}}
\]
Scheduling mechanisms

- **scheduling**: choose next packet to send on link
- **FIFO (first in first out) scheduling**: send in order of arrival to queue
  - real-world example?
  - **discard policy**: if packet arrives to full queue: who to discard?
    - **tail drop**: drop arriving packet
    - **priority**: drop/remove on priority basis
    - **random**: drop/remove randomly
**Scheduling policies: priority**

*Priority scheduling:* send highest priority queued packet

- **multiple classes, with different priorities**
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?
Scheduling policies: still more

Round Robin (RR) scheduling:

- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
- real world example?
Scheduling policies: still more

**Weighted Fair Queuing (WFQ):**
- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?

![Diagram of scheduling policies](Network Layer: Data Plane 4-29)
Chapter 4: outline

4.1 Overview of Network layer
   • data plane
   • control plane

4.2 What’s inside a router

4.3 IP: Internet Protocol
   • datagram format
   • fragmentation
   • IPv4 addressing
   • network address translation
   • IPv6

4.4 Generalized Forward and SDN
   • match
   • action
   • OpenFlow examples of match-plus-action in action
The Internet network layer

host, router network layer functions:

- **routing protocols**
  - path selection
  - RIP, OSPF, BGP

- **IP protocol**
  - addressing conventions
  - datagram format
  - packet handling conventions

- **ICMP protocol**
  - error reporting
  - router “signaling”

transport layer: TCP, UDP

network layer

link layer

physical layer
### IP datagram format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver</td>
<td>IP protocol version number</td>
</tr>
<tr>
<td>head. len</td>
<td>header length (bytes)</td>
</tr>
<tr>
<td>type of service</td>
<td>“type” of data</td>
</tr>
<tr>
<td>length</td>
<td>length</td>
</tr>
<tr>
<td>16-bit identifier</td>
<td>max number of remaining hops (decremented at each router)</td>
</tr>
<tr>
<td>flgs</td>
<td>flags</td>
</tr>
<tr>
<td>fragment offset</td>
<td>for fragmentation/reassembly</td>
</tr>
<tr>
<td>time to live</td>
<td>32 bit source IP address</td>
</tr>
<tr>
<td>upper layer</td>
<td>32 bit destination IP address</td>
</tr>
<tr>
<td>header checksum</td>
<td>options (if any)</td>
</tr>
<tr>
<td>data</td>
<td>data (variable length, typically a TCP or UDP segment)</td>
</tr>
</tbody>
</table>

#### how much overhead?
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead
IP fragmentation, reassembly

- Network links have MTU (max. transfer size) - largest possible link-level frame
  - Different link types, different MTUs
- Large IP datagram divided ("fragmented") within net
  - One datagram becomes several datagrams
  - "Reassembled" only at final destination
  - IP header bits used to identify, order related fragments
IP fragmentation, reassembly

**example:**
- 4000 byte datagram
- MTU = 1500 bytes

```
<table>
<thead>
<tr>
<th>length</th>
<th>ID</th>
<th>fragflag</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

one large datagram becomes several smaller datagrams

```
<table>
<thead>
<tr>
<th>length</th>
<th>ID</th>
<th>fragflag</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>x</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1500</td>
<td>x</td>
<td>1</td>
<td>185</td>
</tr>
<tr>
<td>1040</td>
<td>x</td>
<td>0</td>
<td>370</td>
</tr>
</tbody>
</table>
```

1480 bytes in data field

offset = 1480/8
Chapter 4: outline

4.1 Overview of Network layer
   • data plane
   • control plane
4.2 What’s inside a router
4.3 IP: Internet Protocol
   • datagram format
   • fragmentation
   • IPv4 addressing
   • network address translation
   • IPv6
4.4 Generalized Forward and SDN
   • match
   • action
   • OpenFlow examples of match-plus-action in action
IP addressing: introduction

- **IP address**: 32-bit identifier for host, router interface
- **interface**: connection between host/router and physical link
  - router’s typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- **IP addresses associated with each interface**

223.1.1.1 = 11011111 00000001 00000001 00000001
223 1 1 1 1

Network Layer: Data Plane 4-36
Q: how are interfaces actually connected?
A: we’ll learn about that in chapter 5, 6.

A: wired Ethernet interfaces connected by Ethernet switches

For now: don’t need to worry about how one interface is connected to another (with no intervening router)

A: wireless WiFi interfaces connected by WiFi base station
Subnets

- IP address:
  - subnet part - high order bits
  - host part - low order bits

- what's a subnet?
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without* intervening router

Network consisting of 3 subnets

Network Layer: Data Plane
Subnets

recipe

- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a subnet

subnet mask: /24
IP addressing: CIDR

CIDR: Classless InterDomain Routing

• subnet portion of address of arbitrary length
• address format: a.b.c.d/x, where x is # bits in subnet portion of address

```
  11001000  00010111  00010000  00000000
```

```
  subnet part

  11001000  00010111  00010000  00000000
  host part

  200.23.16.0/23
```
Q: How does a *host* get IP address?

- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config
- **DHCP**: Dynamic Host Configuration Protocol: dynamically get address from as server
  - “plug-and-play”
DHCP: Dynamic Host Configuration Protocol

**goal:** allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/“on”)
- support for mobile users who want to join network (more shortly)

**DHCP overview:**

- host broadcasts “DHCP discover” msg [optional]
- DHCP server responds with “DHCP offer” msg [optional]
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg
DHCP client-server scenario

DHCP client needs address in this network

Network Layer: Data Plane  4-44
DHCP client-server scenario

DHCP server: 223.1.2.5

DHCP discover

Broadcast: is there a DHCP server out there?

DHCP offer

Broadcast: I’m a DHCP server! Here’s an IP address you can use

DHCP request

Broadcast: OK. I’ll take that IP address!

DHCP ACK

Broadcast: OK. You’ve got that IP address!
DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

- address of first-hop router for client
- name and IP address of DNS server
- network mask (indicating network versus host portion of address)
connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP

DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet

Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server

Ethernet demuxed to IP demuxed, UDP demuxed to DHCP
- DCP server formulates DHCP ACK containing client’s IP address, IP address of first-hop router for client, name & IP address of DNS server.
- Encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client.
- Client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router.
DHCP: Wireshark output (home LAN)

Message type: **Boot Request (1)**
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
**Transaction ID: 0x6b3a11b7**
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)

Option: (t=53,l=1) **DHCP Message Type = DHCP Request**
Option: (61) **Client identifier**
  Length: 7; Value: 010016D323688A;
  Hardware type: Ethernet
  Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Option: (t=50,l=4) **Requested IP Address = 192.168.1.101**
Option: (t=12,l=5) **Host Name = "nomad"**
**Option: (55) Parameter Request List**
  Length: 11; Value: 010F03062C2E2F1F21F92B
  1 = Subnet Mask; 15 = Domain Name
  3 = Router; 6 = Domain Name Server
  44 = NetBIOS over TCP/IP Name Server

Network Layer: Data Plane
**IP addresses: how to get one?**

**Q:** how does network get subnet part of IP addr?

**A:** gets allocated portion of its provider ISP’s address space

<table>
<thead>
<tr>
<th>ISP's block</th>
<th>11001000 00010111 00010000 00000000</th>
<th>200.23.16.0/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization 0</td>
<td>11001000 00010111 00010000 00000000</td>
<td>200.23.16.0/23</td>
</tr>
<tr>
<td>Organization 1</td>
<td>11001000 00010111 00010010 00000000</td>
<td>200.23.18.0/23</td>
</tr>
<tr>
<td>Organization 2</td>
<td>11001000 00010111 00010100 00000000</td>
<td>200.23.20.0/23</td>
</tr>
<tr>
<td>...</td>
<td>...                         ...</td>
<td></td>
</tr>
<tr>
<td>Organization 7</td>
<td>11001000 00010111 00011110 00000000</td>
<td>200.23.30.0/23</td>
</tr>
</tbody>
</table>
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:

- Organization 0
  - 200.23.16.0/23
- Organization 1
  - 200.23.18.0/23
- Organization 2
  - 200.23.20.0/23
- Organization 7
  - 200.23.30.0/23

Fly-By-Night-ISP

- “Send me anything with addresses beginning 200.23.16.0/20”

ISPs-R-Us

- “Send me anything with addresses beginning 199.31.0.0/16”

Internet
Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1

Organization 0
- 200.23.16.0/23

Organization 2
- 200.23.20.0/23

Organization 7
- 200.23.30.0/23

ISPs-R-Us

Fly-By-Night-ISP

“Send me anything with addresses beginning 200.23.16.0/20”

“Send me anything with addresses beginning 199.31.0.0/16 or 200.23.18.0/23”

Internet
IP addressing: the last word...

Q: how does an ISP get block of addresses?
A: ICANN: Internet Corporation for Assigned Names and Numbers http://www.icann.org/
  • allocates addresses
  • manages DNS
  • assigns domain names, resolves disputes
NAT: network address translation

all datagrams leaving local network have same single source NAT IP address: 138.76.29.7, different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)
**NAT: network address translation**

*motivation:* local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)
**NAT: network address translation**

**implementation:** NAT router must:

- **outgoing datagrams:** replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #). . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr

- **remember (in NAT translation table)** every (source IP address, port #) to (NAT IP address, new port #) translation pair

- **incoming datagrams:** replace (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table
**NAT: network address translation**

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

NAT translation table

<table>
<thead>
<tr>
<th>WAN side addr</th>
<th>LAN side addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>138.76.29.7, 5001</td>
<td>10.0.0.1, 3345</td>
</tr>
<tr>
<td>……</td>
<td>……</td>
</tr>
</tbody>
</table>

3: reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

---

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/
NAT: network address translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!

- NAT is controversial:
  - routers should only process up to layer 3
  - address shortage should be solved by IPv6
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - NAT traversal: what if client wants to connect to server behind NAT?
Chapter 4: outline

4.1 Overview of Network layer
   • data plane
   • control plane

4.2 What’s inside a router

4.3 IP: Internet Protocol
   • datagram format
   • fragmentation
   • IPv4 addressing
   • network address translation
   • IPv6

4.4 Generalized Forward and SDN
   • match
   • action
   • OpenFlow examples of match-plus-action in action
IPv6: motivation

- initial motivation: 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:
- fixed-length 40 byte header
- no fragmentation allowed
**IPv6 datagram format**

**priority:** identify priority among datagrams in flow

**flow Label:** identify datagrams in same “flow.”

(concept of “flow” not well defined).

**next header:** identify upper layer protocol for data

<table>
<thead>
<tr>
<th>ver</th>
<th>pri</th>
<th>flow label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>payload len</th>
<th>next hdr</th>
<th>hop limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>source address</th>
<th>(128 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>destination address</th>
<th>(128 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data</th>
</tr>
</thead>
</table>
Other changes from IPv4

- **checksum**: removed entirely to reduce processing time at each hop
- **options**: allowed, but outside of header, indicated by “Next Header” field
- **ICMPv6**: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions
Transition from IPv4 to IPv6

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

logical view:
A IPv6
IPv6
IPv6
IPv6
E IPv6
IPv6

physical view:
A IPv6
IPv6
IPv6
IPv6
C IPv4
IPv4
D IPv4
IPv4
E IPv6
IPv6
IPv6
IPv6

IPv4 tunnel connecting IPv6 routers

Network Layer: Data Plane 4-64
Tunneling

**logical view:**
- IPv6
- IPv6
- IPv6

**physical view:**
- IPv6
- IPv6
- IPv4
- IPv4

**flow:** X
**src:** A
**dest:** F
**data**

**A-to-B:** IPv6

**flow:** X
**src:** B
**dest:** E
**Flow:** X
**Src:** A
**Dest:** F
**data**

**B-to-C:** IPv6 inside IPv4

**flow:** X
**src:** B
**dest:** E
**Flow:** X
**Src:** A
**Dest:** F
**data**

**E-to-F:** IPv6

**IPv4 tunnel connecting IPv6 routers**
IPv6: adoption

- Google: 8% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

- **Long (long!) time for deployment, use**
  - 20 years and counting!
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, …
  - Why?
Chapter 4: outline

4.1 Overview of Network layer
   - data plane
   - control plane

4.2 What’s inside a router

4.3 IP: Internet Protocol
   - datagram format
   - fragmentation
   - IPv4 addressing
   - network address translation
   - IPv6

4.4 Generalized Forward and SDN
   - match
   - action
   - OpenFlow examples of match-plus-action in action
Generalized Forwarding and SDN

Each router contains a flow table that is computed and distributed by a logically centralized routing controller.
OpenFlow data plane abstraction

- *flow*: defined by header fields
- generalized forwarding: simple packet-handling rules
  - *Pattern*: match values in packet header fields
  - *Actions*: for *matched packet*: drop, forward, modify, matched packet or send matched packet to controller
  - *Priority*: disambiguate overlapping patterns
  - *Counters*: #bytes and #packets

Flow table in a router (computed and distributed by controller) define router’s match+action rules
OpenFlow data plane abstraction

- **flow**: defined by header fields
- **generalized forwarding**: simple packet-handling rules
  - **Pattern**: match values in packet header fields
  - **Actions**: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - **Priority**: disambiguate overlapping patterns
  - **Counters**: #bytes and #packets

1. src=1.2.*.*, dest=3.4.5.* → drop
2. src = *.*.*.*, dest=3.4.*.* → forward(2)
3. src=10.1.2.3, dest=.*.*.*.* → send to controller

* : wildcard
## OpenFlow: Flow Table Entries

### Rule
- Switch Port
- VLAN ID
- MAC src
- MAC dst
- Eth type
- IP Src
- IP Dst
- IP Prot
- TCP sport
- TCP dport

### Action
- Forward packet to port(s)
- Encapsulate and forward to controller
- Drop packet
- Send to normal processing pipeline
- Modify Fields

### Stats
- Packet + byte counters

### Diagram
- Link layer
- Network layer
- Transport layer
## Examples

### Destination-based forwarding:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

*IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6*

### Firewall:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>

*do not forward (block) all datagrams destined to TCP port 22*

<table>
<thead>
<tr>
<th>Switch</th>
<th>Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>drop</td>
</tr>
</tbody>
</table>

*do not forward (block) all datagrams sent by host 128.119.1.1*
**Destination-based layer 2 (switch) forwarding:**

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port3</td>
</tr>
</tbody>
</table>

*layer 2 frames from MAC address 22:A7:23:11:E1:02 should be forwarded to output port 6*
OpenFlow abstraction

- **match+action**: unifies different kinds of devices

- **Router**
  - **match**: longest destination IP prefix
  - **action**: forward out a link

- **Switch**
  - **match**: destination MAC address
  - **action**: forward or flood

- **Firewall**
  - **match**: IP addresses and TCP/UDP port numbers
  - **action**: permit or deny

- **NAT**
  - **match**: IP address and port
  - **action**: rewrite address and port
OpenFlow example

Example: datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

<table>
<thead>
<tr>
<th>match</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Src = 10.3.<em>.</em></td>
<td>forward(3)</td>
</tr>
<tr>
<td>IP Dst = 10.2.<em>.</em></td>
<td>forward(3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>match</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ingress port = 1</td>
<td>forward(4)</td>
</tr>
<tr>
<td>IP Src = 10.3.<em>.</em></td>
<td>forward(4)</td>
</tr>
<tr>
<td>IP Dst = 10.2.<em>.</em></td>
<td>forward(4)</td>
</tr>
</tbody>
</table>

Example:
datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2
Chapter 4: done!

4.1 Overview of Network layer: data plane and control plane
4.2 What’s inside a router
4.3 IP: Internet Protocol
   • datagram format
   • fragmentation
   • IPv4 addressing
   • NAT
   • IPv6
4.4 Generalized Forward and SDN
   • match plus action
   • OpenFlow example

**Question:** how do forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

**Answer:** by the control plane (next chapter)