Chapter 4: network layer

Chapter 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

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

analogy: taking a trip
- **forwarding**: process of getting through single interchange
- **routing**: process of planning trip from source to destination

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.

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</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>VBR</td>
<td>guaranteed</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>ABR</td>
<td>guaranteed</td>
<td>no</td>
</tr>
<tr>
<td>ATM</td>
<td>UBR</td>
<td>none</td>
<td>no</td>
</tr>
</tbody>
</table>
4.2: 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

- high-seed switching fabric

- router input ports

- router output ports

Input port functions

- 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

- 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

- **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 10100001 which interface?
DA: 11001000 00010111 0001100 10101010 which interface?

Switching fabrics

- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer 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

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 ports

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

This slide in HUGELY important!

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 $rac{RTT \cdot C}{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?

![Round Robin Scheduling Diagram]

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

![Weighted Fair Queuing Diagram]

4.3: The Internet network layer

Host, router network layer functions:
- transport layer: TCP, UDP
  - routing protocols: path selection, RIP, OSPF, BGP
  - IP protocol: addressing conventions, datagram format, packet handling conventions
  - ICMP protocol: error reporting, router signaling

**IP datagram format**

- total datagram length (bytes) for fragmentation/reassembly
- 32 bit source IP address
- 32 bit destination IP address
- options (if any)
- data (variable length, typically a TCP or UDP segment)
- max number of remaining hops (decremented at each router)
- max number of remaining hops
- time to live
- upper layer protocol to deliver payload to
- upper layer protocol
- flags
- fragment offset
- header checksum
- type of service
- length
- header length (bytes)
- “type” of data
- version number
- data (variable length, typically a TCP or UDP segment)
- e.g. timestamp, record route taken, specify list of routers to visit.

- 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

**Example:**

- 4000 byte datagram
- MTU = 1500 bytes

**Fragmentation:**

<table>
<thead>
<tr>
<th>Length</th>
<th>ID</th>
<th>Fragment</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1480</td>
<td>x</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Reassembly:**

- One large datagram becomes several smaller datagrams

**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**

**Question:** How are interfaces actually connected?

**Answer:** 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*

---

**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
```

```
200.23.16.0/23
```
IP addresses: how to get one?

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 server: 223.1.2.5 arriving client

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)

DHCP: example

- 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: FFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP

DHCP: Wireshark output (home LAN)

- 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 DSN server, IP address of its first-hop router
**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 00010101 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 00111110 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**
- **Organization 1**
- **Organization 2**
- **Organization 7**

**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/](http://www.icann.org/)  
- allocates addresses  
- manages DNS  
- assigns domain names, resolves disputes
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)

**implementation:** NAT router must:

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

- **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

- 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?

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>
<tr>
<td>payload len</td>
<td>next hdr</td>
<td>hop limit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>source address</td>
<td>hop limit</td>
<td></td>
</tr>
<tr>
<td>(128 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(128 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 bits

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

---

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?
4.4: 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: Flow Table Entries

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. Modify Fields

**Rule** | **Action** | **Stats**
--- | --- | ---
**Packet + byte counters**

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>VLAN ID</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link layer</td>
<td>Network layer</td>
<td>Transport layer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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
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)