Network Layer: Part I

Network Layer and IP Protocol!

Part I:

- Network Layer Functions and Service Models
  - Network Layer Functions
  - IP Addressing
  - Network Service Models: VC vs. Datagram
- IP Forwarding and IP Protocol
  - IP Datagram Forwarding Model
  - IP and ICMP: Datagram Format, IP Fragmentation, ...
  - DHCP
- Router Architecture

Readings: Textbook: Chapter 4, Sections 4.1-4.4, review section 1.3.2 (datagram vs. virtual circuit)
What Does Network Layer Do?

• **End-to-end deliver packet from sending to receiving hosts, “hop-by-hop” thru network**
  - A network-wide concern!
  - Involves every router, host in the network

• **Compare:**
  - **Transport layer**
    • between two end hosts
  - **Data link layer**
    • over a physical link directly connecting two (or more) physically hosts
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
Network Layer Functions

• Addressing
  - Globally unique address for each routable device
    • Logical address, unlike MAC address (as you’ll see later)
  - Assigned by network operator
    • Need to map to MAC address (as you’ll see later)

• Routing: building a “map” of network
  - Which path to use to forward packets from src to dest

• Forwarding: delivery of packets hop by hop
  - From input port to appropriate output port in a router

Routing and forwarding depend on network service models: datagram vs. virtual circuit
Two Key Network-Layer Functions

- **forwarding**: move packets from router’s input to appropriate router output
  - *analogy*:
    - **routing**: process of planning trip from source to dest
    - **forwarding**: process of getting through single interchange

- **routing**: determine route taken by packets from source to dest.
  - *routing algorithms*
IP Addressing: Basics

- **Globally unique** (for “public” IP addresses)
- **IP address**: 32-bit identifier for host, router interface
- **Interface**: connection between host/router and physical link
  - router’s typically have multiple interfaces
  - host may have multiple interfaces
  - IP addresses associated with each interface
- **Dot notation** (for ease of human reading)

\[
223.1.1.1 = 11011111 \ 00000001 \ 00000001 \ 00000001
\]

\[
\begin{array}{cccc}
223 & 1 & 1 & 1
\end{array}
\]
IP Addressing: Network vs. Host

- Two-level hierarchy
  - network part (high order bits)
  - host part (low order bits)
- *What’s a network?* (from IP address perspective)
  - device interfaces with same network part of IP address
  - can physically reach each other without intervening router
“Classful” IP Addressing

- Class A: 0.0.0.0 to 127.255.255.255
- Class B: 128.0.0.0 to 191.255.255.255
- Class C: 192.0.0.0 to 223.255.255.255
- Class D: 224.0.0.0 to 239.255.255.255

- Disadvantage: inefficient use of address space, address space exhaustion
- e.g., class B net allocated enough addresses for 65K hosts, even if only 2K hosts in that network
**Classless Addressing: CIDR**

**CIDR: Classless InterDomain Routing**

- Network portion of address is of arbitrary length
- Addresses allocated in contiguous blocks
  - Number of addresses assigned always power of 2
- Address format: `a.b.c.d/x`
  - `x` is number of bits in network portion of address

```
11001000  00010111  00010000  00000000
```

```
200.23.16.0/23
```
Special IP Addresses

- Network address: host id = all 0’s
- Directed broadcast address: host id = all 1’s
- Local broadcast address: all 1’s
- Local host address (this computer): all 0’s
- Loopback address
  - network id = 127, any host id (e.g. 127.0.0.1)
IP Addresses: How to Get One?

Q: How does host get IP address?

• “static” assigned: i.e., hard-coded in a file
  - Wintel: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config

• Dynamically assigned: using 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 an “on”
- Support for mobile users who want to join network (more shortly)

**DHCP overview:**
- host broadcasts “DHCP discover” msg
- DHCP server responds with “DHCP offer” msg
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg
DHCP Client-Server Scenario

arriving DHCP client needs address in this network

A 223.1.1.1
223.1.1.2
223.1.1.3
223.1.1.4

B
223.1.1.3
223.1.3.1
223.1.3.2
223.1.3.27

DHCP server 223.1.2.1

223.1.2.2

E

arriving DHCP client needs address in this network
DHCP Client-Server Scenario

DHCP server: 223.1.2.5

DHCP discover
src: 0.0.0.0, 68
dest.: 255.255.255.255, 67
yiaddr: 0.0.0.0
transaction ID: 654

DHCP offer
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 654
Lifetime: 3600 secs

DHCP request
src: 0.0.0.0, 68
dest:: 255.255.255.255, 67
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

DHCP ACK
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs
IP Addresses: How to Get One? ...

**Q:** How does network get network 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 <em>00010111</em> <em>00010000</em> <em>00000000</em></th>
<th>200.23.16.0/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization 0</td>
<td>11001000 <em>00010111</em> <em>00010000</em> <em>00000000</em></td>
<td>200.23.16.0/23</td>
</tr>
<tr>
<td>Organization 1</td>
<td>11001000 <em>00010111</em> <em>00010010</em> <em>00000000</em></td>
<td>200.23.18.0/23</td>
</tr>
<tr>
<td>Organization 2</td>
<td>11001000 <em>00010111</em> <em>00010100</em> <em>00000000</em></td>
<td>200.23.20.0/23</td>
</tr>
<tr>
<td>Organization 7</td>
<td>11001000 <em>00010111</em> <em>00011110</em> <em>00000000</em></td>
<td>200.23.30.0/23</td>
</tr>
</tbody>
</table>
IP Addressing: the Last Word...

Q: How does an ISP get block of addresses?

A: **ICANN**: Internet Corporation for Assigned Names and Numbers
- allocates addresses
- manages DNS
- assigns domain names, resolves disputes
Routing & Forwarding: Logical View of a Router

Control Plane (Routing Processor)
- routing agent (daemon)
- routing table
  - update

Data Plane (Forwarding Engine)
- switching fabric
- input interfaces (ports)
- output interfaces (ports)

Routing & Forwarding:
- logical view of a router

input links
- routing information
Interplay between routing and forwarding

<table>
<thead>
<tr>
<th>Header Value</th>
<th>Output Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

value in arriving packet’s header

routing algorithm

local forwarding table
Network Service Model

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

- guaranteed bandwidth?
- preservation of inter-packet timing (no jitter)?
- loss-free delivery?
- in-order delivery?
- congestion feedback to sender?

The most important abstraction provided by network layer:

virtual circuit

or
datagram?
Network Service Model (cont’d)

Some Possible Examples:

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 Connection vs. Connectionless Service

- datagram network provides network-layer connectionless service
- VC network provides network-layer connection service
- analogous to the transport-layer services, but:
  - service: host-to-host
  - no choice: network provides one or the other
  - implementation: in network core

- network vs transport layer connection service:
  - network: between two hosts, in case of VCs, also involve intervening routers
  - transport: between two processes
Virtual Circuit vs. Datagram

- Objective of both: move packets through routers from source to destination

- Datagram Model:
  - **Routing**: determine next hop to each destination a priori
  - **Forwarding**: destination address in packet header, used at each hop to look up for next hop
    - routes may change during “session”
    - analogy: driving, asking directions at every gas station, or based on the road signs at every turn

- Virtual Circuit Model:
  - **Routing**: determine a path from source to each destination
  - “Call” Set-up: fixed path (“virtual circuit”) set up at “call” setup time, remains fixed thru “call”
  - **Data Forwarding**: each packet carries “tag” or “label” (virtual circuit id, VCI), which determines next hop
  - routers maintain “per-call” state
Virtual Circuits

“source-to-dest path behaves much like telephone circuit” (but actually over packet network)
- performance-wise
- network actions along source-to-dest path

• call setup/teardown for each call before data can flow
  - need special control protocol: “signaling”
  - every router on source-dest path maintains “state” (VCI translation table) for each passing call
  - VCI translation table at routers along the path of a call “weaving together” a “logical connection” for the call

• link, router resources (bandwidth, buffers) may be reserved and allocated to each VC
  - to get “circuit-like” performance

• Compare w/ transport-layer “connection”: only involves two end systems, no fixed path, can’t reserve bandwidth!
VC Implementation

A VC consists of:

1. path from source to destination
2. VC numbers, one number for each link along path
3. entries in forwarding tables in routers along path

- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
  - New VC number comes from forwarding table
VC Translation/Forwarding Table

### Forwarding table in northwest router:

<table>
<thead>
<tr>
<th>Incoming interface</th>
<th>Incoming VC #</th>
<th>Outgoing interface</th>
<th>Outgoing VC #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Routers maintain connection state information!
During data packet forwarding phase, input VCI is used to look up the table, and is “swapped” w/ output VCI (VCI translation, or “label swapping”).
Virtual Circuit: Example

“call” from host A to host B along path:
host A → router 1 → router 2 → router 3 → host B

• each router along path maintains an entry for the call in its VCI translation table
• the entries piece together a “logical connection” for the call

• Exercise: write down the VCI translation table entry for the call at each router
Virtual Circuit: Signaling Protocols

- used to setup, maintain, teardown VC
- used in ATM, frame-relay, X.25
- used in part of today’s Internet: Multi-Protocol Label Switching (MPLS) operated at “layer 2+1/2” (between data link layer and network layer) for “traffic engineering” purpose
Virtual Circuit Setup/Teardown

Call Set-Up:
- Source: select a path from source to destination
  - Use routing table (which provides a “map of network”)
- Source: send VC setup request control (“signaling”) packet
  - Specify path for the call, and also the (initial) output VCI
  - perhaps also resources to be reserved, if supported
- Each router along the path:
  - Determine output port and choose a (local) output VCI for the call
    - need to ensure that no two distinct VCs leaving the same output port have the same VCI!
  - Update VCI translation table (“forwarding table”)
    - add an entry, establishing a mapping between incoming VCI & port no. and outgoing VCI & port no. for the call

Call Tear-Down: similar, but remove entry instead
Datagram Networks: the Internet model

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of “connection”
- packets forwarded using destination host address
  - packets between same source-dest pair may take different paths, when intermediate routes change!
**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>

4 billion possible entries
# IP Forwarding Table

4 billion possible entries!
(in reality, far less, but can still have millions of “routes”)

<table>
<thead>
<tr>
<th>forwarding table entry format</th>
<th>destination network</th>
<th>next-hop (IP address)</th>
<th>link interface</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11001000 00010111 00010000 00000000, 11111111 11111111 11111000 00000000</td>
<td>200.23.16.1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>11001000 00010111 00011000 00000000, 11111111 11111111 11111111 00000000</td>
<td>- (direct)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>11001000 00010111 00011001 00000000, 11111111 11111111 11111000 00000000</td>
<td>200.23.25.6</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>11001000 00010111 00010000 00000000, 11111111 11111111 11111000 00000000</td>
<td>128.30.0.1</td>
<td>3</td>
</tr>
</tbody>
</table>
## Longest Prefix Matching

<table>
<thead>
<tr>
<th>Prefix Match</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 00011000 10101010**
  - Which interface?
IP Datagram Forwarding Model

IP Datagram:

- Datagram remains unchanged, as it travels source to destination
- Addr fields of interest here

<table>
<thead>
<tr>
<th>misc fields</th>
<th>source IP addr</th>
<th>dest IP addr</th>
<th>data</th>
</tr>
</thead>
</table>

Forwarding Table in A:

<table>
<thead>
<tr>
<th>Dest. Net.</th>
<th>next router</th>
<th>Nhops</th>
</tr>
</thead>
<tbody>
<tr>
<td>223.1.1</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
<tr>
<td>223.1.2</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
<tr>
<td>223.1.3</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
</tbody>
</table>
IP Forwarding: Destination in Same Net

Starting at A, send IP datagram addressed to B:

- look up net. address of B in forwarding table
- find B is on same net. as A
- link layer will send datagram directly to B inside link-layer frame
  - B and A are directly connected

<table>
<thead>
<tr>
<th>Dest. Net.</th>
<th>next router</th>
<th>Nhops</th>
</tr>
</thead>
<tbody>
<tr>
<td>223.1.1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>223.1.2</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
<tr>
<td>223.1.3</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
</tbody>
</table>
### IP Forwarding: Destination in Diff. Net

**Starting at A, dest. E:**
- Look up network address of E in forwarding table.
- E on *different* network.
  - A, E not directly attached.
- Routing table: next hop router to E is 223.1.1.4.
- Link layer sends datagram to router 223.1.1.4 inside link-layer frame.
- Datagram arrives at 223.1.1.4.
- Continued.....

### Forwarding Table in A

<table>
<thead>
<tr>
<th>Dest. Net.</th>
<th>next router</th>
<th>Nhops</th>
</tr>
</thead>
<tbody>
<tr>
<td>223.1.1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>223.1.2</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
<tr>
<td>223.1.3</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
</tbody>
</table>

**Misc:**
- 223.1.1.1
- 223.1.2.3
- Data

**Diagram:**
- Diagram showing network topology with routers and connections.
Arriving at 223.1.4, destined for 223.1.2.2
- look up network address of E in router’s forwarding table
- E on *same* network as router’s interface 223.1.2.9
  - router, E directly attached
- link layer sends datagram to 223.1.2.2 inside link-layer frame via interface 223.1.2.9
- datagram arrives at 223.1.2.2!!! (hooray!)
Virtual Circuit vs. Datagram

- Objective of both: move packets through routers from source to destination

- Datagram Model:
  - **Routing**: determine next hop to each destination a priori
  - **Forwarding**: destination address in packet header, used at each hop to look up for next hop
    - routes may change during “session”
  - analogy: driving, asking directions at every gas station, or based on the road signs at every turn

- Virtual Circuit Model:
  - **Routing**: determine a path from source to each destination
  - “Call” Set-up: fixed path (“virtual circuit”) set up at “call” setup time, remains fixed thru “call”
  - **Data Forwarding**: each packet carries “tag” or “label” (virtual circuit id, VCI), which determines next hop
  - routers maintain “per-call” state
## 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>

- Internet model being extended: MPLS, Intserv, Diffserv
  - Section 5.8, sections 7.7-7.9
Datagram or VC: Why?

Internet
- data exchange among computers
  - “elastic” service, no strict timing req.
- “smart” end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at “edge”
- many link types
  - different characteristics
  - uniform service difficult

ATM
- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- “dumb” end systems
  - telephones
  - complexity inside network

MPLS (layer 2 1/2)
- evolve from ATM
  - traffic engineering, fast path restoration (a priori “backup” paths)
Routing & Forwarding: Logical View of a Router

Control Plane (Routing Processor)

routing agent (daemon)

routing table

update

lookup

forwarding table

update

Data Plane (Forwarding Engine)

switching fabric

input interfaces (ports)

output interfaces (ports)

routing information

input links
IP Forwarding & IP/ICMP Protocol

Transport layer: TCP, UDP

Routing protocols
- path selection
- RIP, OSPF, BGP

IP protocol
- addressing conventions
- packet handling conventions

ICMP protocol
- error reporting
- router “signaling”

Data Link layer (Ethernet, WiFi, PPP, …)

Physical Layer (SONET, …)
**IP Datagram Format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version number</td>
<td>Number of data type.</td>
</tr>
<tr>
<td>Header length (bytes)</td>
<td>Total length of header in bytes.</td>
</tr>
<tr>
<td>&quot;Type&quot; of data</td>
<td>Type of data (e.g., IP, TCP, UDP).</td>
</tr>
<tr>
<td>Max number remaining hops</td>
<td>(Decremental at each router)</td>
</tr>
<tr>
<td>Source IP address</td>
<td>32-bit address for source.</td>
</tr>
<tr>
<td>Destination IP address</td>
<td>32-bit address for destination.</td>
</tr>
<tr>
<td>Options (if any)</td>
<td>Options field (e.g., timestamp, record route).</td>
</tr>
<tr>
<td>Data</td>
<td>Payload data (variable length, typically a TCP or UDP segment).</td>
</tr>
</tbody>
</table>

**Note:**

- How much overhead with TCP?
  - 20 bytes of TCP
  - 20 bytes of IP
  - = 40 bytes + app layer overhead

**E.g.:**

- Timestamp, record route taken, specify list of routers to visit.
### IP Datagram Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version number</td>
<td>32 bits</td>
</tr>
<tr>
<td>header length</td>
<td>(bytes)</td>
</tr>
<tr>
<td>“type” of data</td>
<td>16-bit identifier</td>
</tr>
<tr>
<td>max number remaining hops (decremented at each router)</td>
<td>time to live</td>
</tr>
<tr>
<td>upper layer protocol to deliver payload to</td>
<td>upper layer</td>
</tr>
<tr>
<td>data</td>
<td>Options (if any)</td>
</tr>
<tr>
<td>options (if any)</td>
<td>E.g. timestamp, record route taken, specify list of routers to visit.</td>
</tr>
</tbody>
</table>

**How much overhead with TCP?**
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

**Total datagram length (bytes) for fragmentation/reassembly**

**Internet checksum**

**Max number remaining hops**

**Total datagram length (bytes)**

**Upper layer protocol**

**Upper layer protocol**

**32 bit source IP address**

**32 bit destination IP address**

**Data (variable length, typically a TCP or UDP segment)**

**Header length (bytes)**

**“Type” of data**

**Internet checksum**

**Time to live**

**Upper layer**

**Header length (bytes)**

**Upper layer protocol**

**32 bit source IP address**

**32 bit destination IP address**

**Options (if any)**

**Data (variable length, typically a TCP or UDP segment)**

**Header length (bytes)**

**Upper layer protocol**

**32 bit source IP address**

**32 bit destination IP address**

**Options (if any)**

**Data (variable length, typically a TCP or UDP segment)**

**Header length (bytes)**

**Upper layer protocol**

**32 bit source IP address**

**32 bit destination IP address**

**Options (if any)**

**Data (variable length, typically a TCP or UDP segment)**
Fields in IP Datagram

- **IP protocol version:** current version is 4, IPv4, new: IPv6
- **Header length:** number of 32-bit words in the header
- **Type of Service:**
  - 3-bit priority, e.g., delay, throughput, reliability bits, ...
- **Total length:** including header (maximum 65535 bytes)
- **Identification:** all fragments of a packet have same identification
- **Flags:** don’t fragment, more fragments
- **Fragment offset:** where in the original packet (count in 8 byte units)
- **Time to live:** maximum life time of a packet
- **Protocol Type:** e.g., ICMP, TCP, UDP etc
- **IP Option:** non-default processing, e.g., IP source routing option, etc.
IP Fragmentation & Reassembly: Why

- 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: How

- An IP datagram is chopped by a router into smaller pieces if
  - datagram size is greater than network MTU
  - Don’t fragment option is not set
- Each datagram has unique datagram identification
  - Generated by source hosts
  - All fragments of a packet carry original datagram id
- All fragments except the last have more flag set
  - Fragment offset and Length fields are modified appropriately
- Fragments of IP packet can be further fragmented by other routers along the way to destination!
- Reassembly only done at destination host (why?)
  - Use IP datagram id, fragment offset, fragment flags. Length
  - A timer is set when first fragment is received (why?)
IP Fragmentation and Reassembly: Exp

Example

- 4000 byte datagram
- MTU = 1500 bytes

One large datagram becomes several smaller datagrams

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>ID</td>
<td>fragflag</td>
<td>offset</td>
</tr>
<tr>
<td>=4000</td>
<td>=x</td>
<td>=0</td>
<td>=0</td>
</tr>
<tr>
<td>length</td>
<td>ID</td>
<td>fragflag</td>
<td>offset</td>
</tr>
<tr>
<td>=1500</td>
<td>=x</td>
<td>=1</td>
<td>=0</td>
</tr>
<tr>
<td>length</td>
<td>ID</td>
<td>fragflag</td>
<td>offset</td>
</tr>
<tr>
<td>=1500</td>
<td>=x</td>
<td>=1</td>
<td>=1480</td>
</tr>
<tr>
<td>length</td>
<td>ID</td>
<td>fragflag</td>
<td>offset</td>
</tr>
<tr>
<td>=1040</td>
<td>=x</td>
<td>=0</td>
<td>=2960</td>
</tr>
</tbody>
</table>
ICMP: Internet Control Message Protocol

- used by hosts, routers, gateways to communication network-level information
  - error reporting: unreachable host, network, port, protocol
    - echo request/reply (used by ping)
  - network-layer “above” IP:
    - ICMP msgs carried in IP datagrams
  - ICMP message: type, code plus first 8 bytes of IP datagram causing error

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>echo reply (ping)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>dest. network unreachable</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>dest host unreachable</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>dest protocol unreachable</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>dest port unreachable</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>dest network unknown</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>dest host unknown</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>source quench (congestion control - not used)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>echo request (ping)</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>route advertisement</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>router discovery</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>TTL expired</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>bad IP header</td>
</tr>
</tbody>
</table>
ICMP Message Transport & Usage

• ICMP messages carried in IP datagrams
• Treated like any other datagrams
  - But no error message sent if ICMP message causes error
• Message sent to the source
  - 8 bytes of the original header included
• ICMP Usage (non-error, informational): Examples
  - Testing reachability: ICMP echo request/reply
    • ping
  - Tracing route to a destination: Time-to-live field
    • traceroute
  - Path MTU discovery
    • Don’t fragment bit
  - IP direct (for hosts only): inform hosts of better routes
Routing & Forwarding: Logical View of a Router

Control Plane (Routing Processor)

Routing agent (daemon)

Routing table

Update

Data Plane (Forwarding Engine)

Switching fabric

Input interfaces (ports)

Output interfaces (ports)
Router Architecture Overview

Two key router functions:
- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link
Input Port Functions

Decentralized switching:

- given datagram dest., lookup output port using forwarding table in input port memory
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Physical layer:
bit-level reception

Data link layer:
e.g., Ethernet
see chapter 5
Three Types of Switching Fabrics

memory

bus

crossbar

CSci4211: Network Layer: Part I
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
- 1 Gbps bus, Cisco 1900: sufficient speed for access and enterprise routers (not regional or backbone)
Switching Via An Interconnection Network

• overcome bus bandwidth limitations
• Banyan networks, 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 Gbps through the interconnection network
Output Ports

- **Buffering** required when datagrams arrive from fabric faster than the transmission rate
- **Scheduling discipline** chooses among queued datagrams for transmission
Output Port Queueing

- buffering when arrival rate via switch exceeds output line speed
- queueing (delay) and loss due to output port buffer overflow!
Input Port Queuing

• Fabric slower than input ports combined -> queueing may occur at input queues

• Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

• queueing delay and loss due to input buffer overflow!

output port contention at time t - only one red packet can be transferred

green packet experiences HOL blocking
Network Layer: Part I Summary

- **Network Layer Functions**
  - Addressing, Routing and Forwarding
- **IP Addressing Scheme: CIDR**
  - DHCP
- **Network Service Models**
  - Virtual Circuit vs. Datagram
  - Virtual Circuit Model
    - VC set-up/tear-down
    - data forward operations
- **IP Forwarding and IP Protocol**
  - IP Datagram Forwarding Model: dest. in same net vs. diff. net
  - IP and ICMP: Datagram Format, IP Fragmentation, ...