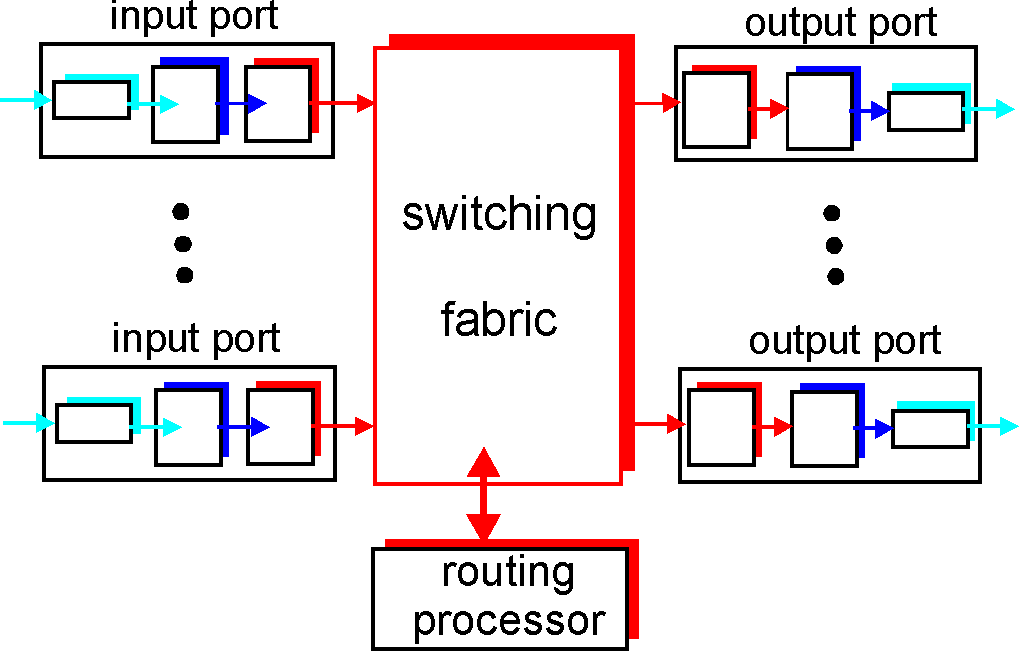
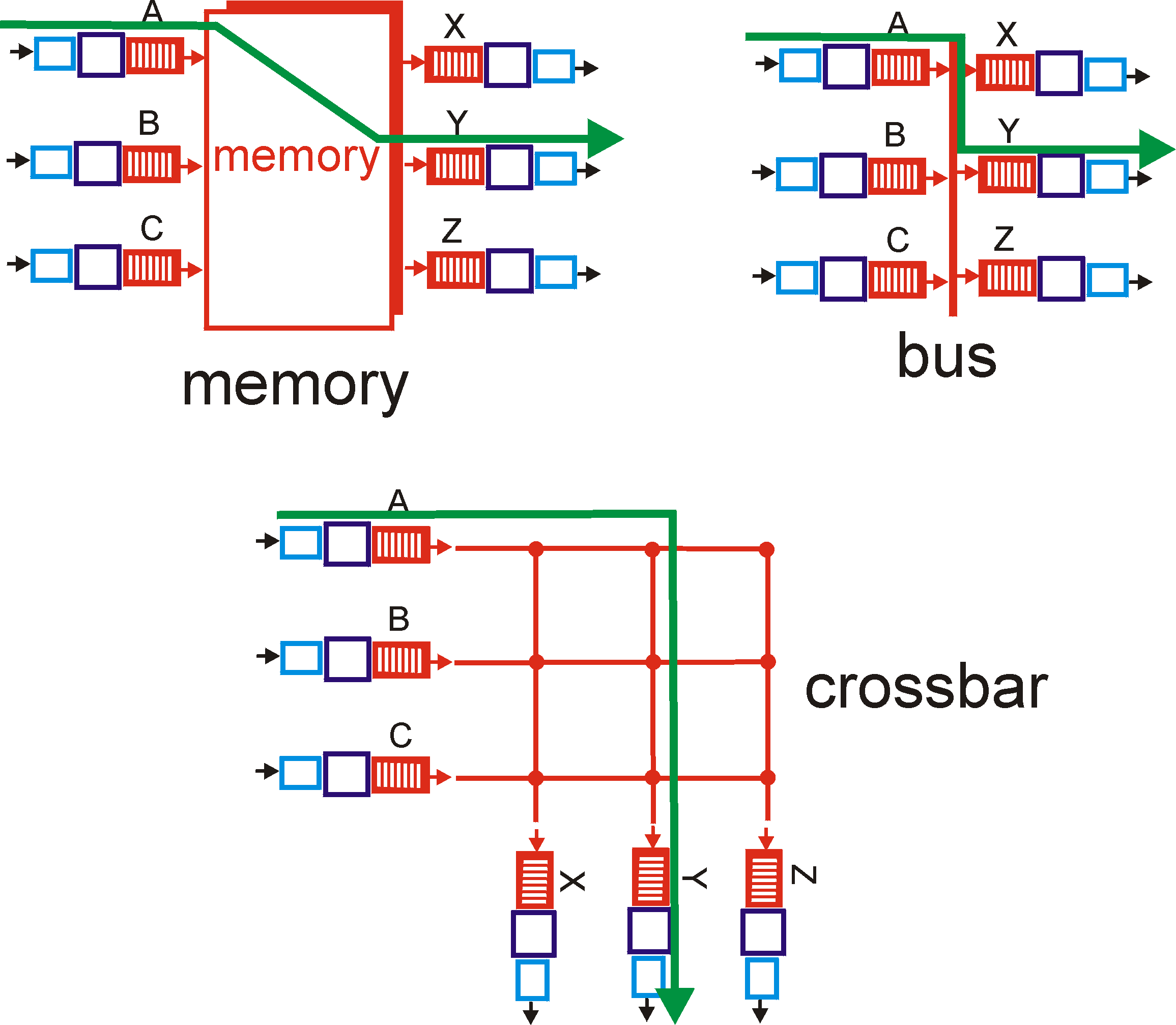
* **Routing** means Determines route from source to destination
* **Forwarding** means Movers packets from input port to proper output port
* **Router**



Run routing process to create and update forwarding table. Forward packets according to forwarding table.

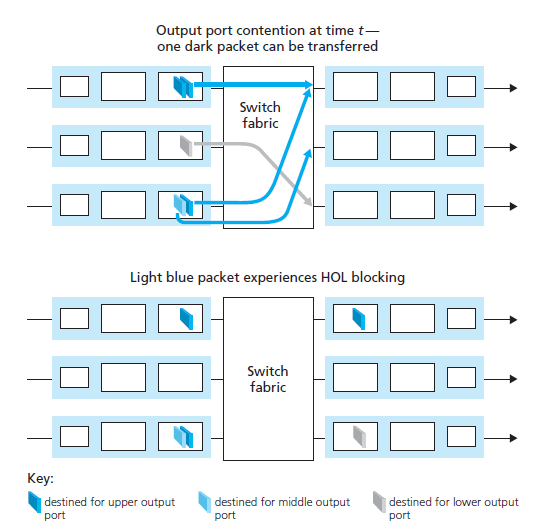


There are various witching fabrics like above.

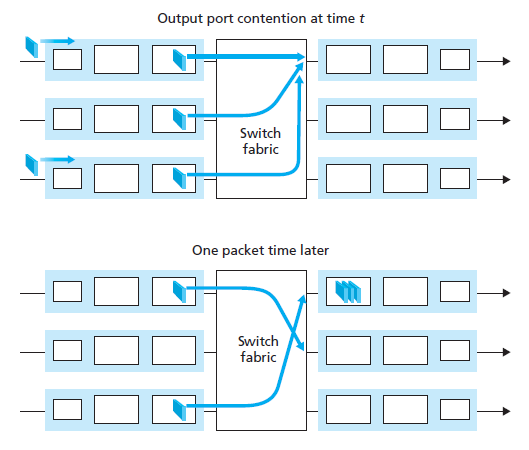
In the memory method, the output port reads packets from the shared memory.

The corssbar method is the fastest because there are several buses.

* **Input Queuing: Head-Of-Line (HOL) Blocking Problem**



* **Output queuing**



* **Switch vs. Router**

|  |  |  |
| --- | --- | --- |
|  | **Switch** | **Router** |
| **Network scale** | Small | Large |
| **Resource efficiency** | Low  (Tree-based routing) | High  (Shortest path routing) |
| **Operating layer** | Layer2 | Layer3 |

* **Routing Algorithms: Link State**

Each node should know network topology and all the link costs. Then finds shortest path to every node. e.g.) Dijkstra algorithm.

* **Routing Algorithms: Distance Vector**

Each node should know its neighbors and link costs to the neighbors. Then computes the next hop to every node exchanging the information with neighbors. e.g.) Bellman-Ford algorithm

* **Dijkstra Algorithm**

where is number of vertices.

2

2

2

5

5

1

1

1

3

3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Step | N’ | v | w | x | y | z |
| 0 | u | 2, u | 5, u | **1, u** | inf | inf |
| 1 | ux | 2, u | 4, x |  | **2, x** | imf |
| 2 | uxy | **2, u** | 3, y |  |  | 4, y |
| 3 | uxyv |  | **3, y** |  |  | 4, y |
| 4 | uxyvw |  |  |  |  | **4, y** |
| 5 | uxyvwz |  |  |  |  |  |

* **Bellman-Ford Algorithm**

where denotes cost of shortest path from to , denotes link cost from to

where , are number of vertices, and edges

2

7

1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | dist | out | y | dist | out | z | dist | out |
| x | 0 | - |  | 2 | x |  | 7 | x |
| y | 2 | y |  | 0 | - |  | 1 | - |
| z | 7 | z |  | 1 | z |  | 0 | y |

|  |  |  |  |
| --- | --- | --- | --- |
|  | x | dist | out |
|  | 0 | - |
|  | 2 | y |
|  | **3** | **y** |

* **Bellman-Ford Algorithm: Bad News Travels Slow**

4 to 60

50

1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| time | | t0 | | t1 | | t2 | | t3 | |
| from | to | dist | out | dist | out | dist | out | dist | out |
| x | x | 0 | - | 0 | - | 0 | - | 0 | - |
| y | 4 | y | 60 | y | 51 | z | 51 | z |
| z | 5 | y | 50 | z | 50 | z | 50 | z |
| y | x | 4 | x | **6** | **z** | 6 | z | **8** | **z** |
| y | 0 | - | 0 | - | 0 | - | 0 | - |
| z | 1 | z | 1 | z | 1 | z | 1 | z |
| z | x | 5 | y | 5 | y | **7** | **y** | 7 | y |
| y | 1 | y | 1 | y | 1 | y | 1 | y |
| z | 0 | - | 0 | - | 0 | - | 0 | - |

44 iterations before algorithm stabilizes

Solution is “Poisoned Reverse”. If routing path is x to y to z, z tells y its distance to x is infinite. So y won’t rout to x via z.

* **Hierarchical Routing: Terminology**

**Autonomous System (AS)**: Group of routers administrated by single organization.

**Intra-AS routing:** All the routers in a single AS should run the same routing protocol.

**Inter-AS routing**: Border routers represent the AS to outer world. All the border routers should run the same routing protocol.

* **Hierarchical Routing**

AS3

AS1

AS2

x

Suppose AS1 learns (via inter-AS protocol) that subnet x reachable via AS3 (gateway 1c) but not via AS2.

Inter-AS protocol propagates reachability info to all internal routers.

Router 1d determines from intra-AS routing info that its interface I is on the least cost path to 1c.

* **Fragmentation and Reassembly**

Link has Maximum Transfer Unit (MTU). If IP packet is larger than MTU, it should be fragmented. Fragmented IP packet should be reassembled at destination.

If exists fragmented IP packet, fragflag set 1 and offset calculated by 8-byte boundaries.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | length | ID | fragflag | offset |  |
|  | 4000 | x | 0 | 0 |  |

fragmented to (20-byte for header)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | length | ID | fragflag | offset |  |
|  | 1500 | x | 1 | 0 |  |
|  | length | ID | fragflag | offset |  |
|  | 1500 | x | 1 | 185 |  |
|  | length | ID | fragflag | offset |  |
|  | 1040 | x | 0 | 370 |  |

* **IP Address**

I get IP address from Network Administrator.  
Network Administrator get IP address from ISP.  
ISP get IP address from ICANN

* **IP Classic Allocation**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class | First Octet Range | Max Hosts | Fromat (Octets) | | | | | | | |
| 4 | | | | | 3 | 2 | 1 |
| A | 1-126 | 2^24-2 | 0 |  | | | | Host ID | | |
| B | 128-191 | 2^16-2 | 10 | |  | | | |  | |
| C | 192-223 | 2^8-2 | 110 | | | NETID | | | |  |
| D | 224-239 |  | 1110 | | | | Multicast Address | | | |
| E | 240-255 |  | 1111 | | | | Experimental | | | |

-2 of ‘Max Hosts’ for  
0x00…00 (Network Identifier)  
0xFF…FF (Broadcast Address)

E.g.) YU has IP address block of 165.229.0.0 ~ 165.229.255.255

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Class B | **10**100101 | 11100101 | xxxxxxxx | xxxxxxxx |
| Gateway | 165 | 229 |  |  |

NETID is 10’0101’1110’0101

* **Classless Inter Domain Routing / Variable Length Subnet Masking (CIDR/VLSM)**

Network id can be arbitrary length. Instead the length of network prefix should be specified.

|  |  |  |  |
| --- | --- | --- | --- |
| Before | Network Prefix |  | output |
|  | 165.229.0.0 |  | port4 |
| After | Network Prefix | Subnet Mask | output |
|  | 165.229.0.0 | 255.255.0.0 | port4 |

* **Reserved IP Addresses**

IP address with all 0’s in host id part for network prefix.  
IP address with all 1’s in host id part for broadcast address.

Private IP addresses: Not used in the public Internet

|  |  |
| --- | --- |
| Class | Private IP Address |
| 1xA | 10.0.0.0~10.255.255.255 |
| 16xB | 172.16.0.0.~172.31.255.255 |
| 256xC | 192.168.0.0 ~ 192.168.255.255 |

* **Internet Control Message Protocol (ICMP)**

Used by host and router to communicate network layer information.

* **ICMP and ping**

1. Source sends ICMP “echo request” (type 8, code 0) to destination.
2. When the echo request arrives to the destination.
3. Router sends to source an ICMP “echo reply” (type 0, code 0).

* **ICMP and tracert**

1. Source sends series of UDP segments to destination.  
   First has Time to Live (TTL) = 1,  
   Second has TTL = 2, etc.  
   Unlikely port number.
2. When nth datagram arrives to nth router.  
   Router discards datagram.  
   And sends to source an ICMP message “TTL expired” (type 11, code 0).  
   This includes name of router & IP address.
3. When ICMP message arrives, source calculates Round Trip Time (RTT).
4. UDP segment eventually arrives at destination host.  
   Destination returns ICMP “host unreachable” packet (type 3, code 3)

* **Address Resolution Protocol (ARP)**

How to determine MAC address of B knowing B’s IP address?

1. A wants to send datagram to B, and B’s MAC address not in A’s ARP table.
2. A broadcasts ARP query packet, containing B’s IP address like below.  
   [dst MAC address = 0xFFFF’FFFF]  
   Then all machines on LAN receive ARP query.
3. B receives ARP packet, replies to A with its (B’s) MAC address like below  
   [src MAC address = (B’s MAC address)]  
   [dst MAC address = (A’s MAC address)]  
   Frame sent to A’s MAC address (unicast)
4. A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (time out).  
   Soft State: Information that times out (goes away) unless refreshed.

* **Multiplexing**

Packets can be exchanged between hosts. But, usually many processes on single host.

Each process is assigned unique transport ID in a host. Actually, each socket is assigned an ID. Process can have multiple sockets.

* **Connectionless Multiplexing**

Anyone can send packets to port #X at any time.

* **Connection-oriented Multiplexing**

Only connected sockets can send packets to port #X. There are multiple sockets with port #X not shown outside and distinguished internally by peer.

* **Pipelines Protocols**

Sender allows multiple, “in-flight”, yet-to-be-acknowledged packets. Two generic forms of pipelined protocols: Go-Back-N, Selective Repeat.

* **Go-Back-N**

Tx window: N, Rx window: 1

[Normal operation]

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0** 1 2 3 4 …

0 **1** 2 3 4 …

0 1 **2** 3 4 …

0 1 2 **3** 4 …

0 1 2 3 **4** …

0 **1 2 3 4** …

0 1 **2 3 4** …

0 1 2 **3 4** …

0 1 2 3 **4** …

pkt0

pkt1

pkt2

pkt3

ACK0

ACK1

ACK2

ACK3

pkt4

pkt5

pkt6

pkt7

0 1 2 3 4 **…**

[Corruption of packets]

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0** 1 2 3 4 …

**0** 1 2 3 4 …

**0** 1 2 3 4 …

**0** 1 23 4 …

**0** 1 2 3 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

pkt0

pkt1

pkt2

pkt3

NACK0

ACK1

ACK2

ACK3

pkt0

pkt1

pkt2

pkt3

0 **1** 2 3 4 …

Sender can transmit N packets at the same time without receiving ACK. If packet is corrupted or lost, retransmit packet stored at buffer.

Receiver should be discard out-of-order packets. If receiver stores out-of-packets, there is no available buffer for the in-order packet.

Also sender should discard the out-of-order ACKs.

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0** 1 2 3 4 …

**0** 1 2 3 4 …

**0** 1 2 3 4 …

**0** 1 23 4 …

**0** 1 2 3 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

pkt0

pkt1

pkt2

pkt3

ACK-1

ACK-1

ACK-1

ACK-1

pkt0

pkt1

pkt2

pkt3

0 **1** 2 3 4 …

[Alternative: Cumulative ACK]

NACK is not required any more. First duplicate ACK acts as a NACK.

* **Go Back to Stop-And-Wait**

We can develop Stop-And-Wait with Cumulative ACK. Since SAW is a special case of GBN with N=1

* **Selective Repeat**

Tx window: N, Rx window: N

Sender can transmit N packets at the same time without receiving ACK. If packet is corrupted or lost, retransmit packet stored at buffer.

Receiver can store out-of-order packets when there are available buffers.

[Cumulative ACK is not good here]

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

0 1 2 3 4 …

0 1 2 3 4 …

0 1 **2** 3 4 …

0 1 **2 3** 4 …

**0 1 2 3** 4 …

**0 1 2 3** 4 …

0 **1 2 3 4** …

pkt0

pkt1

pkt2

pkt3

ACK-1

ACK-1

pkt0

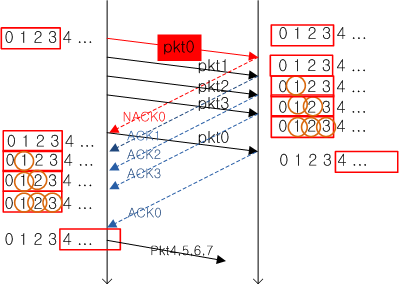
pkt1

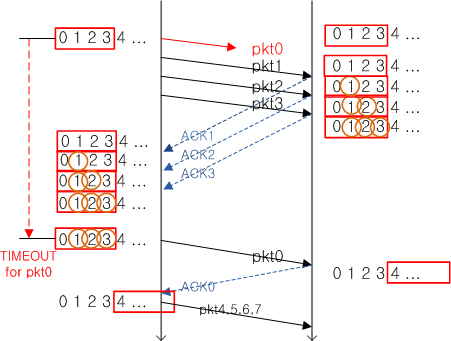
**0** 1 **2 3** 4 …

ACK0

0 1 2 3 4 …

It works, but we could send pkt1 earlier with normal ACK.

  
[For corruption]

  
[For loss]

* **Flow Control**

Sender should not flood receiver buffer.

There are various methods: Window based methods with TxWnd ≤ RxWnd / Impending overflow notification / Available buffer notification / etc.

수신자의 버퍼 상태를 고려해서 송신자가 전송속도를 조절하는 것.

* **Congestion Control**

Informally: “too many sources sending too much data too fast for network to handle”

네트워크를 위해서 송신자가 전송속도를 조절하는 것.

* + Symptoms

Lost packets (buffer overflow at routers)  
Long delays (queueing in router buffers)

* **Model for Congestion**

Mathematical mode: Single queue with finite buffer.



Where .

is new (original) data rate. (전송하려는 양)

is retransmission data rate. (재전송 해야하는 양)

is loss probability.

* **User Datagram Protocol (UDP)**

Connectionless multiplexing / No reliable transfer / No congestion control.

It’s simple and may not need reliable transfer or congestion control like multimedia application.

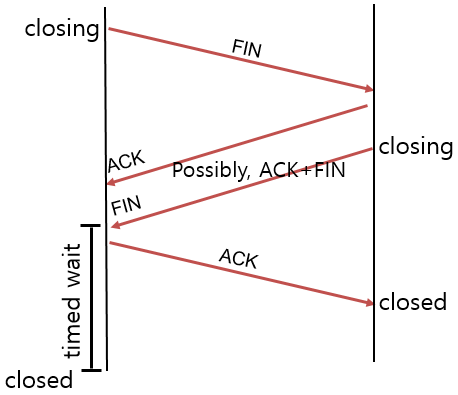
* **Transmission Control Protocol (TCP)**

Connection-oriented multiplexing / Reliable transfer / Congestion control.

* **Connection Management: Setup**

1. Client sends SYN to server.  
   Specifies initial random seq number #m with no data.
2. Server replies with SYN/ACK  
   ACK for SYN (ack: m+1). Specifies server initial random seq number #n with no data.
3. Client replies with ACK  
   ACK for SYN/ACK (ack: n+1) may with data (seq: m+1).

* **Connection Management: Teardown**



1. Client sends FIN to server.
2. Server replies with ACK.  
   Close connection, and sends FIN.
3. Client replies with ACK for FIN  
   Enters “timed wait”
4. Server close connection.

* **ARQ in TCP**

Tx window: K, Rx window: M

Sender is window sliding for reception of smallest outstanding segment. Time is maintained.

Receiver stores every received packets, using cumulative ACK and ACK for every successful reception.

* **ARQ in TCP: Delayed ACK**

Nagle’s algorithm: Reduce TCP segments by aggregating small data.

Reduce ACKs by ACK aggregation. It’s mean that under some condition, ACK can be delayed until arrival of next segment.

* **ARQ in TCP: Fast Retransmission**

TCP throughput drops significantly once timeout occurs. Retransmit possibly lost packet by early detection (3 duplicate ACKs).

* **Congestion Control in TCP**

Loss (timeout)

Loss (timeout)

cwnd > ssthresh

Loss (3dupACK)

New ACK

Loss (timeout)

Loss (3dupACK)

Start

* **Domain Name Service (DNS)**

Map between IP address and Domain name.

* **DNS Hierarchy**

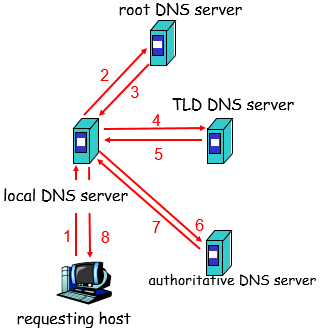
Root DNS Server gives Top Level domain DNS Server’s IP.

Top Level Domain (TLD) DNS Server gives Authoritative DNS Server’s IP.

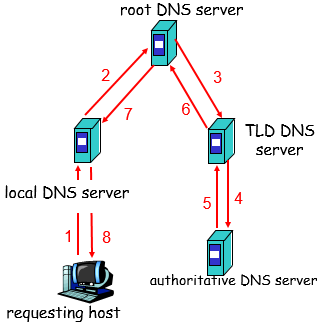
Authoritative DNS Server gives Local DNS Server’s IP.

Local DNS Server gives target host’s IP.

* **Domain Name Resolution: Iterated Query**



* **Domain Name Resolution: Recursive Query**



* **Dynamic Host Configuration Protocol (DHCP)**

Allow host to dynamically obtain IP address when it joins network.

1. Host broadcasts “DHCP discover”  
   src: 0.0.0.0:68  
   dst: 255.255.255.255:67  
   yiaddr: 0.0.0.0
2. DHCP server responds with “DHCP offer”  
   src: 165.229.1.10:67  
   dst: 255.255.255.255:68  
   yiaddr: 165.229.1.4  
   Lifetime: 3600 sec
3. Host requests IP address with “DHCP request”  
   src: 0.0.0.0:68  
   dst: 255.255.255.255:67  
   yiaddr: 165.229.1.4  
   Lifetime: 3600 sec
4. DCHP server sends address with “DHCP ACK”  
   src: 165.229.1.10:67  
   dst: 255.255.255.255:68  
   yiaddr: 165.229.1.4  
   Lifetime: 3600 sec

* **DHCP Relay**

DHCP server may not be on the same subnet with client. For this, run DHCP relay at each subnet. DHCP relay forwards DHCP messages from Client to DHCP server.

* **Hyper Text Transfer Protocol (HTTP)**

Web page consists of objects and base HTML-file which includes several referenced objects. Each object is addressable by a URL.

|  |  |
| --- | --- |
| www.wikipedia.org/static/images/logo.png | |
| host name | path name |

Client initiates TCP connection to server, port usually 80. Server accepts TCP connection from client. HTTP messages exchanged between browser and Web server

HTTP is stateless. Server maintains no information about past client requests.

* **HTTP Request Message**

|  |
| --- |
| GET /somedir/page.html HTTP/1.1  Hist: www.someurl.com  User-agent: Mozilla/4.0  Connection: close  Accept-language:fr |

* **HTTP Request Methods**

GET for downloading files from Server.  
POST for input from Client to Server.  
PUT for uploading files to Server.  
DLELTE for deleting files in Server.  
…

* **HTTP Response Message**

|  |
| --- |
| HTTP/1.1 200 OK  Connection close  Date: Thu, 06 Aug 1998 12:00:15 GMT  Server: Apache/1.3.0 (Unix)  Last-Modified: Mon, 22 Jun 1998 ...  Content-Length: 6821  Content-Type: text/html  **datas...** |

* **File Distribution: Client-Server**

Server sequentially sends N copies: time.

Client i takes time to download.

Time to distribute file F to N clients.

Where

is server upload bandwidth.  
 is peer i upload bandwidth.  
 is peer i download bandwidth.

Increases linearly in N (for large N).

* **File Distribution: P2P**

Server must send one copy: time.

Client i takes time to download.

Client i uploads and Server uploads

and

Upload time =

* **Network Address Translation (NAT)**

Private address not used in public Internet. So multiple devices in local network use only one public address. NAT allows them to share public addresses well.

1. Host 10.0.0.1 sends datagram to 165.229.11.5:80
2. NAT router changes datagram source address from 10.0.0.1:3345 to 165.229.11.5:5001, updates table.
3. Reply arrives dest. address 165.229.11.5:5001
4. NAT router changes datagram destination address from 165.229.11.5:6001 to 10.0.0.1:3345

* **NAT Traversal Problem**

Client wants to connect to server which address is 10.0.0.1:50000 on the network using NAT. But client cannot use 10.0.0.1 to destination address. Since only one externally visible address such as public IP of the NAT network that contains the server

* + Manually configure NAT table

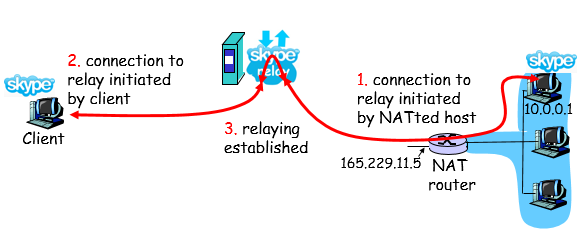
Modify table 165.229.11.5:x passes to 10.0.0.1:50000 manually.

* + Automatically configure NAT table

By Internet Gateway Device (IGD) Protocol. Host learns NAT IP address. Add/Remove NAT table dynamically

* + Relaying

Clients establish connections to relay. Relay bridges packets between connections.



1. Connection to relay initiated by NATted host.
2. Connection to relay initiated by client.
3. Relaying established.