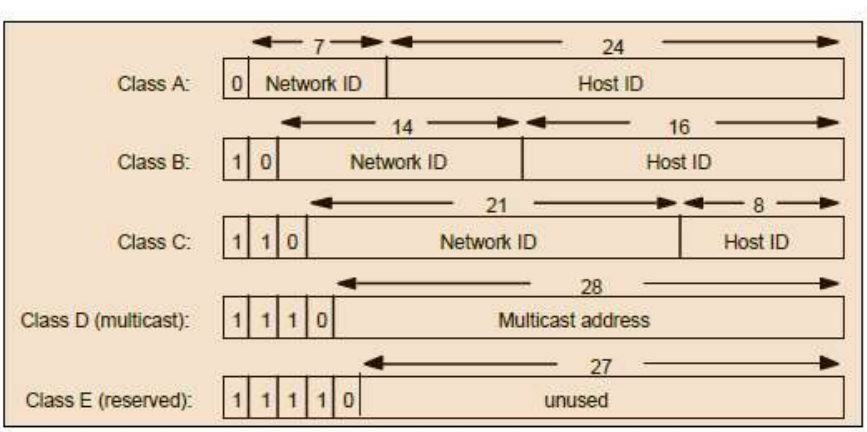
Internet Protocol (IP)

- provides a **datagram** service
  - packets are handled and delivered independently
- **best-effort** protocol
  - may loose, reorder or duplicate packets

- each packet must contain an **IP address** of its destination

Addresses - IPv4

- The **32** bits of an IPv4 address are broken into **4 octets**, or 8 bit fields (0-255 value in decimal notation).
- For networks of different size,
  - the first one (for large networks) to three (for small networks) octets can be used to identify the **network**, while
  - the rest of the octets can be used to identify the **node** on the network.



Distance-Vector Routing Algorithm

- Distributed Bellman-Ford
- Each node constructs a one-dimensional array (a vector) containing the "distance"(costs) to all other nodes and next hop id
- Routers exchange their routing tables with immediate neighbors
- Information includes the distance and next hop id
- Typical exchange periods (30s-several minutes)

9.1 Distance-Vector Routing-Update Algorithm

Distance-vector is the simplest routing-update algorithm, used by the Routing Information Protocol, or RIP. Version 2 of the protocol is specified in RFC 2453.

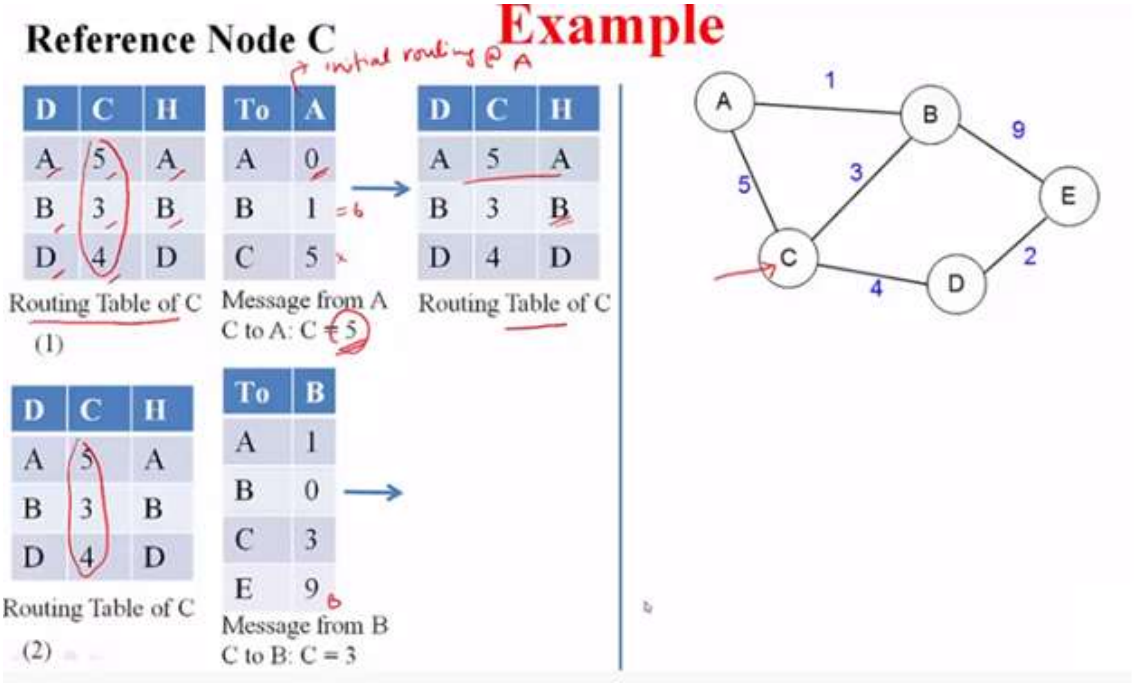
Routers identify their router neighbors (through some sort of neighbor-discovery mechanism), and add a third column to their forwarding tables representing the total **cost** for delivery to the corresponding destination. These costs are the "distance" of the algorithm name. Forwarding-table entries are now of the form ⟨destination,next_hop,cost⟩.

9.1.1 Distance-Vector Update Rules

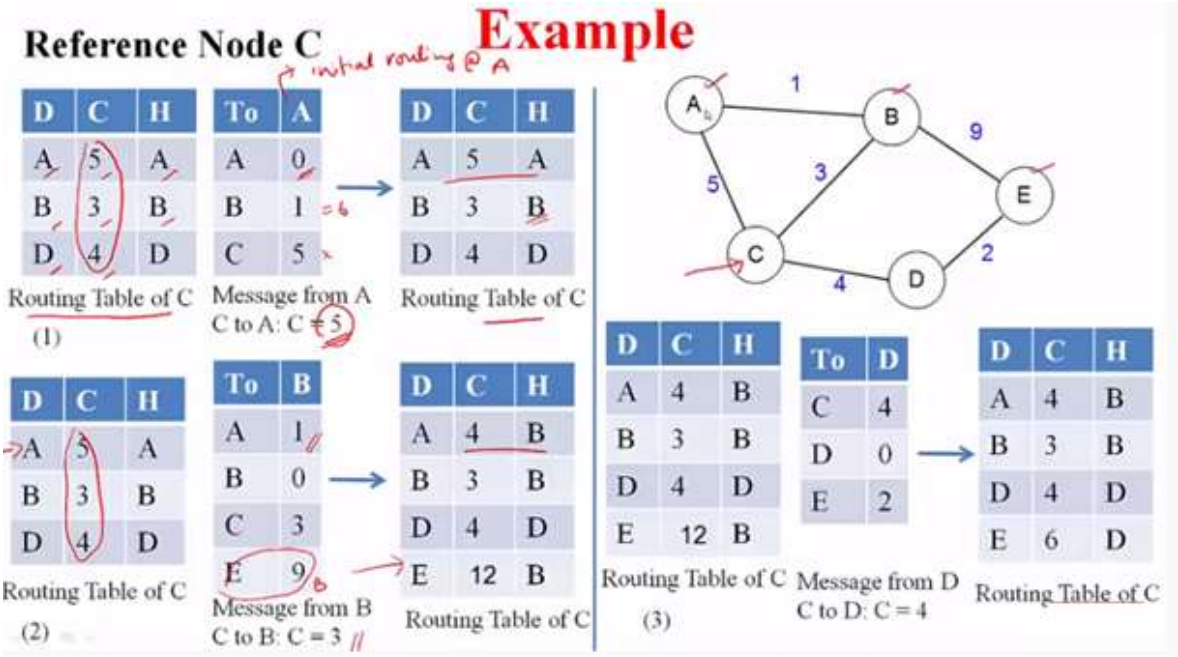
Let A be a router receiving a report ⟨D,c_D⟩ from neighbor N at cost c_N. Note that this means A can reach D *via N* with cost $c = c_D + c_N$. A updates its own table according to the following three rules:

1. **New destination**: D is a previously unknown destination. A adds ⟨D,N,c⟩ to its forwarding table.
2. **Lower cost**: D is a known destination with entry ⟨D,M,c_{old}⟩, but the new total cost c is less than c_{old}. A switches to the cheaper route, updating its entry for D to ⟨D,N,c⟩. It is possible that M=N, meaning that N is now reporting a cost decrease to D. (If $c = c_{old}$, A ignores the new report; see exercise 5.5.)
3. **Next_hop increase**: A has an existing entry ⟨D,N,c_{old}⟩, and the new total cost c is *greater* than c_{old}. Because this is a cost increase from the neighbor N that A is currently using to reach D, A must incorporate the increase in its table. A updates its entry for D to ⟨D,N,c⟩.

Reference Node C — Example



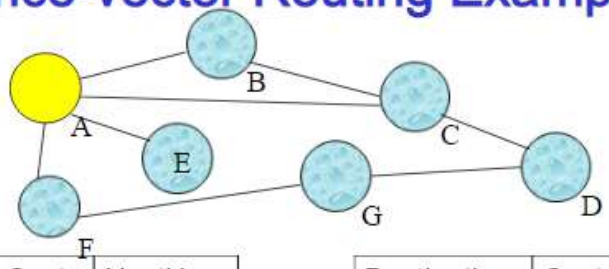
Reference Node C — Example



Points to Note

- No topology change, convergence in a few rounds
  - After one message exchange, each node knows about nodes two hops away
  - After two message exchanges, each node knows about nodes three hops away
  - And so on...
- No node has global knowledge
- Fully distributed, yet maintains correct view

Distance-vector Routing Example



Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

Initial routing table at A

Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Final routing table at A

Routing Information Protocol(RIP)

- A quite popular routing protocol built on the distance-vector algorithm

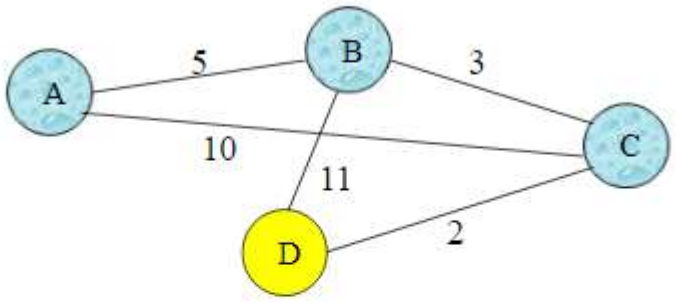
Command	Version	Zero
Family of Net 1		Address of Net 1
Address of Net 1 ... (14 Bytes)		
Distance to Net 1		
Family of Net 2		Address of Net 2
Address of Net 2 ... (14 Bytes)		
Distance to Net 2		
.....		

RIP packet format

Link-State Routing Algorithm

- Routers broadcast their neighbor connections to all routers in the network
- Information mainly connectivity and cost of the link to each neighbor
- Every node has a complete map of the network.
- Dijkstra's Shortest Path Algorithm can be used to select the best route to the destination.
- No slow convergence problems.
- Somewhat larger capacity requirement.

Link-State Routing Example



A	B	C	D
B/5	A/5	A/10	B/11
C/10	C/3	B/3	C/2
	D/11	D/2	

Link State Database in router D

Open Shortest Path First Protocol(OSPF)

Most widely used, based on link-state routing algorithm. Improvements added:

•Introduces hierarchy into routing by allowing networks partitioned into areas

•Authentication of routing messages

•Allows load balancing

Link State routing

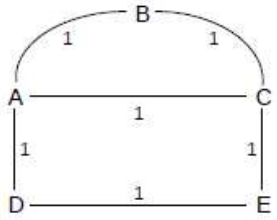
Idea

- Two Phases
- Phase 1: Reliable flooding
  - Initial State: Each node knows the cost to its neighbors
  - Final State: Each node knows the entire graph (network topology)
- Phase 2: Route calculation
  - Each node uses Dijkstra's algorithm on the graph to calculate optimal routes to all nodes

Distance vector Routing

9.1.2 Example 1

For our first example, no links will break and thus only the first two rules above will be used. We will start out with the network below with empty forwarding tables; all link costs are 1.



After initial neighbor discovery, here are the forwarding tables. Each node has entries only for its directly connected neighbors:

A: ⟨B,B,1⟩ ⟨C,C,1⟩ ⟨D,D,1⟩
B: ⟨A,A,1⟩ ⟨C,C,1⟩
C: ⟨A,A,1⟩ ⟨B,B,1⟩ ⟨E,E,1⟩
D: ⟨A,A,1⟩ ⟨E,E,1⟩
E: ⟨C,C,1⟩ ⟨D,D,1⟩

Now let D report to A; it sends records ⟨A,1⟩ and ⟨E,1⟩. A ignores D's ⟨A,1⟩ record, but ⟨E,1⟩ represents a new destination; A therefore adds ⟨E,D,2⟩ to its table. Similarly, let A now report to D, sending ⟨B,1⟩ ⟨C,1⟩ ⟨D,1⟩ ⟨E,2⟩ (the last is the record we just added). D ignores A's records ⟨D,1⟩ and ⟨E,2⟩ but A's records ⟨B,1⟩ and ⟨C,1⟩ cause D to create entries ⟨B,A,2⟩ and ⟨C,A,2⟩. A and D's tables are now, in fact, complete.

Now suppose C reports to B; this gives B an entry ⟨E,C,2⟩. If C also reports to E, then E's table will have ⟨A,C,2⟩ and ⟨B,C,2⟩. The tables are now:

A: ⟨B,B,1⟩ ⟨C,C,1⟩ ⟨D,D,1⟩ ⟨E,D,2⟩
B: ⟨A,A,1⟩ ⟨C,C,1⟩ ⟨E,C,2⟩
C: ⟨A,A,1⟩ ⟨B,B,1⟩ ⟨E,E,1⟩
D: ⟨A,A,1⟩ ⟨E,E,1⟩ ⟨B,A,2⟩ ⟨C,A,2⟩

E: ⟨C,C,1⟩ ⟨D,D,1⟩ ⟨A,C,2⟩ ⟨B,C,2⟩

We have two missing entries: B and C do not know how to reach D. If A reports to B and C, the tables will be complete; B and C will each reach D via A at cost 2. However, the following sequence of reports might also have occurred:

- E reports to C, causing C to add ⟨D,E,2⟩
- C reports to B, causing B to add ⟨D,C,3⟩

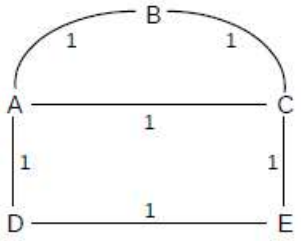
In this case we have 100% reachability but B routes to D via the longer-than-necessary path B–C–E–D. However, one more report will fix this: suppose A reports to B. B will received ⟨D,1⟩ from A, and will update its entry ⟨D,C,3⟩ to ⟨D,A,2⟩.

Note that A routes to E via D while E routes to A via C; this asymmetry was due to indeterminateness in the order of initial table exchanges.

If all link weights are 1, and if each pair of neighbors exchange tables once before any pair starts a second exchange, then the above process will discover the routes in order of length, *ie* the shortest paths will be the first to be discovered. This is not, however, a particularly important consideration.

9.1.2 Example 1

For our first example, no links will break and thus only the first two rules above will be used. We will start out with the network below with empty forwarding tables; all link costs are 1.



After initial neighbor discovery, here are the forwarding tables. Each node has entries only for its directly connected neighbors:

A: ⟨B,B,1⟩ ⟨C,C,1⟩ ⟨D,D,1⟩
B: ⟨A,A,1⟩ ⟨C,C,1⟩
C: ⟨A,A,1⟩ ⟨B,B,1⟩ ⟨E,E,1⟩
D: ⟨A,A,1⟩ ⟨E,E,1⟩
E: ⟨C,C,1⟩ ⟨D,D,1⟩

Now let D report to A; it sends records ⟨A,1⟩ and ⟨E,1⟩. A ignores D's ⟨A,1⟩ record, but ⟨E,1⟩ represents a new destination; A therefore adds ⟨E,D,**2**⟩ to its table. Similarly, let A now report to D, sending ⟨B,1⟩ ⟨C,1⟩ ⟨D,1⟩ ⟨E,2⟩ (the last is the record we just added). D ignores A's records ⟨D,1⟩ and ⟨E,2⟩ but A's records ⟨B,1⟩ and ⟨C,1⟩ cause D to create entries ⟨B,A,2⟩ and ⟨C,A,2⟩. A and D's tables are now, in fact, complete.

Now suppose C reports to B; this gives B an entry ⟨E,C,2⟩. If C also reports to E, then E's table will have ⟨A,C,2⟩ and ⟨B,C,2⟩. The tables are now:

A: ⟨B,B,1⟩ ⟨C,C,1⟩ ⟨D,D,1⟩ ⟨E,D,2⟩
B: ⟨A,A,1⟩ ⟨C,C,1⟩ ⟨E,C,2⟩

TCP vs UDP

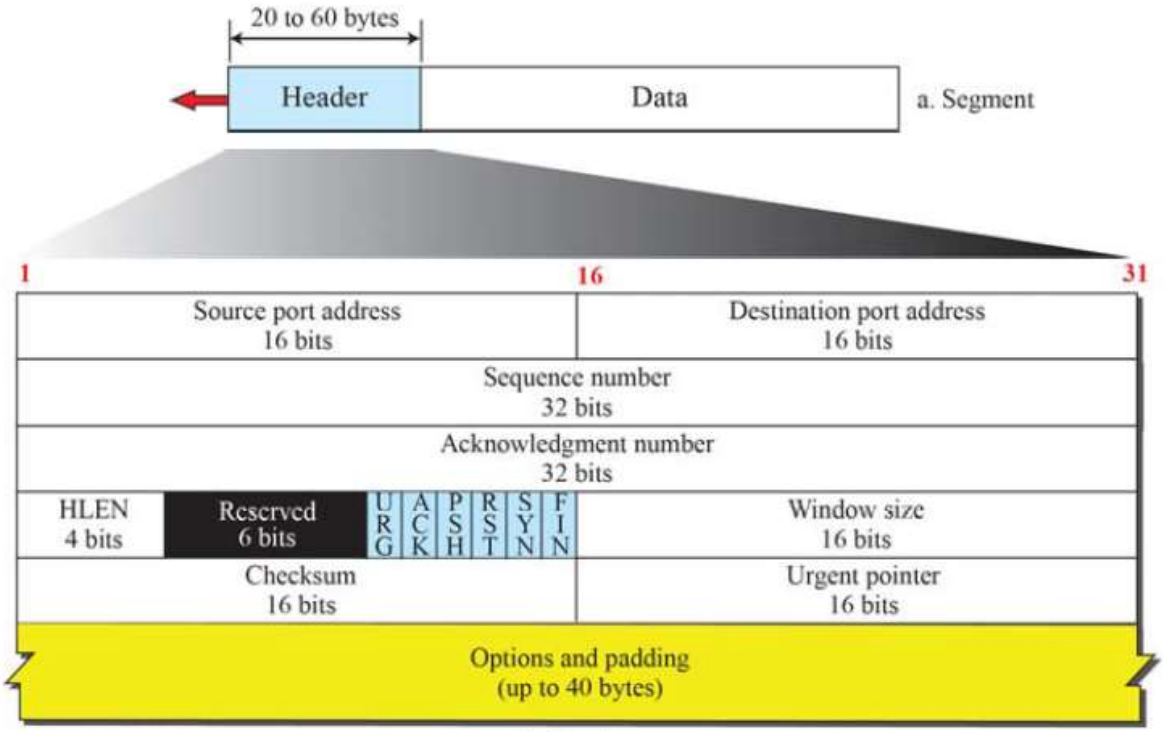
- Both use **port numbers**
  - application-specific construct serving as a communication endpoint
  - 16-bit unsigned integer, thus ranging from 0 to 65535

  ☞ to provide **end-to-end** transport
- UDP: User Datagram Protocol
  - no acknowledgements
  - no retransmissions
  - out of order, duplicates possible
  - connectionless, i.e., app indicates destination for each packet
- TCP: Transmission Control Protocol
  - reliable **byte-stream channel** (in order, all arrive, no duplicates)
    - similar to file I/O
  - flow control
  - connection-oriented
  - bidirectional

Some well-known ports used with UDP and TCP

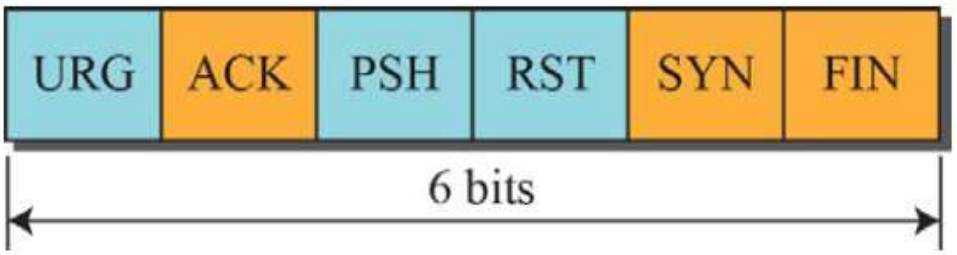
Port	*Protocol*	*UDP*	*TCP*	*Description*
7	Echo	√		Echoes back a received datagram
9	Discard	√		Discards any datagram that is received
11	Users	√	√	Active users
13	Daytime	√	√	Returns the date and the time
17	Quote	√	√	Returns a quote of the day
19	Chargen	√	√	Returns a string of characters
20, 21	FTP		√	File Transfer Protocol
23	TELNET		√	Terminal Network
25	SMTP		√	Simple Mail Transfer Protocol
53	DNS	√	√	Domain Name Service
67	DHCP	√	√	Dynamic Host Configuration Protocol
69	TFTP	√		Trivial File Transfer Protocol
80	HTTP		√	Hypertext Transfer Protocol
111	RPC	√	√	Remote Procedure Call
123	NTP	√	√	Network Time Protocol
161, 162	SNMP		√	Simple Network Management Protocol

TCP segment (packet) format



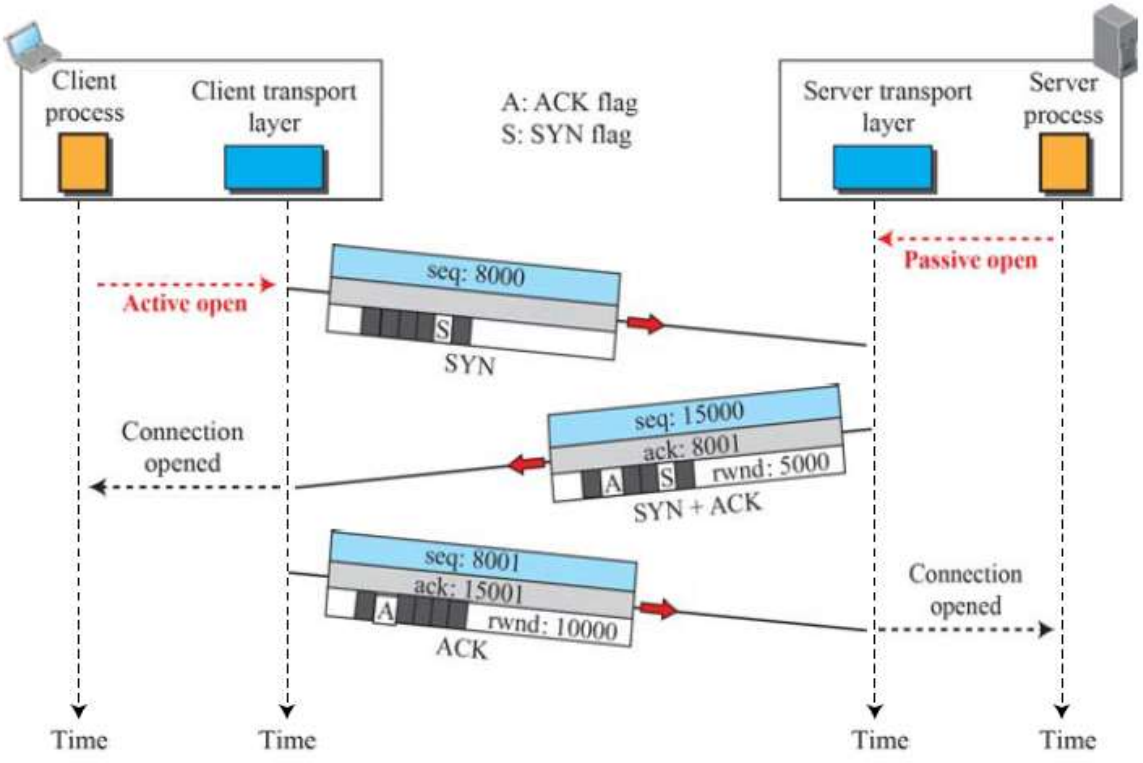
b. Header

Control field

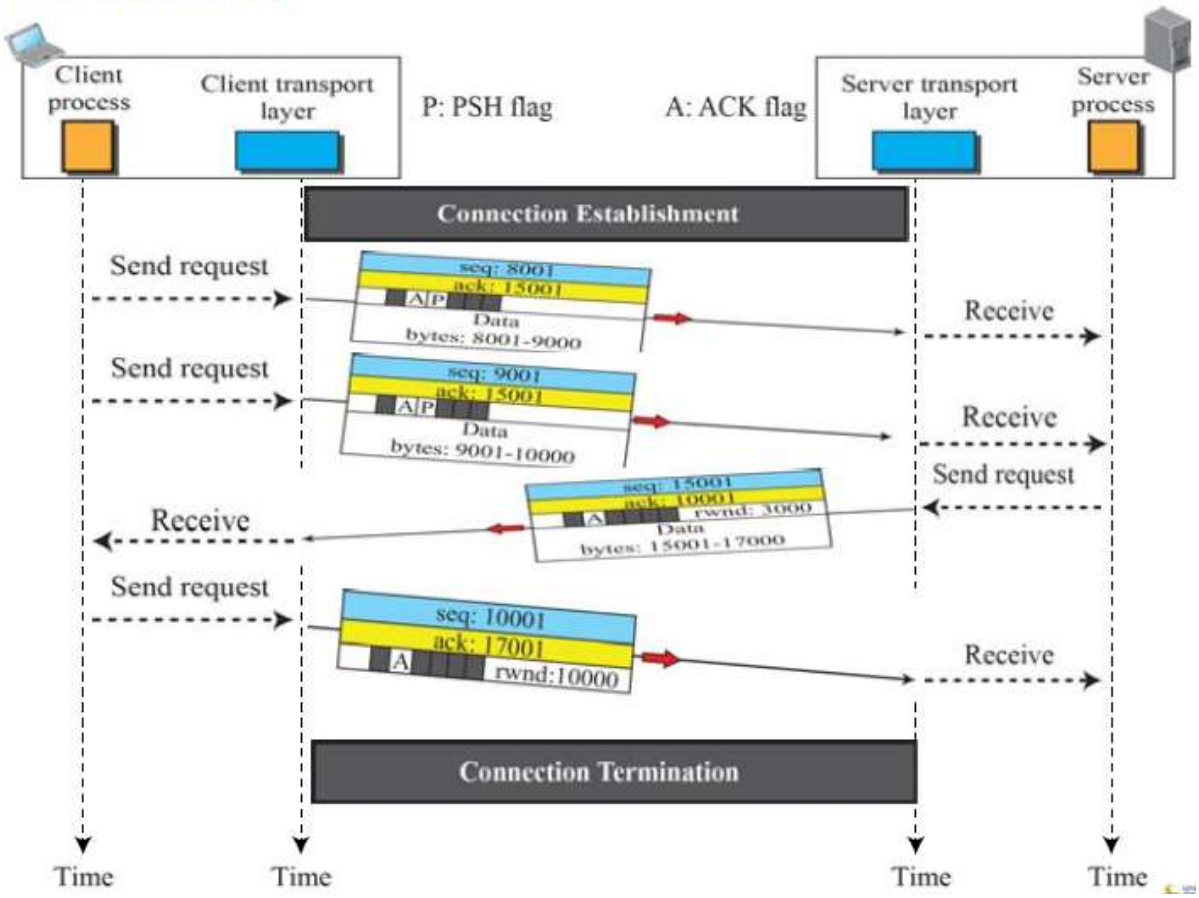


URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

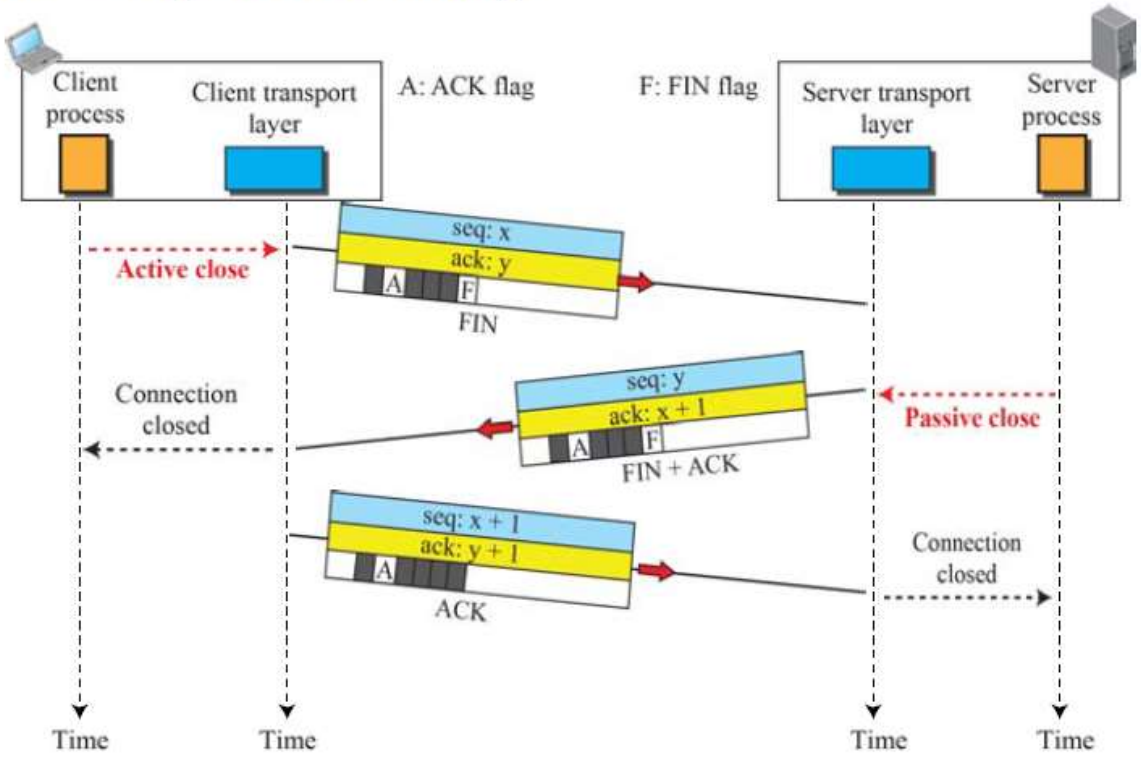
Connection establishment using three-way handshaking



Data transfer

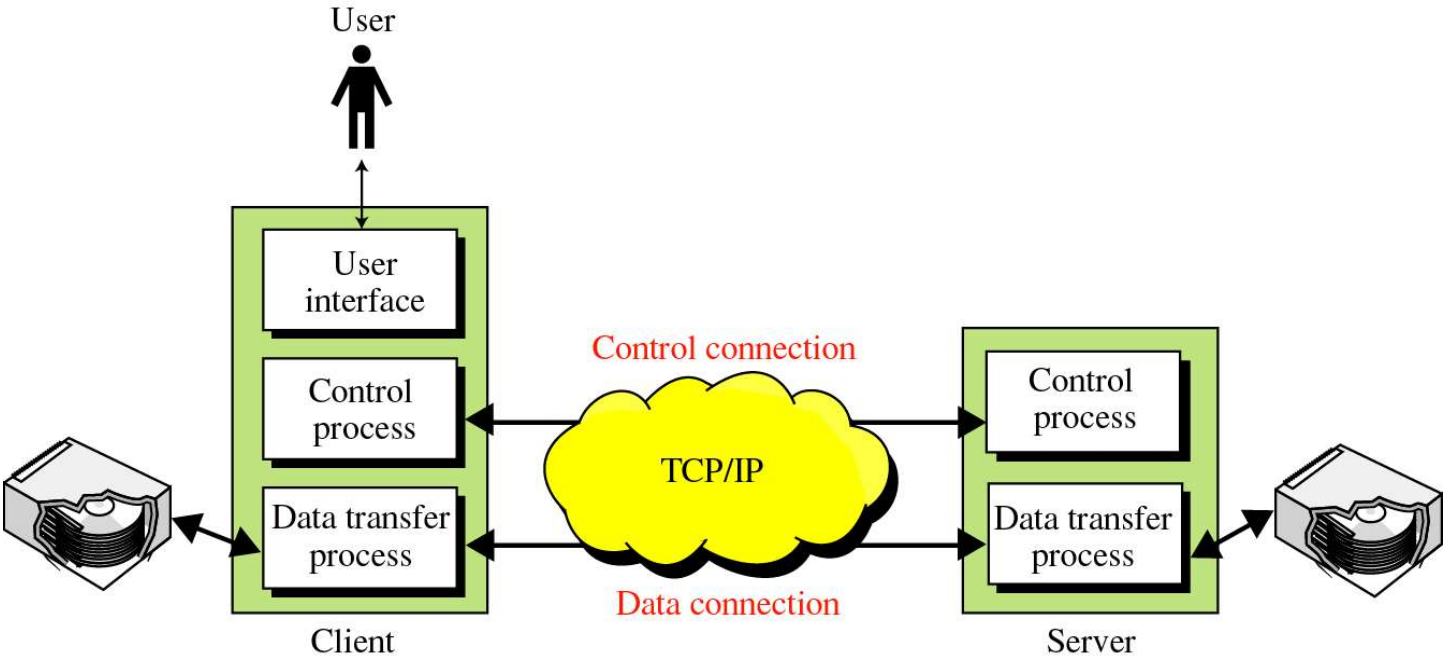


Connection termination using three-way handshaking



FTP- File Transfer Protocol

TFTP – Trivial FTP



Protocol Details

- Employs TCP and server listens on port 21
- Separate control and data channel
  - Control channel used for authorization, browsing directory listing; kept open during a session
  - Data channel supports file transfer; closed after each transfer (one file per connection)
- Through a session, FTP protocol maintains state
  - E..g for a given user, current directory as user explores directory tree

Commands

- ASCII based, sent over control channel
- USER *username*
- PASS *password*
- LIST (return list of files in current directory)
- RETR *filename* (retrieves file)
- STOR *filename* (stores file onto remote host)

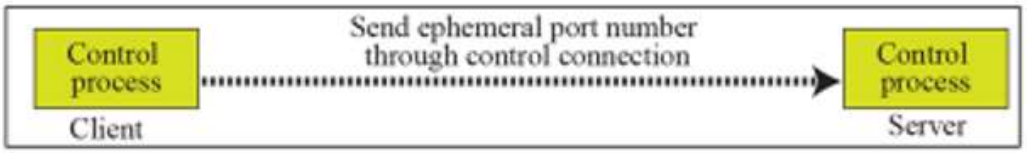
Figure 21.2 ***Opening the control connection***

a. First, passive open by server

b. Later, active open by client

Figure 21.3 *Creating the data connection*

a. First, passive open by client

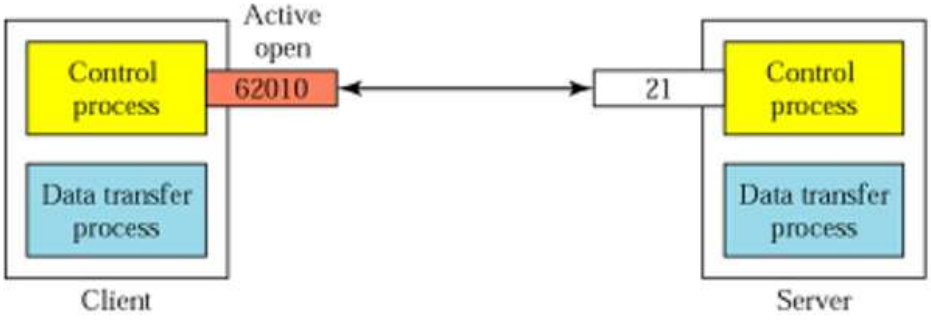


b. Second, sending of ephemeral port

c. Third, active open by server

Opening the control connection

a. Passive open by server

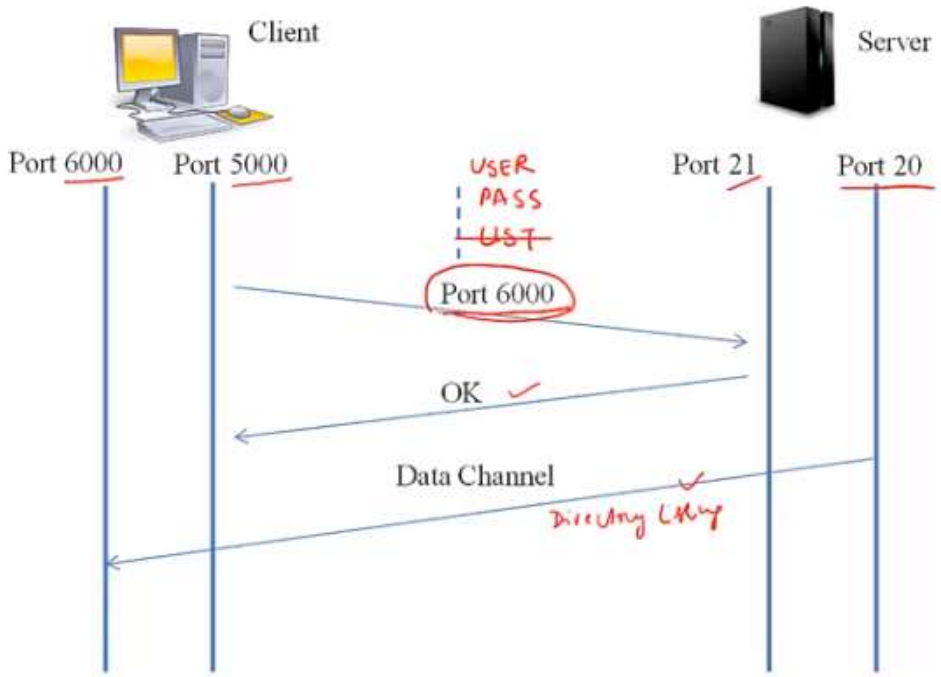


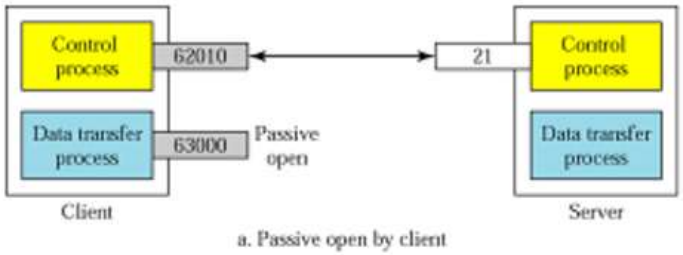
b. Active open by client

Active Mode

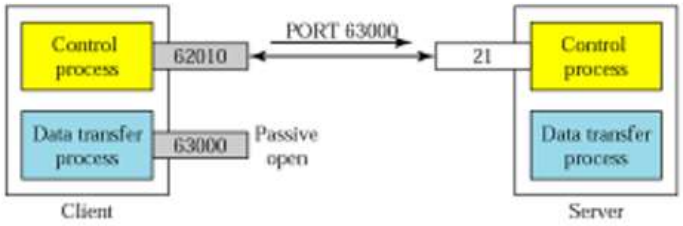
Control → TCP

Data → TCP

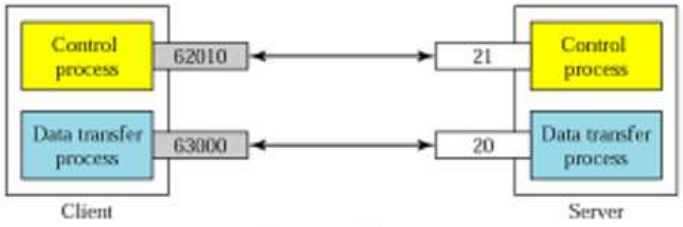




a. Passive open by client



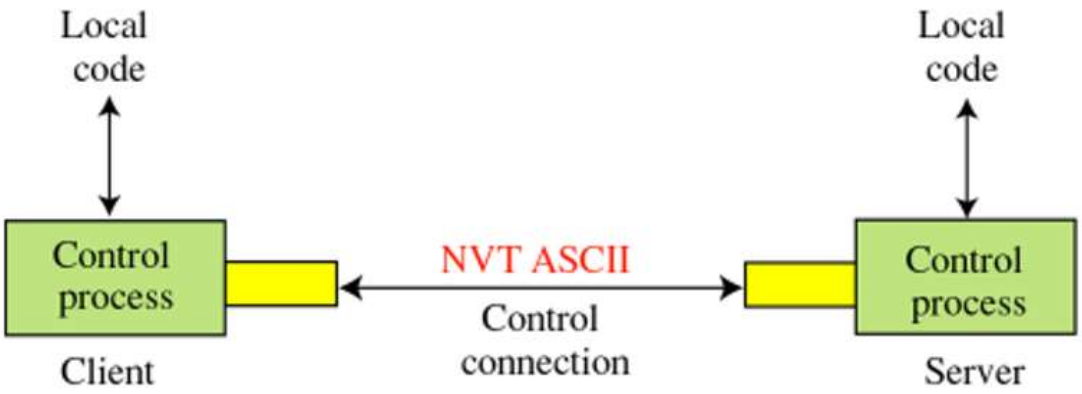
b. Sending ephemeral port number to server



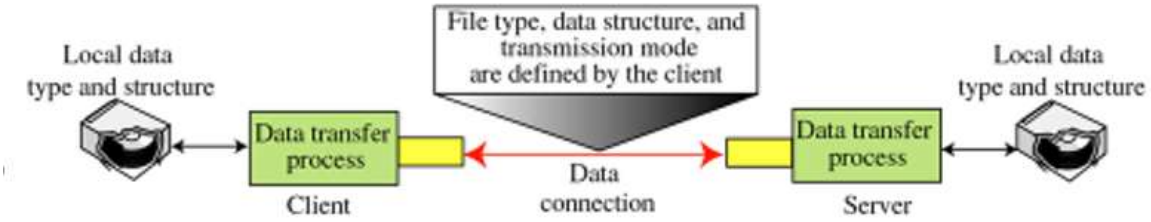
c. Active open by server

Creating the data connection

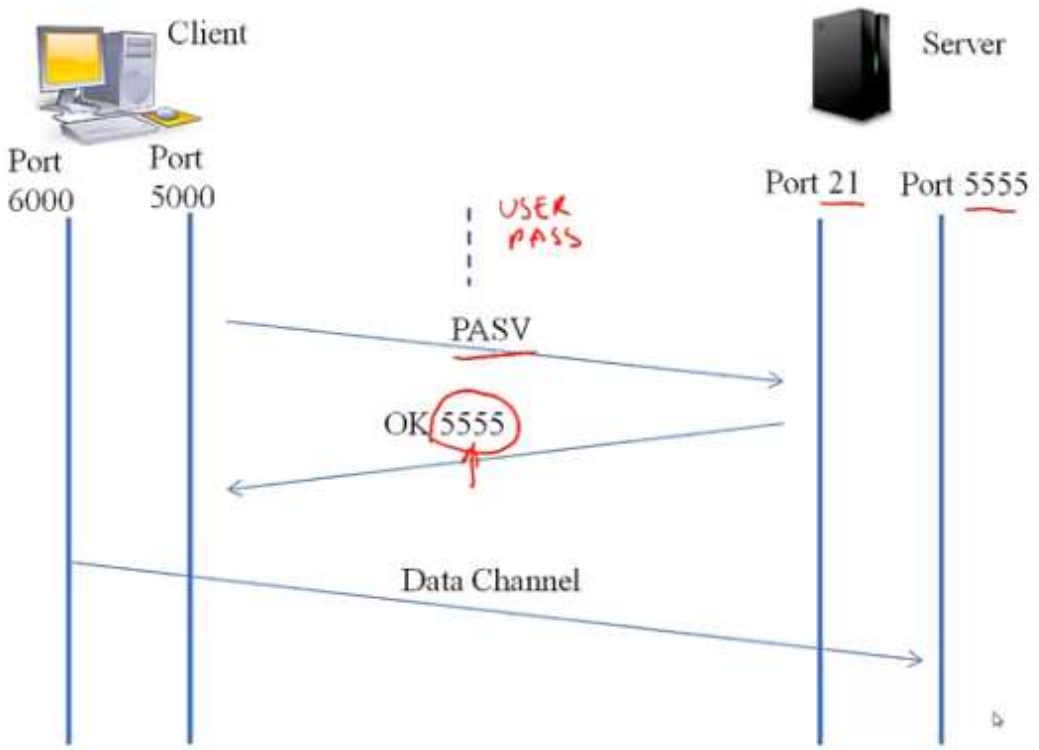
Using the control connection



Using the data connection

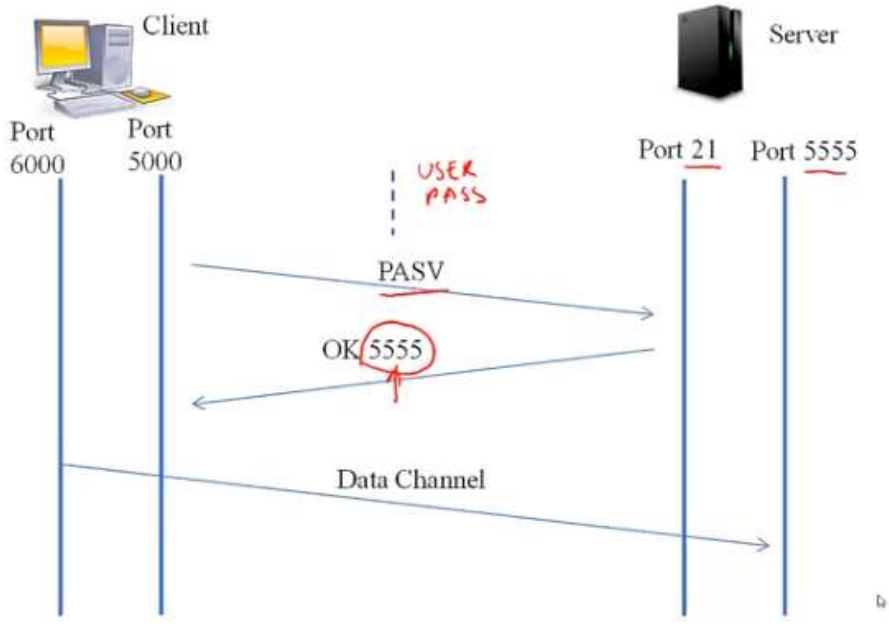


Passive Mode



Passive Mode

Useful for NAT/firewall traversal (basically where incoming connections from outside are blocked)



FTP Control Channel, TCP Port 21: All commands you send and the ftp server's responses to those commands will go over the control connection, but any data sent back (such as "ls" directory lists or actual file data in either direction) will go over the data connection.
FTP Data Channel, TCP Port 20: This port is used for all subsequent data transfers between the client and server.
In addition to these channels, there are several varieties of FTP.

Types of FTP

From a networking perspective, the two main types of FTP are active and passive. In active FTP, the FTP server initiates a data transfer connection back to the client. For passive FTP, the connection is initiated from the FTP client.
From a user management perspective there are also two types of FTP: regular FTP in which files are transferred using the username and password of a regular user FTP server, and anonymous FTP in which general access is provided to the FTP server using a well known universal login method.
Take a closer look at each type.

Active FTP

The sequence of events for active FTP is:

1. Your client connects to the FTP server by establishing an FTP control connection to port 21 of the server. Your commands such as 'ls' and 'get' are sent over this connection.
2. Whenever the client requests data over the control connection, the server initiates data transfer connections back to the client. The source port of these data transfer connections is always port 20 on the server, and the destination port is a high port (greater than 1024) on the client.
3. Thus the ls listing that you asked for comes back over the port 20 to high port connection, not the port 21 control connection.

FTP active mode therefore transfers data in a counter intuitive way to the TCP standard, as it selects port 20 as it's source port (not a random high port that's greater than 1024) and connects back to the client on a random high port that has been pre-negotiated on the port 21 control connection.

Active FTP may fail in cases where the client is protected from the Internet via many to one NAT (masquerading). This is because the firewall will not know which of the many servers behind it should receive the return connection.

Passive FTP: Passive FTP works differently:

1. Your client connects to the FTP server by establishing an FTP control connection to port 21 of the server. Your commands such as ls and get are sent over that connection.
2. Whenever the client requests data over the control connection, the client initiates the data transfer connections to the server. The source port of these data transfer connections is always a high port on the client with a destination port of a high port on the server.

Passive FTP should be viewed as the server never making an active attempt to connect to the client for FTP data transfers. Because client always initiates the required connections, passive FTP works better for clients protected by a firewall.

As Windows defaults to active FTP, and Linux defaults to passive, you'll probably have to accommodate both forms when deciding upon a security policy for your FTP server.