MODULE 02

SYLLABUS: Transport Layer: Services, Protocols, UDP, TCP. Network Layer: Introduction, Network-layer protocols, Unicast routing, Multicast routing – Multi casting Basics, Intra domain and inter-domain routing, Next generation IP, Quality of Service. [HANDS-ON SECTIONS SKIPPED – NEEDED ONLY FOR LAB]

TO WATCH VIDEO CLASSES OF THIS COURSE: TAP HERE OR SCAN QR CODE

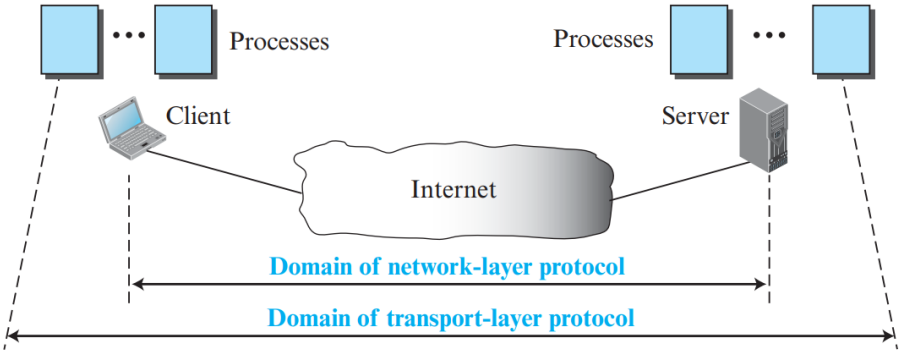
INTRODUCTION TO TRANSPORT LAYER

Transport-Layer Services

The transport layer sits between the application layer and the network layer. It gives services to the application layer and receives services from the network layer. It is the end-to-end logical vehicle for data transfer.

Process-to-Process Communication

- A **process** is a running application-layer program that uses the transport layer.
- The **network layer** gives **host-to-host** delivery. It can only deliver to the destination computer.
- The **transport layer** gives **process-to-process** delivery. It hands the message to the correct process.
- Only the two end hosts use the transport layer. Routers use only the first three layers.



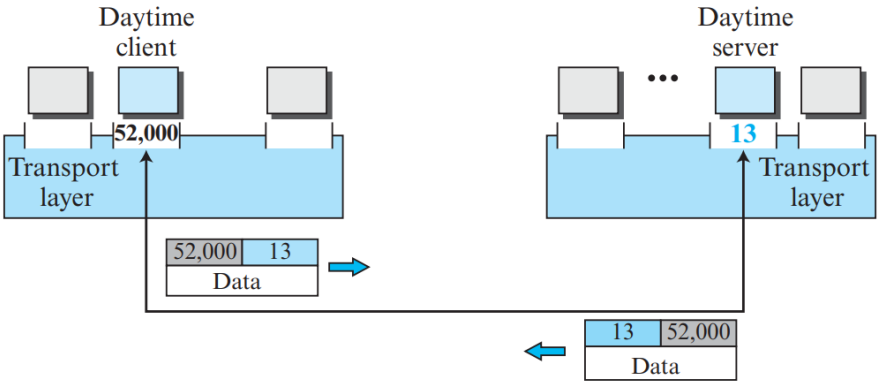
About Figure: The network layer's domain is the path across the Internet between hosts. The transport layer's domain extends further, to the processes (client and server) running on those hosts.

Addressing: Port Numbers

- Process-to-process communication mostly uses the **client-server paradigm**. Client and server processes have the same name (for example daytime client and daytime server).
- Computers run many processes at once, so four things must be identified: **local host, local process, remote host, remote process**.
- Hosts are identified by **IP addresses**. Processes are identified by **port numbers**.
- A port number is a **16-bit integer from 0 to 65,535**.

Type	Meaning
Ephemeral port number	Chosen by the client. Short-lived, because a client's life is short. Recommended above 1,023
Well-known port number	Universal port for a server, so every client knows it without asking (exceptions exist)

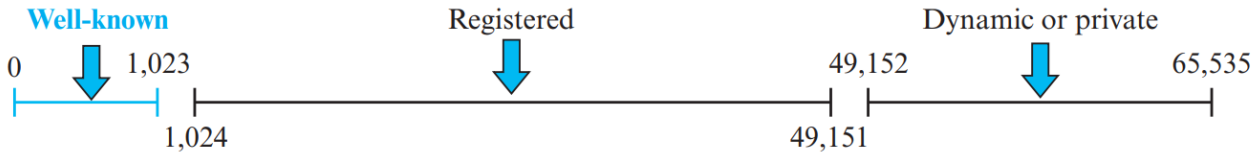
Example: a daytime client uses ephemeral port 52,000. The daytime server uses well-known port 13. The destination **IP address selects the host**. The destination **port number selects the process** on that host.



About Figure: A daytime client (port 52,000) and a daytime server (port 13) exchange data through their transport layers. Each packet carries the source and destination port numbers.

ICANN ranges

Range	Name	Control
0 to 1,023	**Well-known ports**	Assigned and controlled by ICANN
1,024 to 49,151	**Registered ports**	Not controlled by ICANN; only registered with it to prevent duplication
49,152 to 65,535	**Dynamic (private) ports**	Neither controlled nor registered; used as temporary or private ports



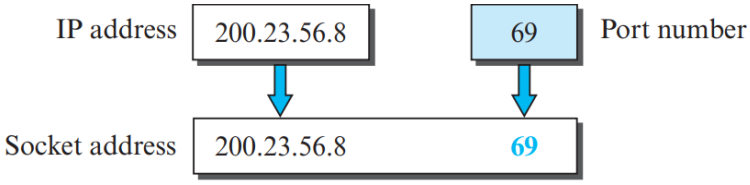
About Figure: The number line 0 to 65,535 is divided at 1,023, 1,024, 49,151 and 49,152 into the three ranges.

Some well-known ports:

Port	Protocol	Port	Protocol
13	Daytime	69	TFTP
20, 21	FTP	80	HTTP
23	TELNET	111	RPC
25	SMTP	123	NTP
53	DNS	161, 162	SNMP
67	DHCP		

Socket address = IP address + port number.

A client socket address defines the client process uniquely, and a server socket address defines the server process. The two IP addresses are in the network-layer header; the two port numbers are in the transport-layer header.



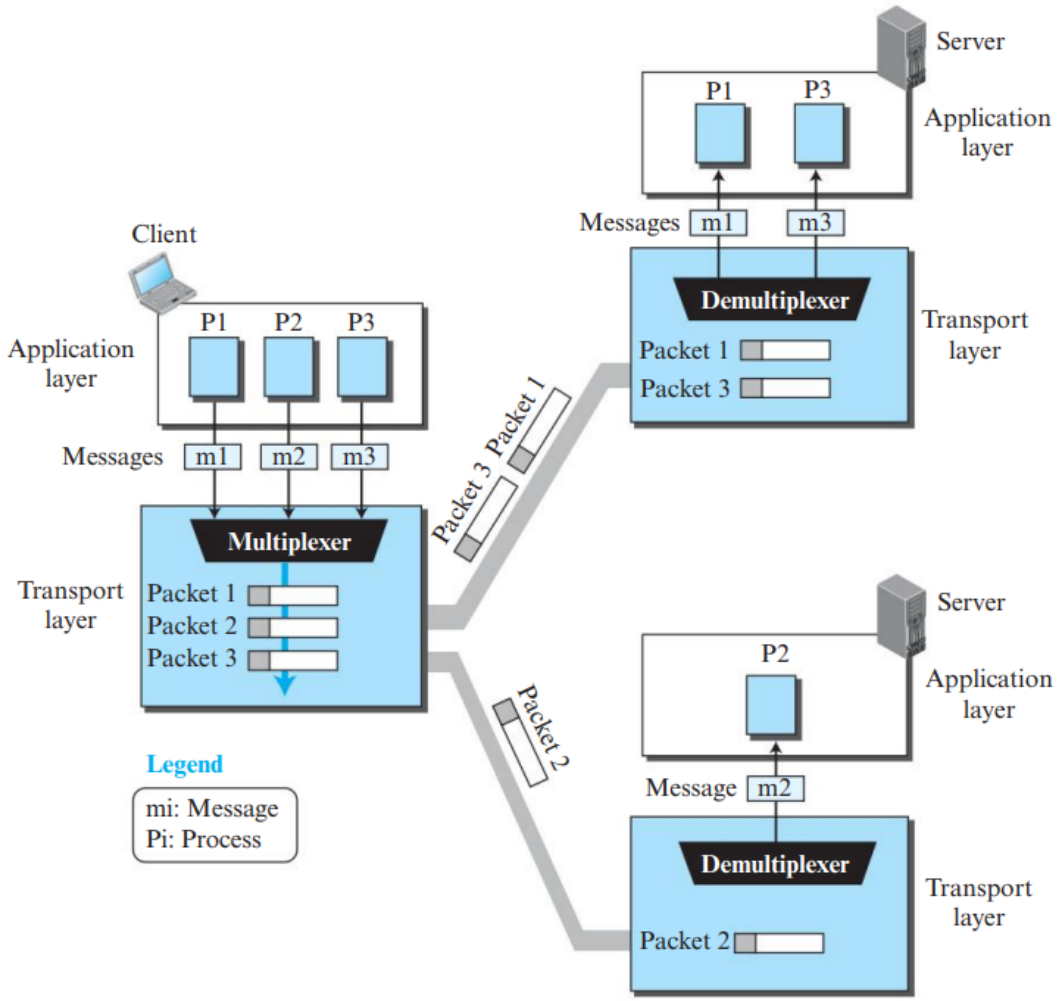
About Figure: A socket address is an IP address (for example 200.23.56.8) joined with a port number (for example 69).

Encapsulation and Decapsulation

- **Encapsulation (sender):** the process gives the message and a pair of socket addresses to the transport layer. The transport layer adds its header. The packet is called a **user datagram**, **segment** or **packet**, depending on the protocol.
- **Decapsulation (receiver):** the header is dropped and the message is delivered to the process. The sender's socket address is passed up so the process can reply.

Multiplexing and Demultiplexing

- **Multiplexing** (many to one): the transport layer at the **source** accepts messages from several processes.
- **Demultiplexing** (one to many): the transport layer at the **destination** delivers messages to several processes.



About Figure: Three client processes P1, P2 and P3 pass messages to the client transport layer (multiplexer). P1 and P3 reach one server, which demultiplexes them. P2 reaches another server.

Flow Control

Flow control balances the rate at which a producer creates data with the rate at which a consumer can use it. If items are produced faster than consumed, the consumer is overwhelmed and may discard data.

- **Pushing:** the producer sends whenever it produces, with no request. The consumer may be overwhelmed, so flow control feedback is needed.
- **Pulling:** the producer sends only after the consumer requests. No flow control is needed.

At the transport layer there are four entities: sender process, sender transport layer, receiver transport layer, receiver process.

- The sending process only produces and **pushes** message chunks.
- The sending transport layer is a consumer (of messages) and a producer (of packets, pushed to the receiver).
- The receiving transport layer is a consumer (of packets) and a producer. Delivery to the process is normally by **pulling**.
- So flow control is needed (1) from the sending transport layer to the sending process and (2) from the receiving transport layer to the sending transport layer.

Buffers

Flow control normally uses one buffer at the sender and one at the receiver. When the buffer is full, the consumer tells the producer to stop; when there is space, it tells the producer to resume.

Error Control

The network layer (IP) is unreliable, so the transport layer must provide reliability if the application needs it. Error control at the transport layer:

1. **Detects and discards** corrupted packets.
2. **Keeps track** of lost and discarded packets and **resends** them.
3. **Recognises duplicate** packets and discards them.
4. **Buffers out-of-order** packets until the missing ones arrive.

Error control involves only the sending and receiving transport layers.

Sequence numbers

- Packets are numbered so the sender knows what to resend and the receiver can spot duplicates and out-of-order packets (gaps in numbers).
- With an *m*-bit field, numbers are **modulo 2^m**.

$$\text{Range of sequence numbers: } 0 \leq \text{seqNo} \leq 2^m - 1$$

Where m = number of bits in the sequence-number field. For m = 4, numbers run 0 to 15 and then wrap around to 0.

Acknowledgment (ACK)

- The receiver sends a **positive ACK** for packets that arrive safe. Corrupted packets are discarded silently.
- The sender starts a **timer** when a packet is sent. If no ACK arrives before it expires, the packet is resent.
- Duplicates are discarded. Out-of-order packets are either discarded (treated as lost) or stored.

Combination of Flow and Error Control and the Sliding Window

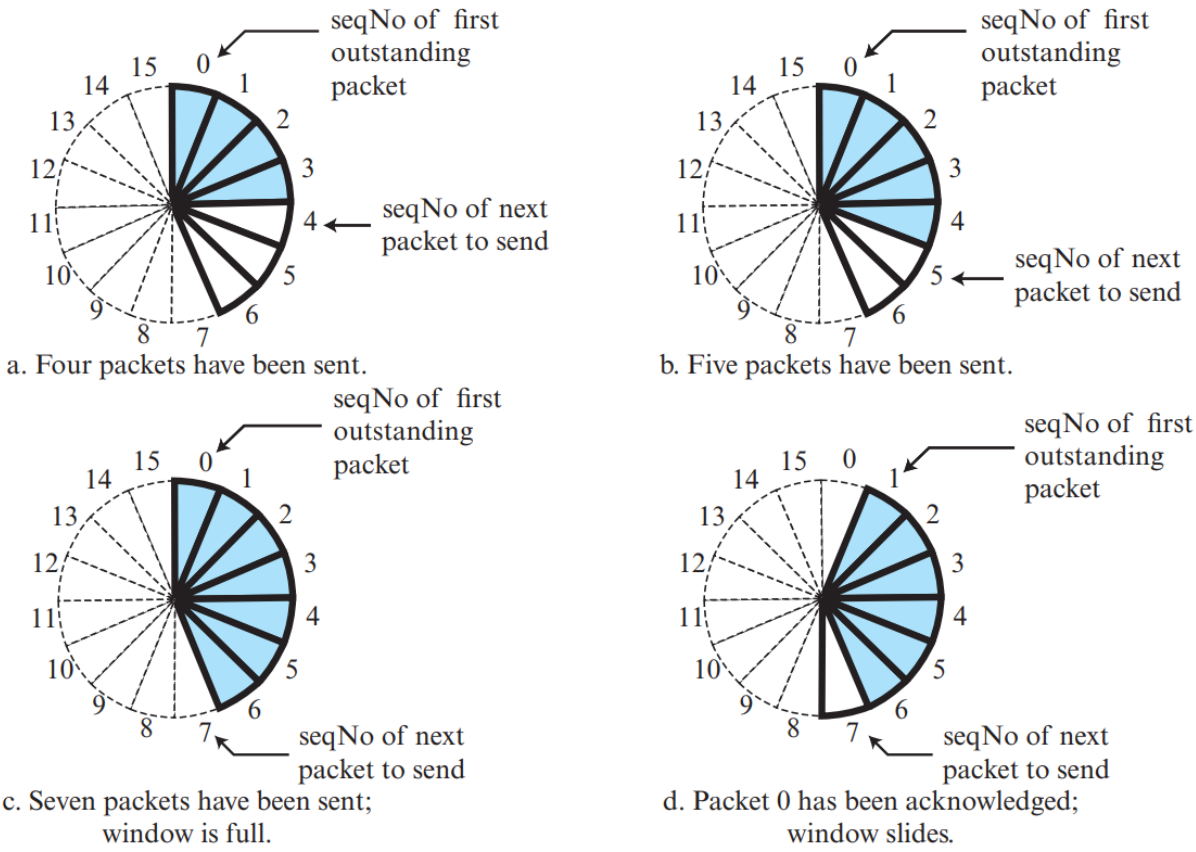
Flow control needs **buffers**. Error control needs **sequence and acknowledgment numbers**. Both are combined with **numbered buffers**.

- **Sender:** the number of the next free location x is used as the packet's sequence number. A copy is stored at x until its ACK arrives, then it is purged.
- **Receiver:** a packet with sequence number y is stored at location y until the application reads it, and an ACK is sent.

Sliding window

The numbered buffer is shown as a **window** over the circle of sequence numbers 0 to $2^m - 1$.

- At the sender, the slice of a sent packet is **marked**. When all slices are marked, the buffer is full.
- When an ACK arrives, its slice is unmarked. If consecutive slices at the beginning are unmarked, the window **slides** and frees slices at the end.
- The window is an abstraction. In reality, computer variables hold the sequence numbers of the next packet to send and the last packet sent.



About Figure: Sequence numbers 0 to 15 (m = 4) form a circle with a send window of size 7. Parts (a) to (d) show packets being sent, the window filling, and sliding when packet 0 is acknowledged.
Most protocols draw the window in **linear** form, which carries the same information.

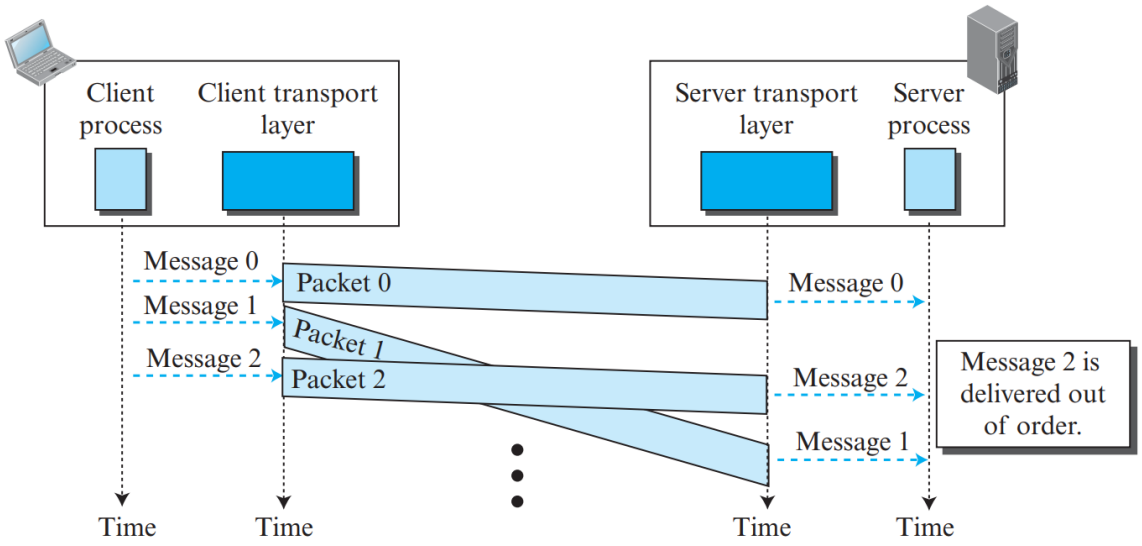
Congestion Control

- **Congestion** occurs when the load on the network (packets sent) is greater than its capacity (packets it can handle).
- **Congestion control** is the set of mechanisms that keep the load below capacity.
- It happens because routers and switches have **queues**. If packets arrive faster than they are processed, the queues overflow.
- Congestion at the transport layer is the result of congestion at the network layer. TCP implements its own congestion control (later in this module).

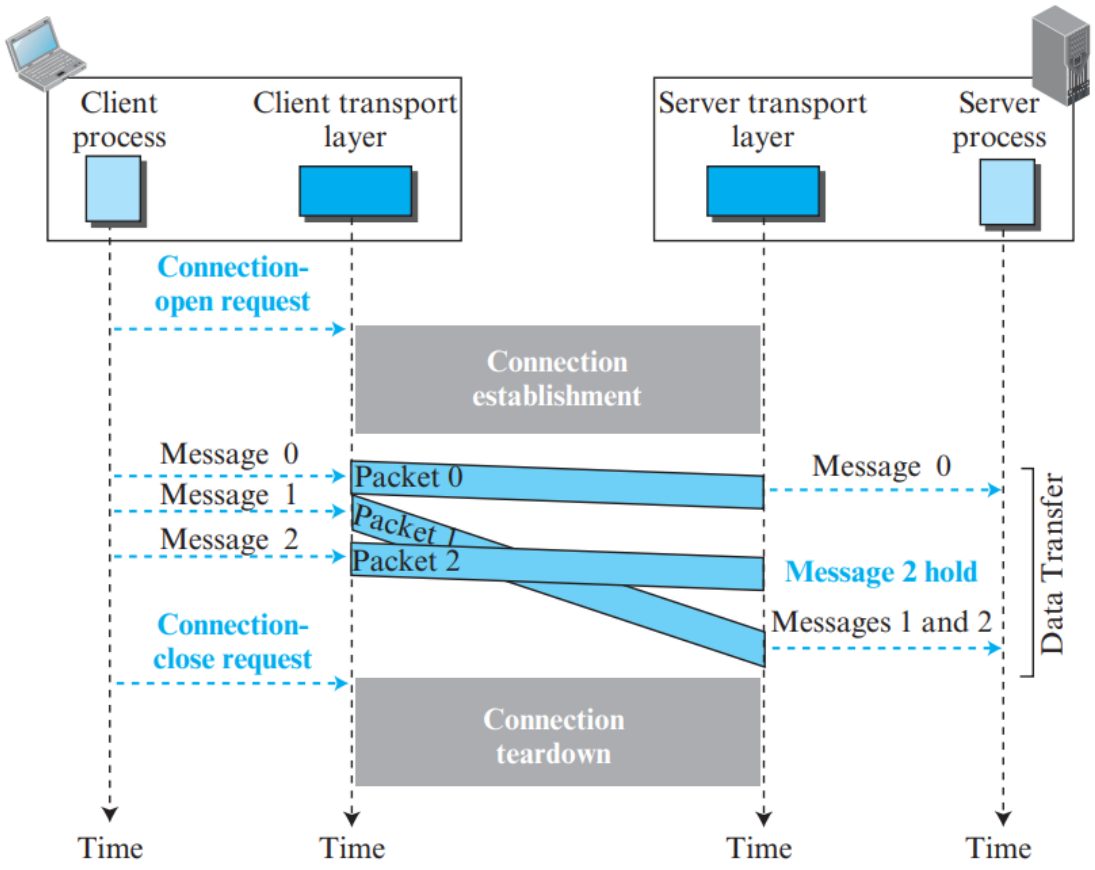
Connectionless and Connection-Oriented Services

At the transport layer, *connectionless* means **independence between packets**, and *connection-oriented* means **dependence**. (At the network layer these terms refer to the paths taken.)

Parameter	Connectionless service	Connection-oriented service
Handling of chunks	Each chunk is a separate packet, with no relation to others	Logical connection first; data exchange only after it
Phases	None	Connection establishment, data transfer, teardown
Problems	Packets may arrive **out of order**. Lost packets are not noticed, as there is no numbering	Packets are numbered, so reordering and loss can be handled
Flow, error, congestion control	**Cannot** be effectively implemented	**Can** be implemented
Involves	Only the two ends	Only the two **hosts** (end to end), not the routers



About Figure: (ABOVE: Connectionless): Three messages (0, 1, 2) sent in order. Packet 1 is delayed, so message 2 is delivered before message 1 at the server (out of order).



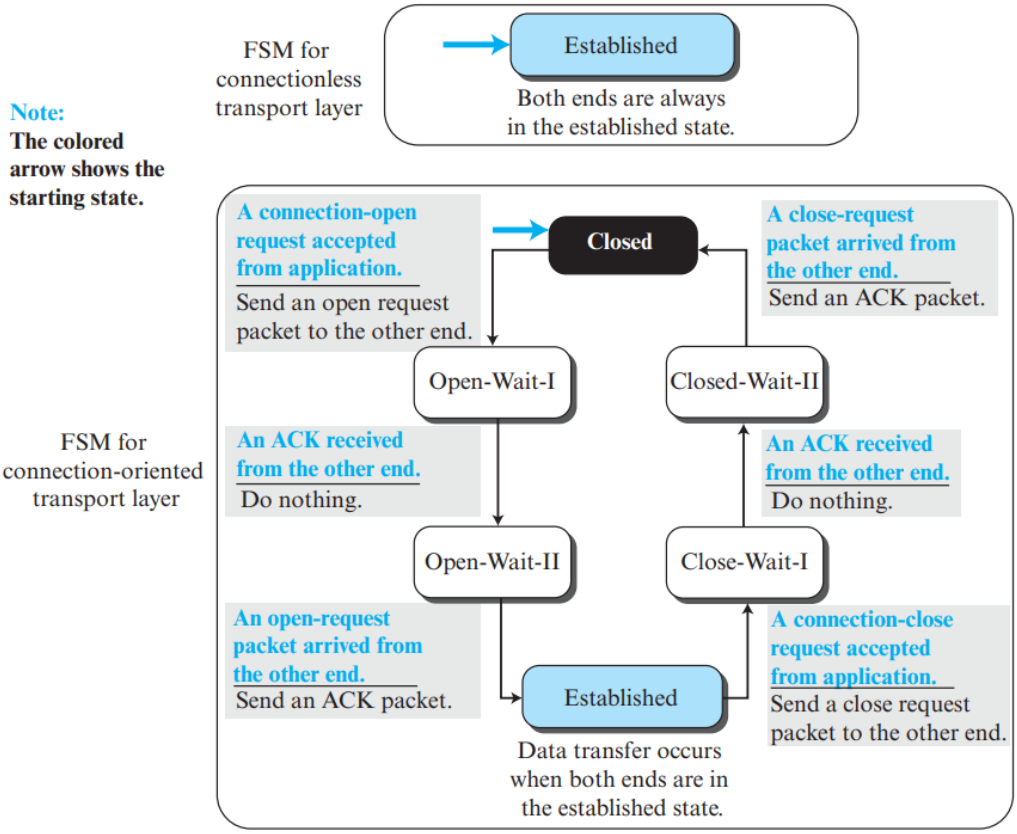
About Figure: (ABOVE: Connection-oriented service): A time line showing connection establishment, data transfer (messages 0, 1, 2), and connection teardown. A late message is held until earlier ones arrive.

Finite State Machine (FSM)

A transport layer is modelled as a machine with a finite number of **states**. It stays in one state until an **event** occurs. Each event causes **actions** and a move to the **next state** (possibly the same). One state is the initial state.

- **Connectionless FSM:** a single state, **established**.
- **Connection-oriented FSM:** the machine goes through three states to reach established and three to close.

Opening	Closing
Closed: no connection	Application sends a close request → send close-request packet → **Close-Wait-I**
Open request from the application → send open request → **Open-Wait-I**	ACK received → **Close-Wait-II**
ACK received → **Open-Wait-II** (one-way connection)	Close-request packet from the other end arrives → send ACK → **Closed**
Open request from the other end arrives → send ACK → **Established**	



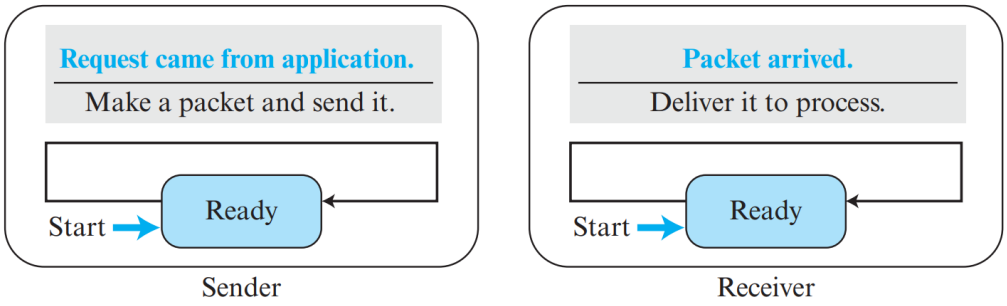
About Figure: The connectionless FSM has one state (established). The connection-oriented FSM has closed, open-wait-I, open-wait-II, established, close-wait-I and close-wait-II, with events and actions on each arrow.

Transport-Layer Protocols

A transport protocol is built by combining the services above. The general protocols below lead to the design of UDP and TCP. First they are treated as **unidirectional** (data in one direction only).

1. Simple Protocol

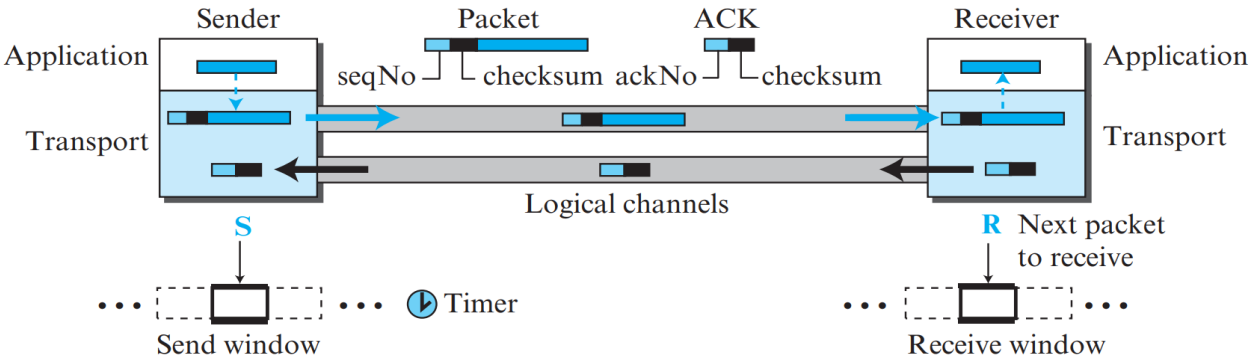
- **Connectionless**, with **no flow or error control**.
- It assumes the receiver can always handle any packet immediately.
- The sender makes a packet from each message and sends it. The receiver extracts the message and delivers it.
- Each FSM has one state, **ready**. The sender acts on a request from the application; the receiver acts on packet arrival.
- **UDP** is a slight modification of this protocol.



About Figure: The sender FSM (ready) makes and sends a packet on a request from the application. The receiver FSM (ready) delivers the message on packet arrival.

2. Stop-and-Wait Protocol

- **Connection-oriented**, with **flow and error control**.
- Both sender and receiver use a **sliding window of size 1**.
- The sender sends **one packet at a time** and waits for its ACK before the next.
- A **checksum** is added to each packet. A corrupted packet is **silently discarded**; the silence tells the sender the packet was corrupted or lost.
- The sender starts a **timer** on every packet and keeps a **copy** until it is acknowledged.
    - ACK before time-out: stop the timer, send the next packet.
    - Time-out: resend the packet.
- Only one packet and one ACK can be in the channel at a time.



About Figure: Sender and receiver transport layers with a send window (variable S, a timer) and a receive window (variable R). Packets carry seqNo and checksum; ACKs carry ackNo and checksum.

Sequence numbers

- Only two numbers are needed: **0 and 1** (arithmetic **modulo 2**).
- Reason: if the packet numbered x is sent, three things can happen.
    1. Packet and ACK arrive safely, so the next packet is x + 1.
    2. The packet is lost or corrupted. The sender resends x after time-out.
    3. The packet arrives, but the **ACK is lost**. The sender resends x (a duplicate). The receiver expects x + 1, so it knows x is a duplicate.
- The receiver must tell cases 1 and 3 apart, so x and x + 1 are enough. No x + 2 is needed.

Acknowledgment numbers

- The ackNo always announces the sequence number of the **next packet expected**.
- Packet 0 received safe → ACK with ackNo = 1. Packet 1 received safe → ACK with ackNo = 0.

$$\text{Acknowledgment number in Stop-and-Wait, ackNo} = (S+1) \bmod 2$$

Where S is the sequence number of the packet just sent and ackNo is the sequence number of the next packet expected.

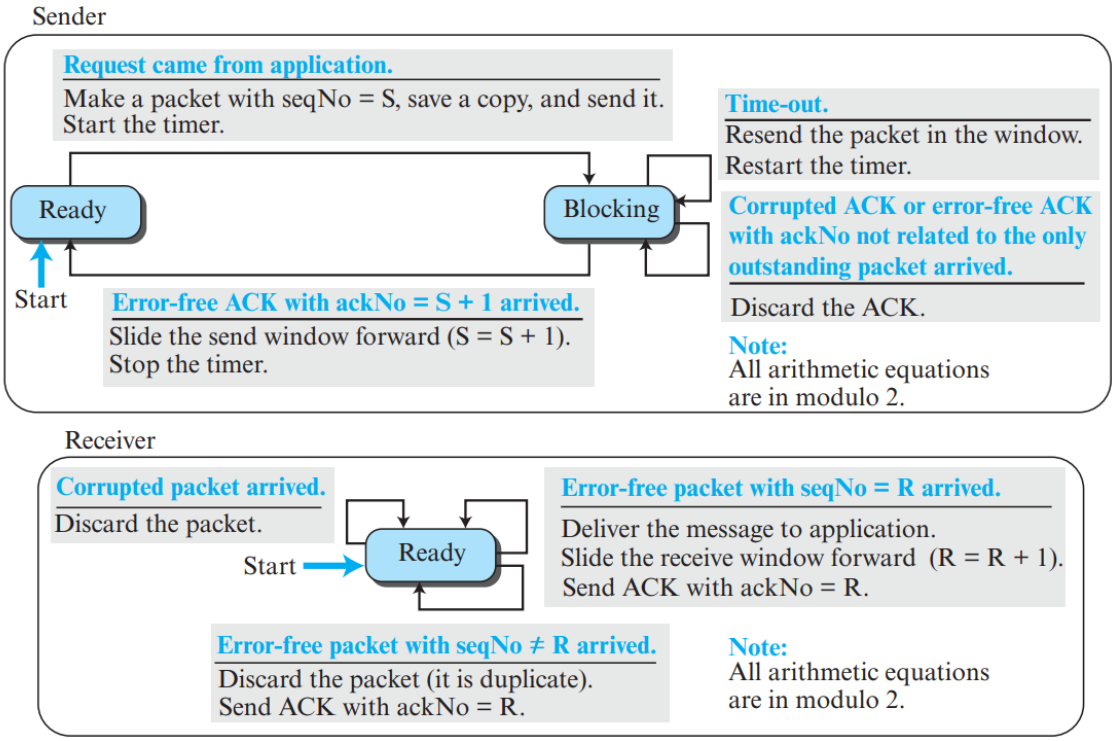
FSMs

Sender **(S initialised to 0)**

State	Event	Action
Ready	Request from the application	Make packet with seqNo = S, store a copy, send it, start the timer → **Blocking**
Blocking	Error-free ACK with ackNo = (S + 1) mod 2	Stop the timer, slide the window (S = (S + 1) mod 2) → **Ready**
Blocking	Corrupted ACK, or ACK with wrong ackNo	Discard the ACK
Blocking	Time-out	Resend the packet, restart the timer

Receiver **(R initialised to 0; always Ready)**

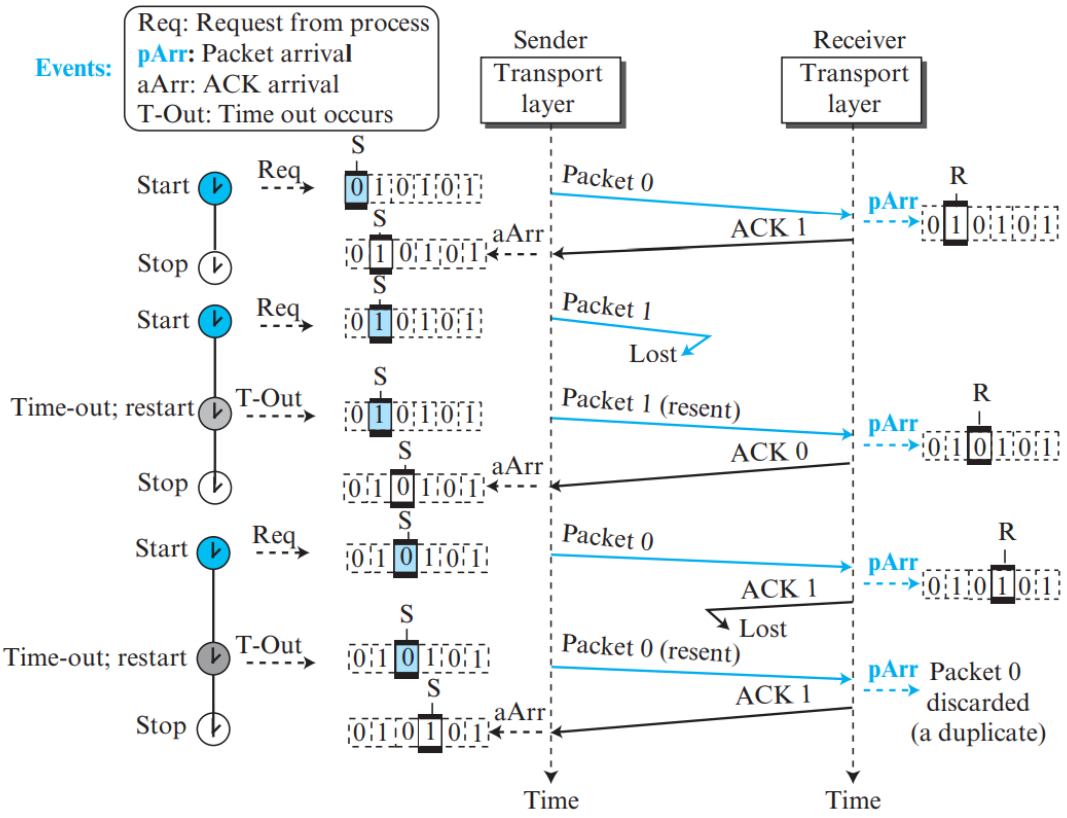
Event	Action
Error-free packet with seqNo = R	Deliver the message, slide the window (R = (R + 1) mod 2), send ACK with ackNo = R
Error-free packet with seqNo ≠ R (duplicate)	Discard it, send ACK with ackNo = R
Corrupted packet	Discard it



About Figure: Sender FSM with ready and blocking states and the receiver FSM with one ready state, showing each event and its action.

Lost packet and lost acknowledgment (Example)

- Packet 0 is sent and acknowledged.
- **Packet 1 is lost.** The timer expires, the sender resends packet 1, and it is acknowledged.
- Packet 0 is sent and received, but the **ACK is lost**. The sender cannot tell whether the packet or the ACK was lost, so it resends packet 0 after time-out.
- The receiver finds it is a **duplicate** (it expects packet 1), discards it and sends the ACK again.



About Figure: A time line of sender and receiver with the events: lost packet 1 (resent after time-out), and lost ACK (packet 0 resent and discarded as a duplicate).

Efficiency

- Stop-and-Wait is inefficient on a **thick** (high bandwidth) and **long** (long round-trip delay) channel.
- The **bandwidth-delay product** is the volume of the "pipe" in bits: the bits a sender can transmit while waiting for an ACK.

$$\text{Bandwidth-delay product, } BDP = B \times RTT$$

Where B = bandwidth (bits per second) and RTT = round-trip time (seconds). BDP is in bits.

$$\text{Link utilisation, } U = \frac{N \times L}{BDP} \times 100\,\%$$

Where U = utilisation (%), N = number of packets sent before waiting for an ACK, L = packet length in bits.

Example: line bandwidth 1 Mbps, round trip of 1 bit takes 20 ms, packets of 1000 bits (Stop-and-Wait, N = 1).

- BDP = $(1 \times 10^6) \times (20 \times 10^{-3})$ = **20,000 bits**.
- U = 1000 / 20,000 × 100 = **5 %**. The link capacity is wasted.

Example*:* same link, but a protocol that can send **15 packets** before waiting.

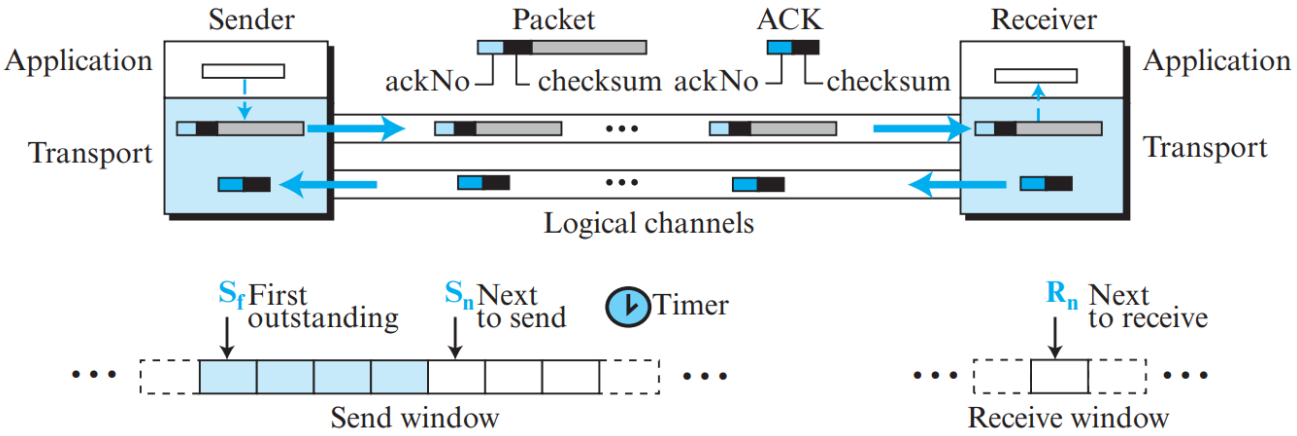
- U = (15 × 1000) / 20,000 × 100 = **75 %**.

Pipelining means starting a task before the previous one has ended. Stop-and-Wait has none. Go-Back-N and Selective-Repeat use it: several packets are sent before feedback arrives, which fills the pipe.

MODEL QUESTION (3 Marks): Explain the working of the Stop-and-Wait protocol. Describe the sender's handling of a lost packet and of a lost acknowledgment.

3. Go-Back-N Protocol (GBN)

- Several packets can be **outstanding** (sent but not acknowledged), so the channel stays busy.
- The **receiver can buffer only one packet**; there is no buffering of out-of-order packets.
- The sender keeps a copy of every sent packet until its ACK arrives.



About Figure: Sender and receiver with a send window and a receive window, a single timer at the sender, and several packets and ACKs in the channel at once.

Sequence numbers: modulo 2^m

Acknowledgment numbers: cumulative. The ackNo is the sequence number of the **next packet expected**. ackNo = 7 means all packets up to 6 arrived safe.

Send window

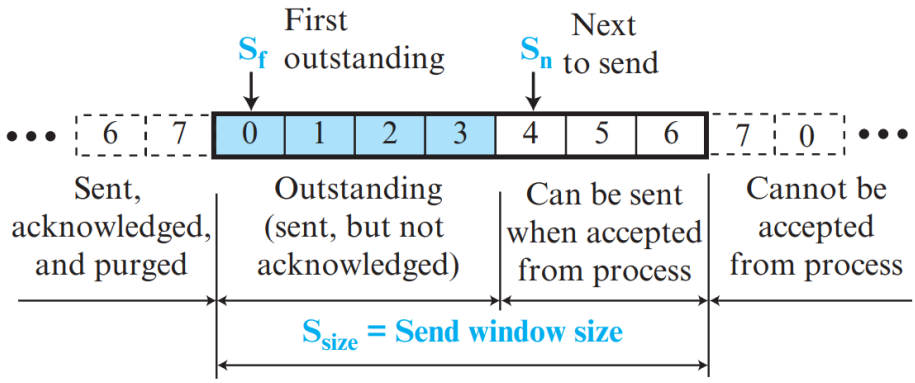
- An imaginary box covering the sequence numbers of packets that can be in transit or can be sent.
- **Maximum size = $2^m - 1$.**
- Defined by three variables: **Sf** (first outstanding packet), **Sn** (next packet to be sent), **Ssize** (window size, fixed).

Maximum send window in Go-Back-N, $S_{size} = 2^m - 1$

Where S_{size} = send window size and m = number of bits in the sequence-number field.

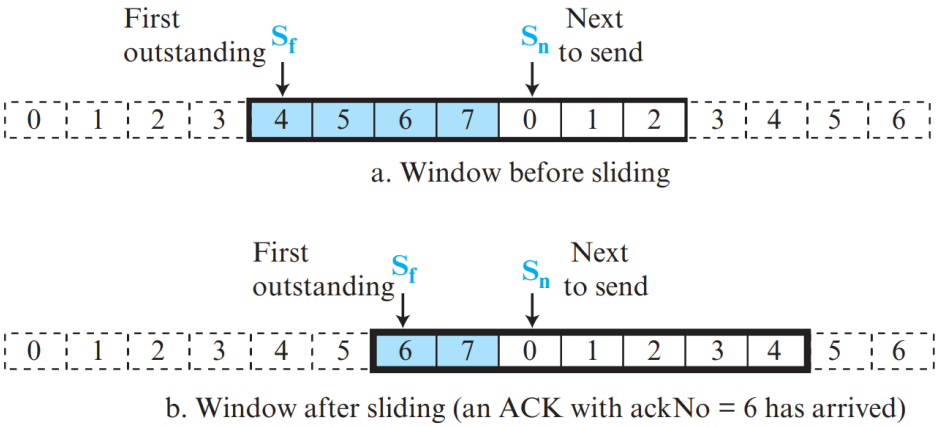
The window divides sequence numbers into four regions:

1. Left of the window: packets already **acknowledged** and purged.
2. **Outstanding** packets: sent, status unknown.
3. Packets that **can be sent** once received from the process.
4. Right of the window: numbers that **cannot be used** until the window slides.



About Figure: A send window of size 7 (m = 3) with Sf at the first outstanding packet and Sn at the next to send. The four regions are marked.

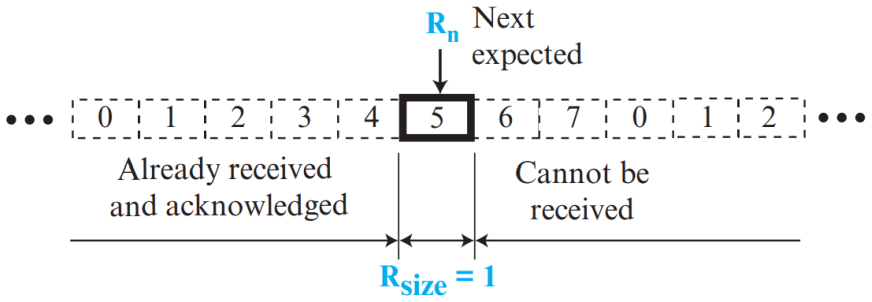
The window **slides** when an error-free ACK with ackNo ≥ Sf and < Sn (in modular arithmetic) arrives, setting Sf = ackNo.



About Figure: The send window before and after an ACK with ackNo = 6 arrives. Sf moves to 6.

Receive window

- **Size is always 1**, with one variable **Rn** (next packet expected).
- Only a packet with seqNo = Rn is accepted and acknowledged. Anything else (earlier or later) is **discarded**.
- The window slides **one slot** at a time: Rn = (Rn + 1) mod 2^m.



About Figure: A receive window of size 1 at Rn. Numbers to its left are received and acknowledged, and numbers to its right cannot be received.

Timer and resending

- Only **one timer** is used, because the timer of the first outstanding packet always expires first.
- On time-out, the sender **resends all outstanding packets**. This is why it is called **Go-Back-N**: it goes back N locations.
- Example: Sf = 3, Sn = 7 and the timer expires, so packets 3, 4, 5 and 6 are resent.

FSMs

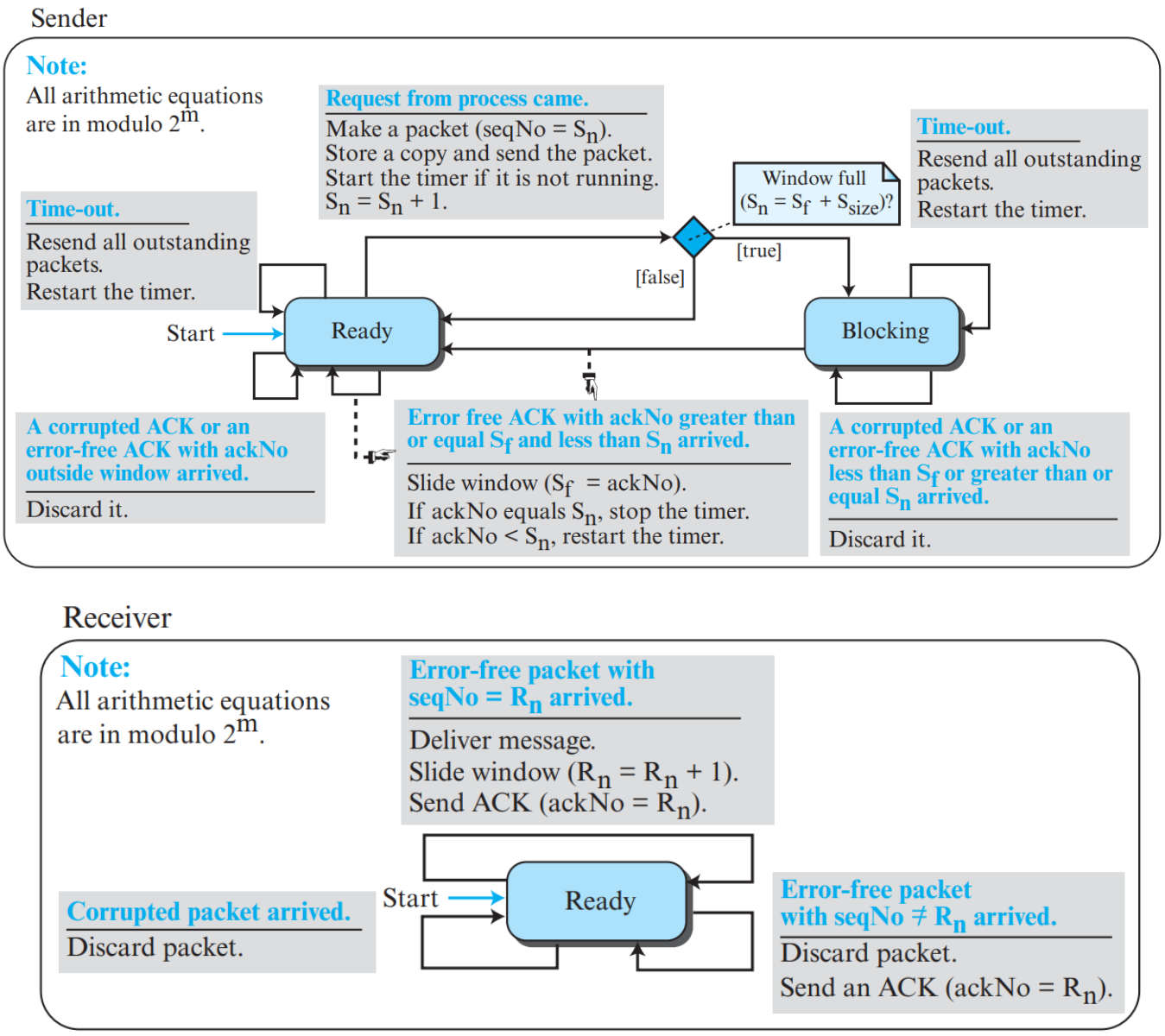
(Sf = Sn = 0 at the start; all arithmetic is modulo 2^m)

Sender

State	Event	Action
Ready	Request from the application	Make packet (seqNo = Sn), store a copy, send it, start the timer if not running, Sn = Sn + 1. If the window is full (Sn = Sf + Ssize), go to **Blocking**
Ready/ Blocking	Error-free ACK with Sf ≤ ackNo < Sn	Slide the window (Sf = ackNo). If ackNo = Sn, stop the timer; otherwise restart it. (Blocking → Ready)
Ready/ Blocking	Corrupted ACK or ACK outside the window	Discard it
Ready/ Blocking	Time-out	Resend all outstanding packets, restart the timer

***Receiver* (Rn = 0; always Ready)**

Event	Action
Error-free packet with seqNo = Rn	Deliver the message, Rn = (Rn + 1) mod 2^m, send ACK with ackNo = Rn
Error-free packet with seqNo ≠ Rn	Discard it, send ACK with ackNo = Rn
Corrupted packet	Discard it

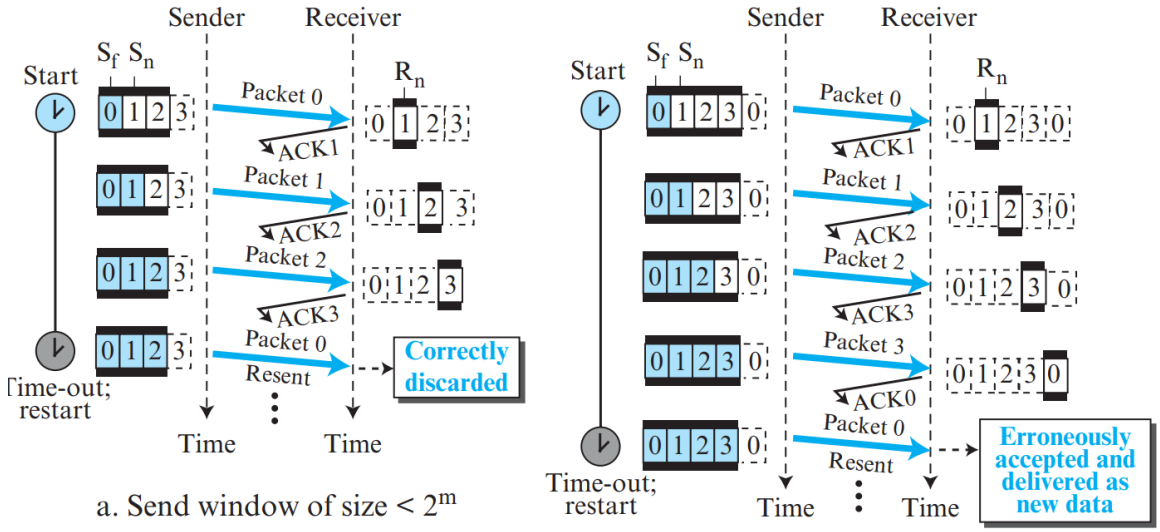


About Figure: The sender FSM with ready and blocking states and a window-full test, and the receiver FSM with one ready state.

Why the send window must be less than 2^m

Take m = 2, so numbers are 0 to 3.

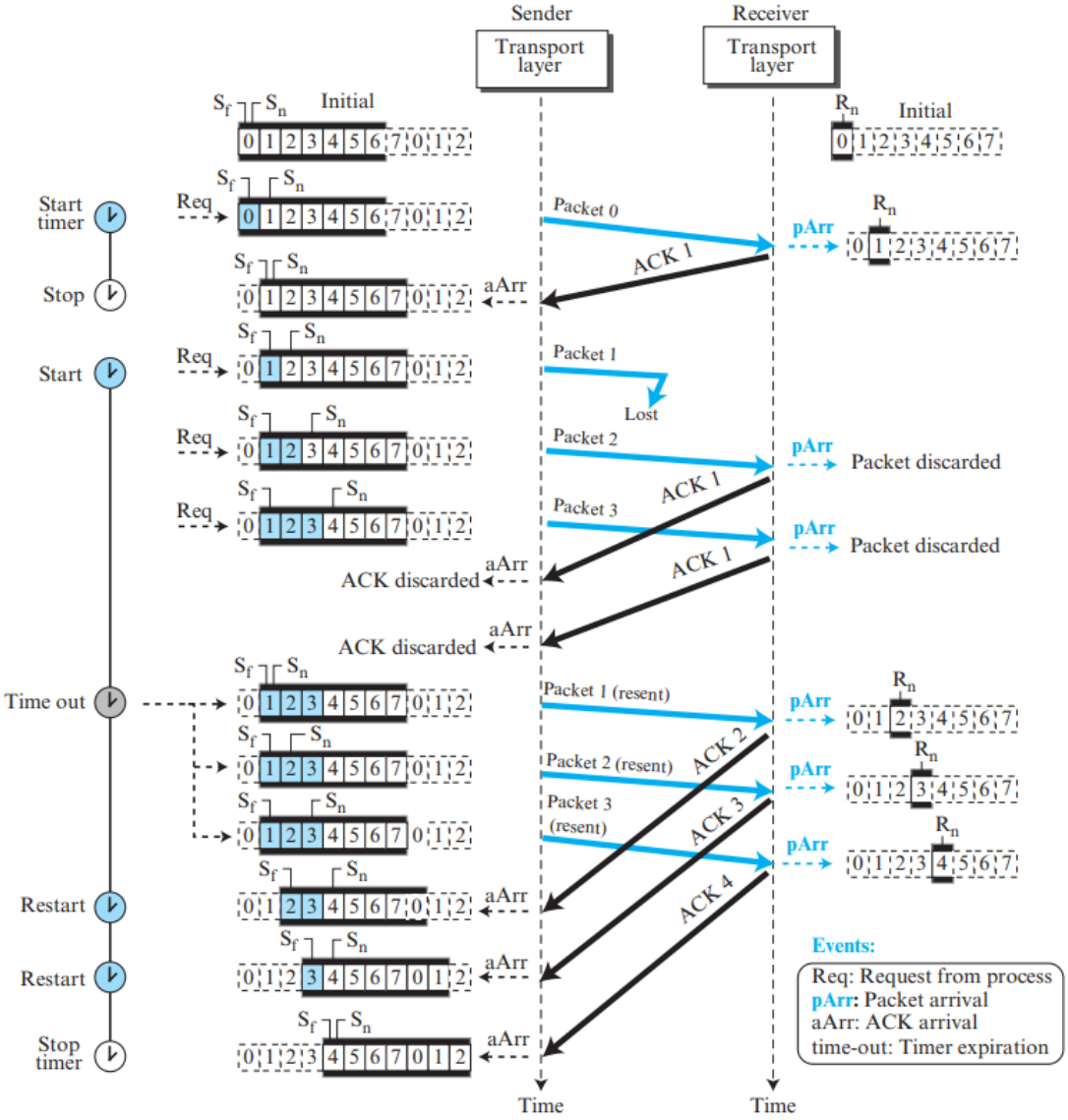
- **Window 3 (< 2^m):** if all three ACKs are lost, the timer expires and packets 0, 1, 2 are resent. The receiver now expects packet 3, so the resent packet 0 is **correctly discarded** as a duplicate.
- **Window 4 (= 2^m):** if all ACKs are lost, the resent packet 0 arrives when the receiver expects packet 0 (next cycle). It is **wrongly accepted as new data**.



Explanation of the Figure: (a) A send window of size 3 where the duplicate packet 0 is correctly discarded. (b) A window of size 4 where the duplicate is erroneously accepted as new data.

Examples

- **Example 3.7:** data packets are not lost, but ACKs are delayed and one is lost (ACK 2). **ACK 3 is cumulative**, so it acts as both ACK 2 and ACK 3.
- **Example 3.8:** packets 0 to 3 are sent and **packet 1 is lost**. The receiver discards packets 2 and 3 (out of order) and sends ACK 1 each time. These ACKs are useless to the sender, because ackNo equals Sf, not greater. After the time-out the sender **resends packets 1, 2 and 3**, which are acknowledged.



Explanation of the Figure: Flow diagram for Example - A time line in which packet 1 is lost, packets 2 and 3 are discarded at the receiver, duplicate ACK 1 messages are discarded, and packets 1, 2, 3 are resent after the time-out.

Go-Back-N versus Stop-and-Wait: Stop-and-Wait is a Go-Back-N protocol with m = 1, send window size $2^m - 1 = 1$, and modulo-2 arithmetic.

Drawback of GBN: if the network loses many packets, every loss makes the sender resend all outstanding packets, even ones received safely. This adds to congestion and can cause an avalanche effect.

4. Selective-Repeat Protocol (SR)

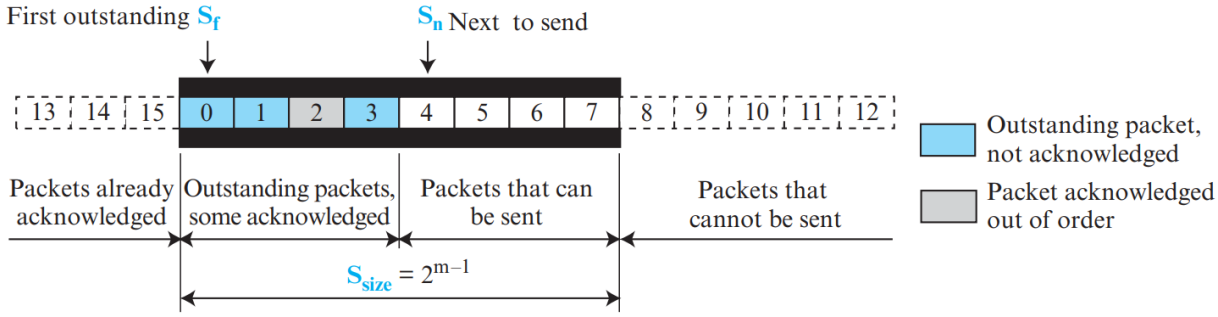
SR resends **only the packets that are actually lost**.

Windows

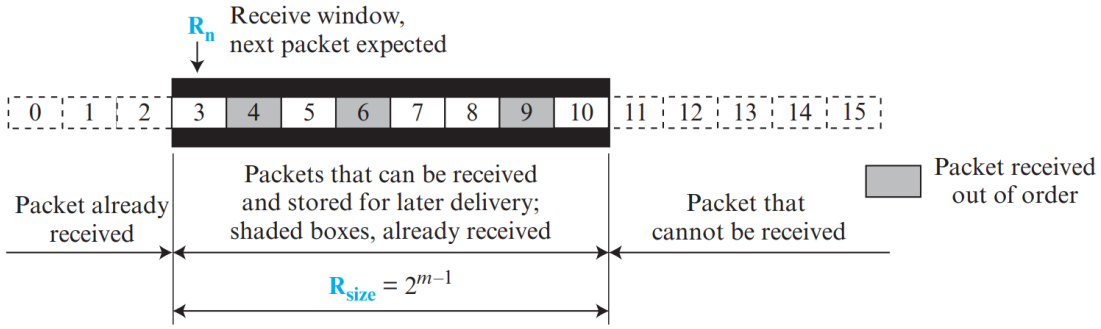
- **Send window maximum size** = $2^{(m-1)}$ (much smaller than in GBN).
- **Receive window size = send window size.**
- The receiver **accepts and stores out-of-order packets**. But a reliable protocol never delivers packets out of order to the application.

Maximum window size in Selective-Repeat, $S_{size} = R_{size} = 2^{m-1}$

Where S_size and R_size are the send and receive window sizes and m is the sequence-number field size in bits. Example: for m = 4, the maximum window is 8 (compared with 15 in GBN).



About Figure: A send window of size $2^{(m-1)}$ showing outstanding packets, some already acknowledged out of order, and packets that can or cannot be sent.



About Figure: A receive window with shaded slots for packets received out of order and waiting for earlier packets. Slots outside the window cannot be received.

Timer

In theory, one timer per packet, so only the expired packet is resent. In practice most implementations use **one timer**.

Acknowledgments

In SR, an ackNo is **selective**: it gives the sequence number of **one single packet** received safe, with no feedback about others.

Example: the sender sends packets 0 to 5 and receives ackNo = 3.

- **GBN:** packets 0, 1, 2 received safe, and the receiver expects packet 3.
- **SR:** packet 3 received safe. Nothing is said about the others.

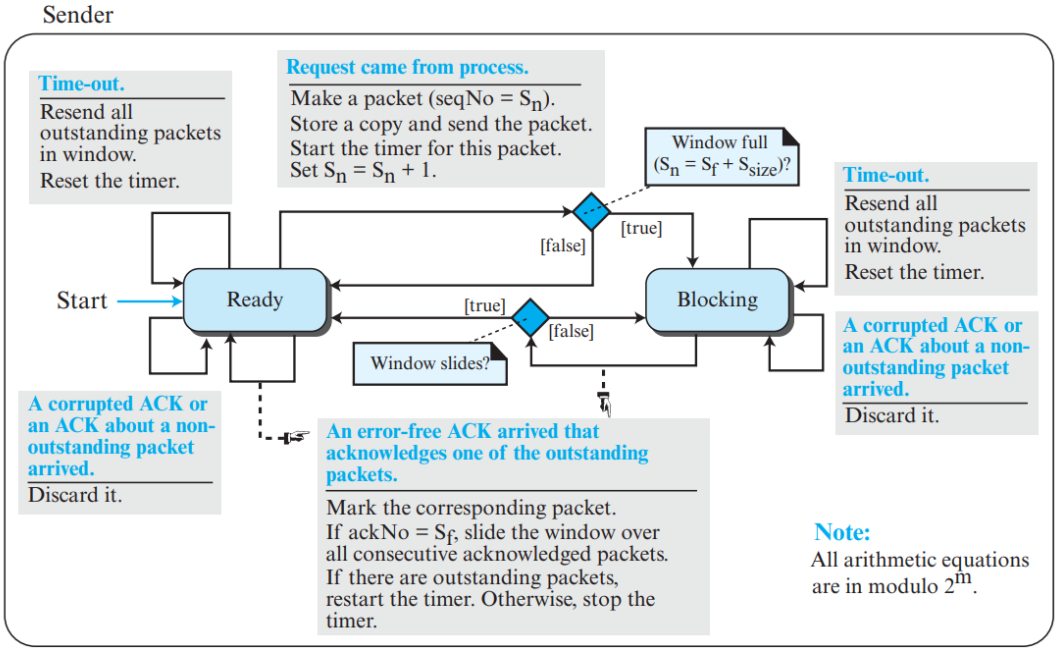
FSMs

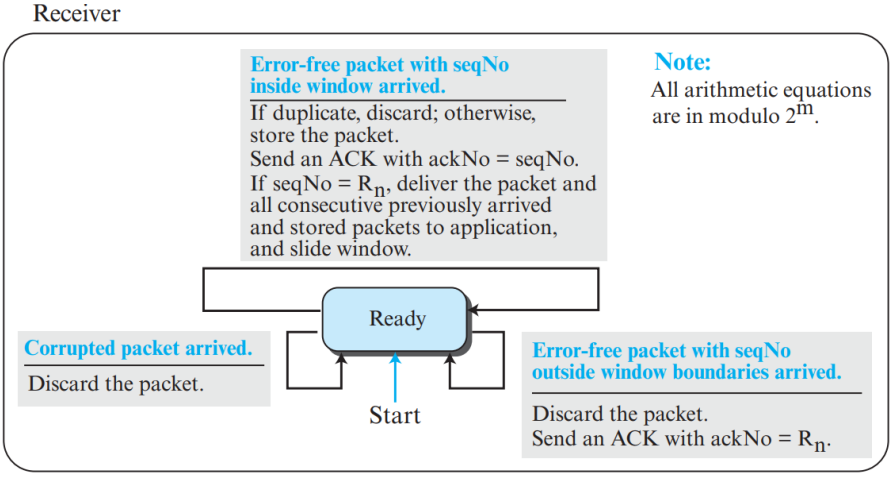
Sender

Event	Action
Request from the application	Make packet (seqNo = Sn), store a copy, send it, start the timer for it, Sn = Sn + 1. If the window is full, go to **Blocking**
Error-free ACK for an outstanding packet	**Mark that packet** acknowledged. If ackNo = Sf, slide the window over all consecutive acknowledged packets. Restart the timer if packets remain outstanding, otherwise stop it
Corrupted ACK or ACK for a non-outstanding packet	Discard it
Time-out	Resend all unacknowledged packets in the window, reset the timer

Receiver (always Ready)

Event	Action
Error-free packet with seqNo **inside** the window	Store it if not a duplicate, send ACK with ackNo = seqNo. If seqNo = Rn, deliver it and all consecutive stored packets and slide the window
Error-free packet with seqNo **outside** the window	Discard it, send ACK with ackNo = Rn (so the sender can slide its window if earlier ACKs were lost)
Corrupted packet	Discard it

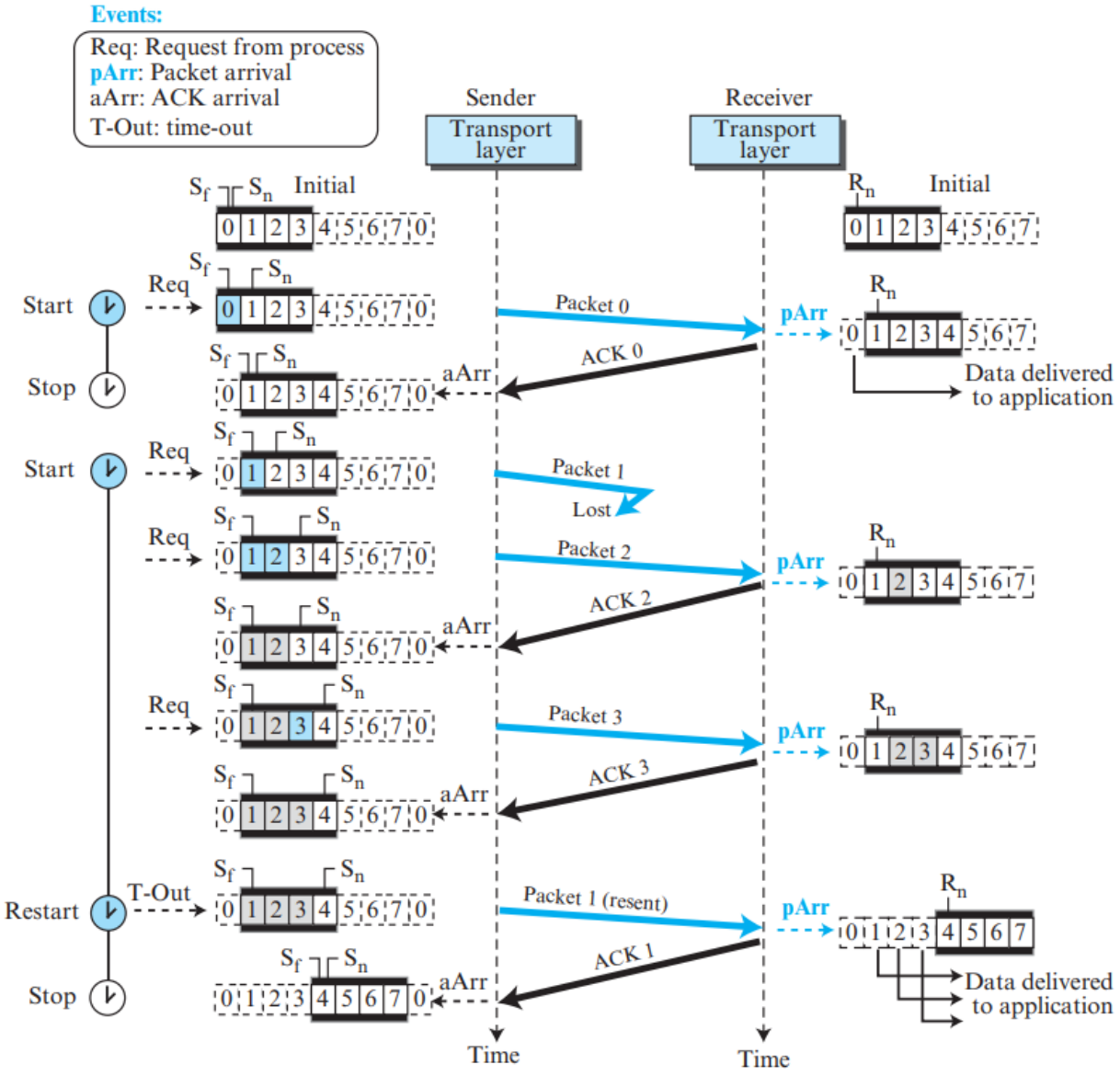




About Figure: The sender FSM (ready and blocking) and receiver FSM (ready) for Selective-Repeat, with the events and actions above.

Example: (packet 1 lost, as in previous Example)

- Packet 0 is acknowledged. **Packet 1 is lost.**
- Packets 2 and 3 arrive out of order. They are **stored and acknowledged** but not delivered, because packet 1 is missing.
- After time-out only **packet 1** is resent and acknowledged.
- Packets 1, 2, 3 are now delivered together.
- Delivery to the application needs two conditions: **(1) a set of consecutive packets has arrived, and (2) the set starts at the beginning of the window.**

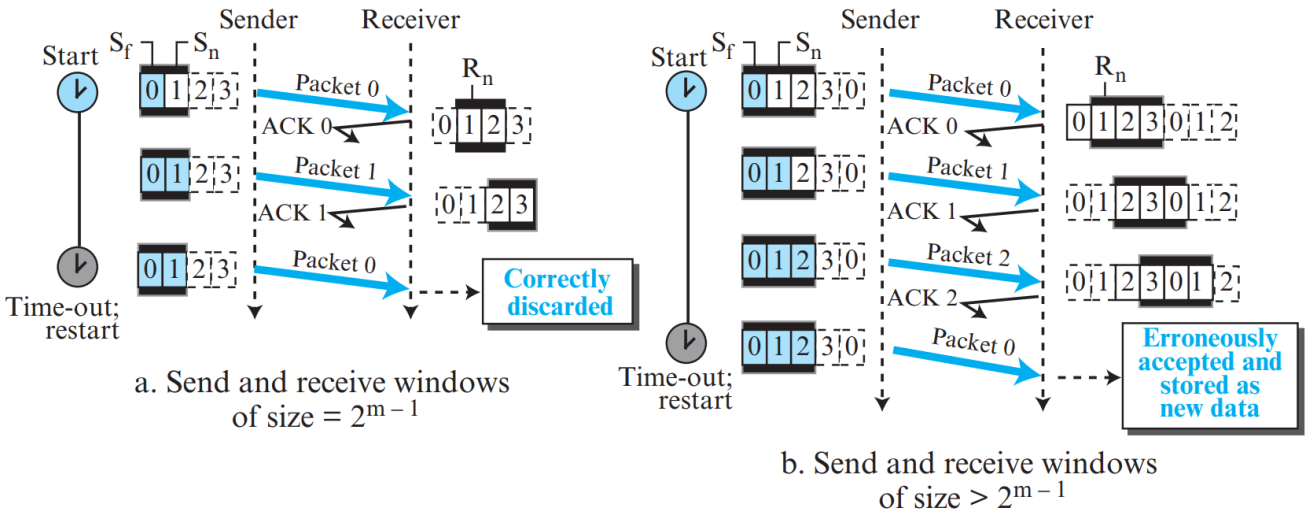


About Figure: Time line for SR with packet 1 lost. Packets 2 and 3 are stored and marked, and packets 1, 2, 3 are delivered to the application when the resent packet 1 arrives.

Why the window size is at most $2^{(m-1)}$

Take m = 2.

- **Window 2 (=** $2^{(m-1)}$**):** if all ACKs are lost, the resent packet 0 is outside the receiver's window (it now expects 2, 3). It is **correctly discarded** as a duplicate.
- **Window 3:** the resent packet 0 falls inside the receiver's window, so it is **wrongly accepted** as a new packet of the next cycle.

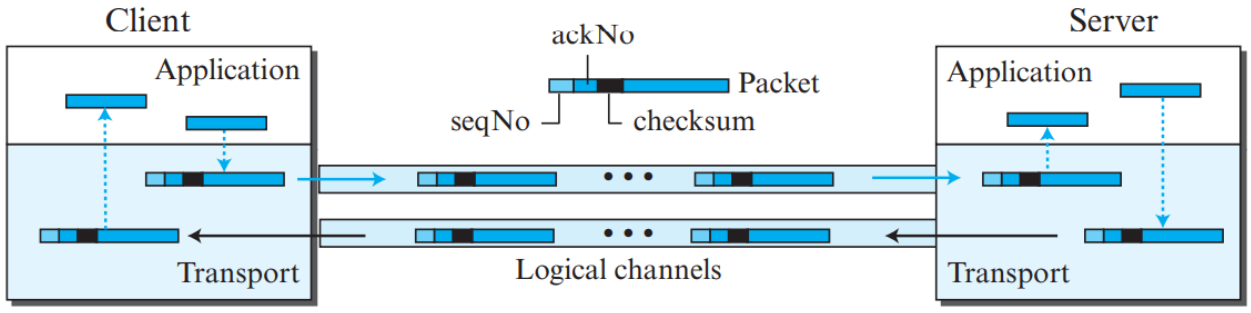


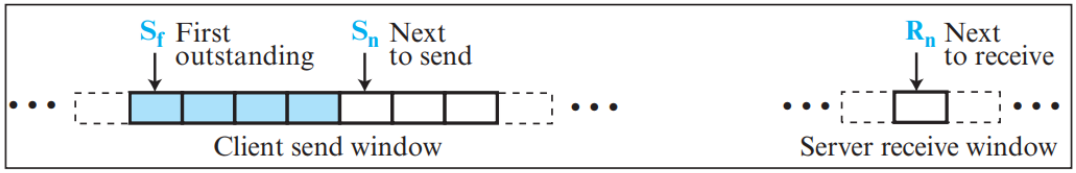
About Figure: (a) Windows of size 2 where the duplicate is correctly discarded. (b) Windows larger than $2^{(m-1)}$, where the duplicate is erroneously accepted as new data.

Parameter	Stop-and-Wait	Go-Back-N	Selective-Repeat
Send window size	1	Up to $2^m - 1$	Up to $2^{(m-1)}$
Receive window size	1	1	Same as send window
Out-of-order packets	Not applicable	**Discarded**	**Stored** until the missing ones arrive
ACK type	ackNo = next expected	**Cumulative**	**Selective** (single packet)
On loss	Resend the packet	Resend **all** outstanding packets	Resend **only** the lost packet
Receiver complexity	Simple	Simple (one variable)	More complex (buffers, marking)
Pipelining	No	Yes	Yes

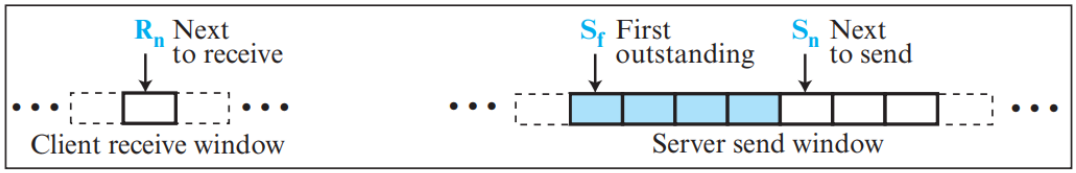
Bidirectional Protocols: Piggybacking

- The protocols above are **unidirectional**. In real life, data and ACKs flow in both directions.
- **Piggybacking** improves efficiency: a packet carrying data from A to B also carries the **acknowledgment** for packets received from B (and the reverse).
- Each end then uses **two independent windows**: one send and one receive.





Windows for communication from client to server



Windows for communication from server to client

About Figure: The client and the server each have a send window and a receive window. Every packet carries seqNo, ackNo and a checksum together with data.

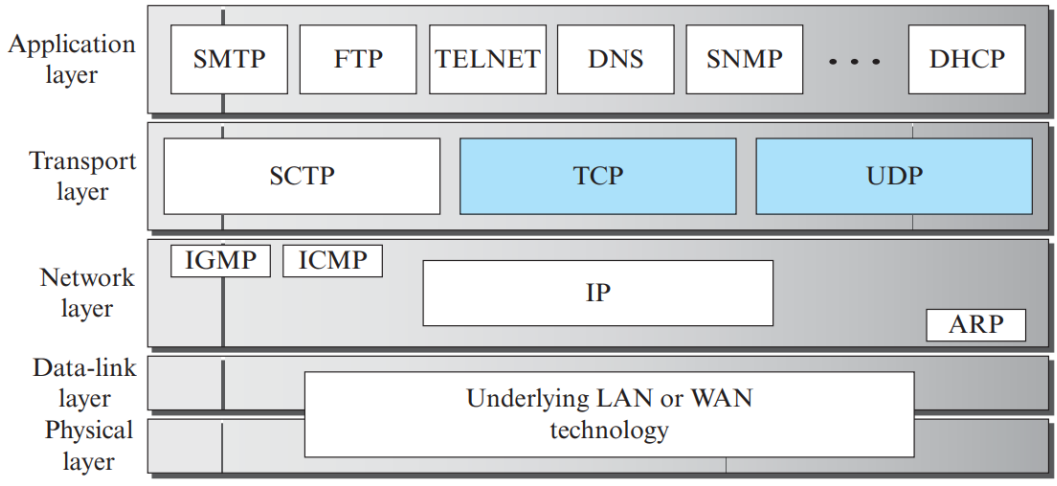
Internet Transport-Layer Protocols

- The Internet uses two main transport protocols, **UDP** and **TCP**, placed between the application layer and the network layer.
- **UDP:** unreliable, connectionless; used for simplicity and efficiency where error control is done by the application.
- **TCP:** reliable, connection-oriented; used where reliability is important.
- Both use **port numbers** for process-to-process communication.

UDP & TCP

1. User Datagram Protocol (UDP)

UDP is a **connectionless, unreliable** transport protocol. It adds nothing to IP except **process-to-process** instead of host-to-host communication. It is very simple and has minimum overhead. A process that sends a small message and does not care much about reliability can use UDP.



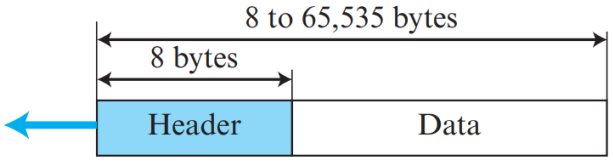
About Figure: UDP and TCP (and SCTP) sit between the application protocols above (SMTP, FTP, TELNET, DNS, SNMP, DHCP) and IP below.

User Datagram

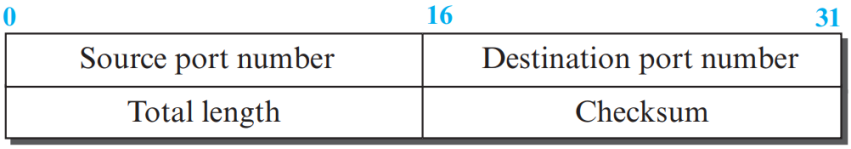
A UDP packet is a **user datagram** with a **fixed 8-byte header** of four 2-byte (16-bit) fields.

Field	Size	Meaning
Source port number	16 bits	Port of the sending process
Destination port number	16 bits	Port of the receiving process
Total length	16 bits	Header plus data, in bytes
Checksum	16 bits	Optional error check

- The 16-bit length can define 0 to 65,535 bytes, but a user datagram travels inside an IP datagram of at most 65,535 bytes. So the total length must be less.



a. UDP user datagram



b. Header format

About Figure: (a) A user datagram is an 8-byte header followed by data (8 to 65,535 bytes in all). (b) The header holds source port, destination port, total length and checksum, each 16 bits.

Example: A UDP header in hexadecimal is CB84000D001C001C. Find the details.

Item	Working	Result
Source port	First 4 hex digits CB84 = 12×4096 + 11×256 + 8×16 + 4	**52,100**
Destination port	Next 4 digits 000D	**13**
Total length	Next 4 digits 001C	**28 bytes**
Data length	28 − 8 (header)	**20 bytes**
Direction	Destination port 13 is a well-known port	**Client to server**
Client process	Port 13	**Daytime**

UDP Services

Process-to-process communication. Uses socket addresses (IP address + port number).

Connectionless service

- Each user datagram is **independent**, even if from the same source process to the same destination.
- Datagrams are **not numbered**. There is no connection establishment or termination.
- Each datagram may take a different path.
- A process cannot send a stream and expect UDP to cut it into related datagrams. Each request must fit in **one** user datagram.

$$\text{Maximum message size for UDP, } M_{max} = 65535 - 8 - 20 = 65507$$

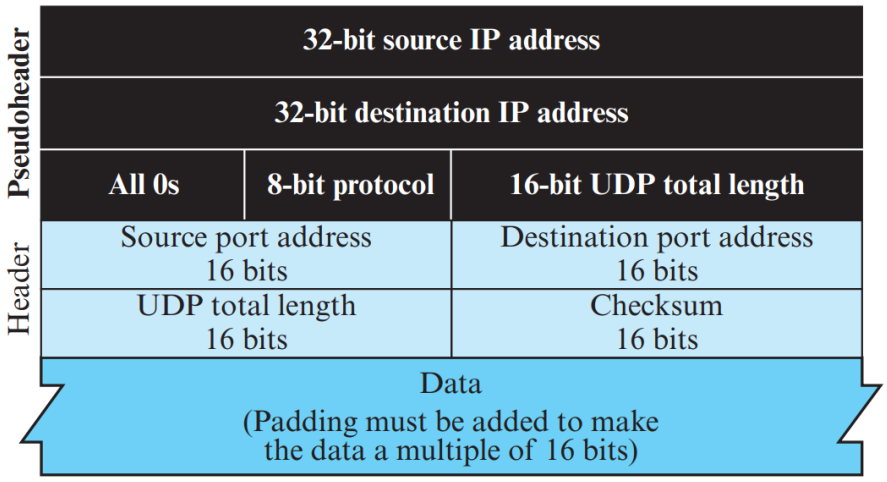
Where M_{max} = largest message (bytes) that can use UDP, 8 = UDP header and 20 = IP header.

Flow control: none. No window mechanism, so the receiver may overflow. The process must provide it if needed.

Error control: none, except the checksum. The sender does not know if a message is lost or duplicated. A datagram with a bad checksum is **silently discarded**. The process must provide error control if needed.

Checksum

- Calculated over three parts: a **pseudoheader**, the **UDP header** and the **data** from the application.
- The **pseudoheader** is part of the IP header (source IP, destination IP, protocol, UDP length) with some fields filled with 0s.
- *Why include the pseudoheader?* If the IP header were corrupted, a datagram could arrive "safe" but be delivered to the **wrong host**.
- The **protocol field** (value **17** for UDP) ensures the packet belongs to UDP and not TCP. If it changes in transit, the receiver detects it and drops the packet.
- Padding is added to make the data a multiple of 16 bits.



About Figure: The pseudoheader holds the 32-bit source IP address, the 32-bit destination IP address, 8 bits of zeros, the 8-bit protocol field and the 16-bit UDP length. It is placed before the UDP header (ports, length, checksum) and the data.

Optional checksum

- The sender may choose **not** to calculate it. The field is then filled with **all 0s**.
- If the calculated checksum is all 0s, it is changed to **all 1s** before sending (the sum is complemented twice).

***Example:* checksum value is sent in each case?**

Situation	Value sent	Reason
(a) Checksum not included	**All 0s**	Shows the checksum was not calculated
(b) Included, sum is all 1s	**All 1s**	Complementing the sum gives all 0s, which would look like case (a), so it is complemented again
(c) Included, sum is all 0s	Never happens	Every term would have to be 0, but pseudoheader fields are nonzero

Congestion control: none. UDP assumes its packets are small and sporadic and cannot cause congestion. This may not hold for real-time audio and video.

Encapsulation and decapsulation: UDP encapsulates a message in a user datagram and decapsulates at the receiver.

Queuing: queues are associated with ports. A client process requests a port number from the OS. Some implementations create an incoming and an outgoing queue per process, others only an incoming queue.

Multiplexing and demultiplexing: a host has **one UDP** but many processes may use it, so UDP multiplexes and demultiplexes.

Comparison with the simple protocol: UDP is the connectionless simple protocol plus an **optional checksum** for error detection (no feedback is sent to the sender).

UDP Applications

Features

Feature	Advantage	Disadvantage
Connectionless service	Short request and response need only **2 packets** (a connection-oriented service needs at least 9), so less delay	Long messages split by the application may arrive **out of order**
Lack of error control	No waiting for resent parts, so no **uneven delay** (good for real-time audio and video)	Lost or corrupted data is not recovered
Lack of congestion control	Does not add extra traffic in an error-prone network (TCP resends and may worsen congestion)	Real-time traffic may create congestion

Examples

- **DNS:** uses UDP. The request and response each fit in one user datagram, so connectionless service is no problem.
- **SMTP:** cannot use UDP. E-mail can be long, and the user datagrams may arrive out of order. The extra delay of connection-oriented service is not critical for e-mail.
- **Large file download:** UDP is not suitable. Reliability is needed, and uneven delay does not matter.
- **Skype or interactive real-time:** UDP is suitable. A lost frame is simply ignored (a very short blank spot), whereas resending would break synchronisation.

Typical UDP uses

- Simple **request-response** communication with little concern for flow and error control (not bulk transfer like FTP).
- Processes with **internal flow and error control**, for example **TFTP**.
- **Multicasting** (built into UDP software, not TCP).
- Management processes such as **SNMP**.
- Route-updating protocols such as **RIP**.
- **Interactive real-time** applications that cannot tolerate uneven delay.

Transmission Control Protocol (TCP)

TCP is a **connection-oriented, reliable** protocol and the most common transport protocol in the Internet.

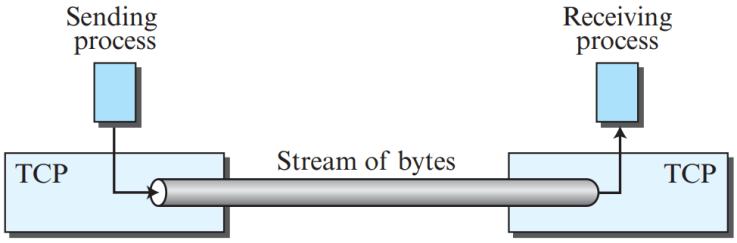
It has *three* phases: **connection establishment, data transfer and connection teardown**. It uses a combination of GBN and SR ideas: **checksum** (error detection), **retransmission** of lost or corrupted segments, **cumulative and selective acknowledgments**, and **timers**.

TCP Services

Process-to-process communication: uses port numbers.

Stream delivery service

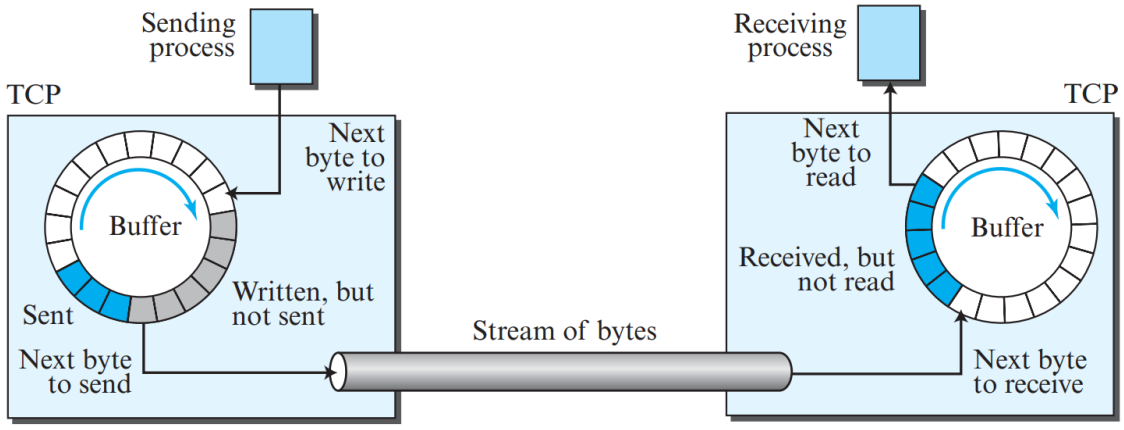
- UDP processes send **messages with boundaries**. TCP is **stream-oriented**: the sender gives data as a **stream of bytes**, and the receiver reads it as a stream of bytes.
- TCP creates an imaginary **"tube"** between the processes.



About Figure: The sending process writes a stream of bytes into TCP, and the receiving process reads the stream from TCP, as if connected by a tube.

Sending and receiving buffers

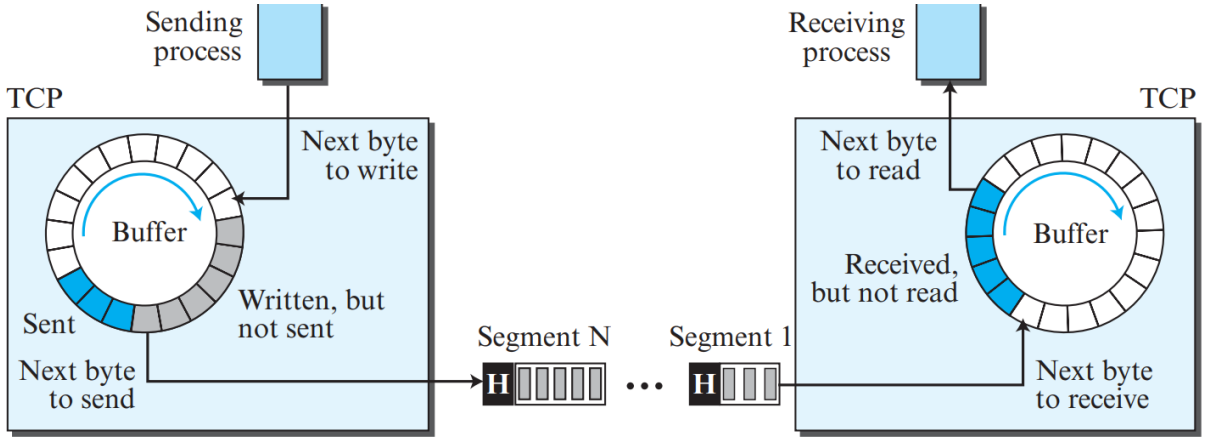
- The processes may write and read at different rates, so TCP uses a **sending buffer** and a **receiving buffer**, one for each direction. A buffer is often a **circular array of 1-byte locations**.
- **Sender buffer** has three kinds of chambers: **empty** (can be filled by the process), **sent but not acknowledged** (kept until the ACK arrives), and **bytes to be sent**. Acknowledged chambers are recycled.
- **Receiver buffer** has two kinds: **empty** (to be filled from the network) and **received bytes** waiting for the process to read them.



About Figure: Circular buffers at the sender (next byte to write, sent bytes, bytes written but not sent) and at the receiver (next byte to read, received but not read, next byte to receive).

Segments

- The network layer sends **packets**, not a stream. TCP groups a number of bytes into a packet called a **segment**, adds a header, and gives it to IP.
- Segments need not be the same size. They may arrive out of order, be lost or corrupted and be resent. All this is handled by TCP and hidden from the process.



About Figure: Bytes taken from the sending buffer are grouped into segments, each with a header (H), and the receiver rebuilds the byte stream in its buffer.

Full-duplex communication: data flows in **both directions at the same time**. Each endpoint has its own sending and receiving buffer.

Multiplexing and demultiplexing: done like UDP, but a **connection must be established for each pair of processes**.

Connection-oriented service: between processes A and B, three phases occur: (1) the two TCPs establish a **logical connection**, (2) data are exchanged in both directions, (3) the connection is terminated. It is a **logical, not physical**, connection: segments travel in IP datagrams and may be out of order, lost, corrupted or on different routes.

Reliable service: uses **acknowledgments** to check safe arrival.

TCP Features

Numbering system: TCP numbers **bytes**, not segments. The header has no segment-number field; it has a **sequence number** and an **acknowledgment number**, both referring to byte numbers.

- **Byte number:** TCP numbers all data bytes in a connection, independently in each direction. Numbering starts at an **arbitrary number between 0 and** $2^{32} - 1$. For example, if the first number is 1,057 and 6,000 bytes are sent, they are numbered 1,057 to 7,056.
- **Sequence number:** the number of the **first data byte** in the segment.
    1. The first segment's sequence number is the **ISN (initial sequence number)**, a random number.
    2. Any other segment's sequence number is the previous segment's sequence number plus the **number of bytes (real or imaginary)** it carried.
- Control segments with no data, such as SYN and FIN, are treated as carrying **one imaginary byte** so they can be acknowledged.

$$\text{Sequence number of the next segment, } seq_{next} = seq_{prev} + n$$

Where seq_prev = sequence number of the previous segment and n = number of bytes (real or imaginary) it carried.

Example: A file of 5,000 bytes is sent in five segments of 1,000 bytes each. The first byte is numbered 10,001.

Segment	Sequence number	Byte range
1	10,001	10,001 to 11,000
2	11,001	11,001 to 12,000
3	12,001	12,001 to 13,000
4	13,001	13,001 to 14,000
5	14,001	14,001 to 15,000

Acknowledgment number

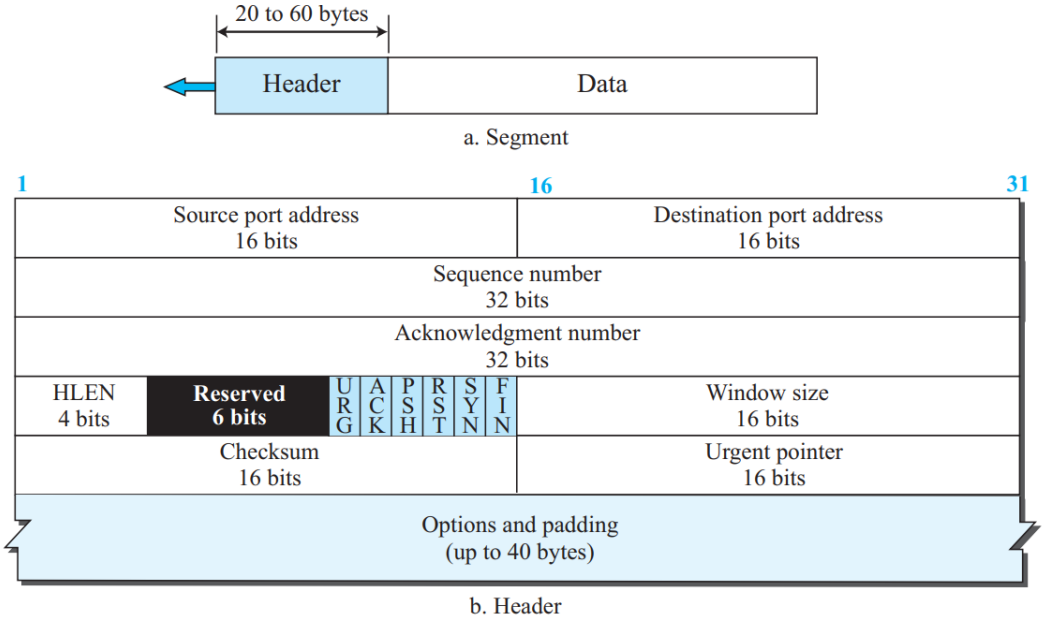
- Defines the number of the **next byte the party expects to receive**.
- It is **cumulative**: the party takes the number of the last byte received safely, adds 1 and announces it. An ackNo of 5,643 means all bytes up to 5,642 have arrived.

$$\text{Acknowledgment number, } ackNo = x + 1$$

Where x = number of the last byte received safe and in order.

TCP Segment

A segment has a **header of 20 to 60 bytes** followed by data. The header is 20 bytes without options and up to 60 bytes with up to 40 bytes of options.



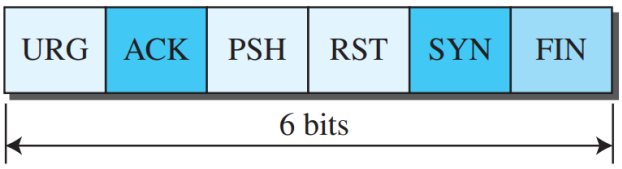
About Figure: The header has source port, destination port, sequence number, acknowledgment number, HLEN, reserved bits, six control flags, window size, checksum, urgent pointer, and options with padding. Data follows the header.

Field	Size	Function
Source port	16 bits	Port of the sending application
Destination port	16 bits	Port of the receiving application
Sequence number	32 bits	Number of the first data byte in the segment. During establishment each side picks a random **ISN**
Acknowledgment number	32 bits	Next byte expected. If byte x was received, returns x + 1. Can be piggybacked with data
Header length (HLEN)	4 bits	Header length in **4-byte words**; value between 5 (20 bytes) and 15 (60 bytes)
Reserved	6 bits	For future use
Control	6 bits	Six flags (below)
Window size	16 bits	The **receive window (rwnd)** in bytes, set by the receiver. Maximum 65,535 bytes. The sender must obey it
Checksum	16 bits	Same method as UDP, but **mandatory**. Pseudoheader protocol value is **6**
Urgent pointer	16 bits	Valid only if URG is set. Added to the sequence number it gives the last urgent byte
Options	up to 40 bytes	Optional information

$$\text{TCP header length, Header length (bytes)} = \text{HLEN} \times 4$$

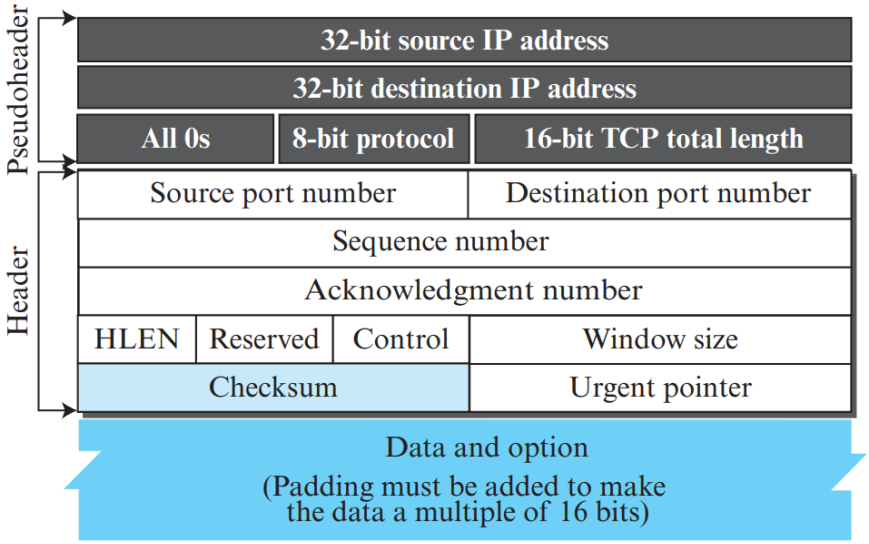
Where HLEN is the 4-bit field (5 to 15), giving 20 to 60 bytes.

Control flags



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

About Figure: The six control bits URG, ACK, PSH, RST, SYN, FIN with a short description of each.



About Figure: The same pseudoheader as in UDP (source IP, destination IP, zeros, protocol, TCP total length) placed before the TCP header and data for the checksum.

Encapsulation: a TCP segment is encapsulated in an **IP datagram**, which is encapsulated in a **frame** at the data-link layer.

A TCP Connection

How can TCP be connection-oriented over connectionless IP?

The TCP connection is **logical**. TCP uses IP only to deliver individual segments and controls the connection itself. Retransmission and reordering are done by TCP; IP is unaware.

Connection Establishment: Three-Way Handshake

Both sides send data (full duplex), so each side must **initialise** and get approval from the other first.

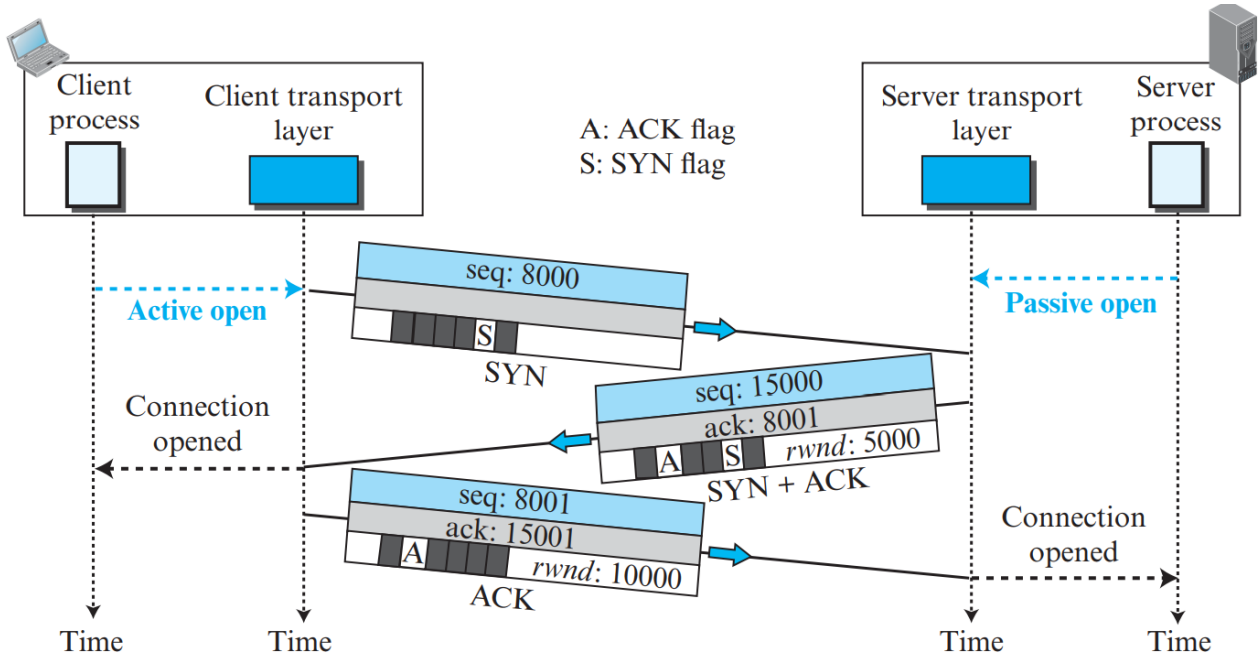
- The **server** program tells its TCP it is ready to accept connections: **passive open**.
- The **client** issues an **active open** to a particular server.

Steps:

1. **SYN segment (client to server):** Only the **SYN flag** is set. The client picks a random **initial sequence number (ISN)**. No acknowledgment number and no window size (a window is defined only with an ACK). It carries no data, but it **consumes one sequence number**: it is treated as carrying **one imaginary byte**.
2. **SYN + ACK segment (server to client):** Flags SYN and ACK are set. It has a **dual purpose**:
   - As a **SYN**, it starts communication in the other direction with the server's own ISN.
   - As an **ACK**, it acknowledges the client's SYN. The ackNo is the client's ISN + 1 (the next sequence number expected). It also announces the server's **rwnd**.
   - It carries no data, but it **consumes one sequence number**, because it also needs to be acknowledged.

3. **ACK segment (client to server):** Acknowledges the server's SYN + ACK (ackNo = server ISN + 1). If it carries **no data it consumes no sequence number**. Some implementations let this segment carry the first data, in which case it consumes as many numbers as the data bytes.

Example: the client's ISN is 8000, so the SYN is acknowledged with ackNo = 8001. The server's ISN is 15000, so the third segment carries ackNo = 15001.



About Figure: A time line: SYN (seq 8000) from client to server, then SYN + ACK (seq 15000, ack 8001) from server to client, then ACK (seq 8001, ack 15001). Both ends show the connection as opened.

Why a SYN consumes one sequence number:

The SYN segment carries no data, but it must be **acknowledged**. The acknowledgment number tells the sender which byte is expected next, so the SYN has to occupy a sequence number position (one imaginary byte). The ACK then confirms it by announcing ISN + 1. Segments that only acknowledge (an ACK without data) need no acknowledgment themselves, so they consume none.

Rules

- A SYN segment cannot carry data, but it consumes one sequence number.
- A SYN + ACK segment cannot carry data, but it consumes one sequence number.
- An ACK segment, if carrying no data, consumes no sequence number.

SYN flooding attack

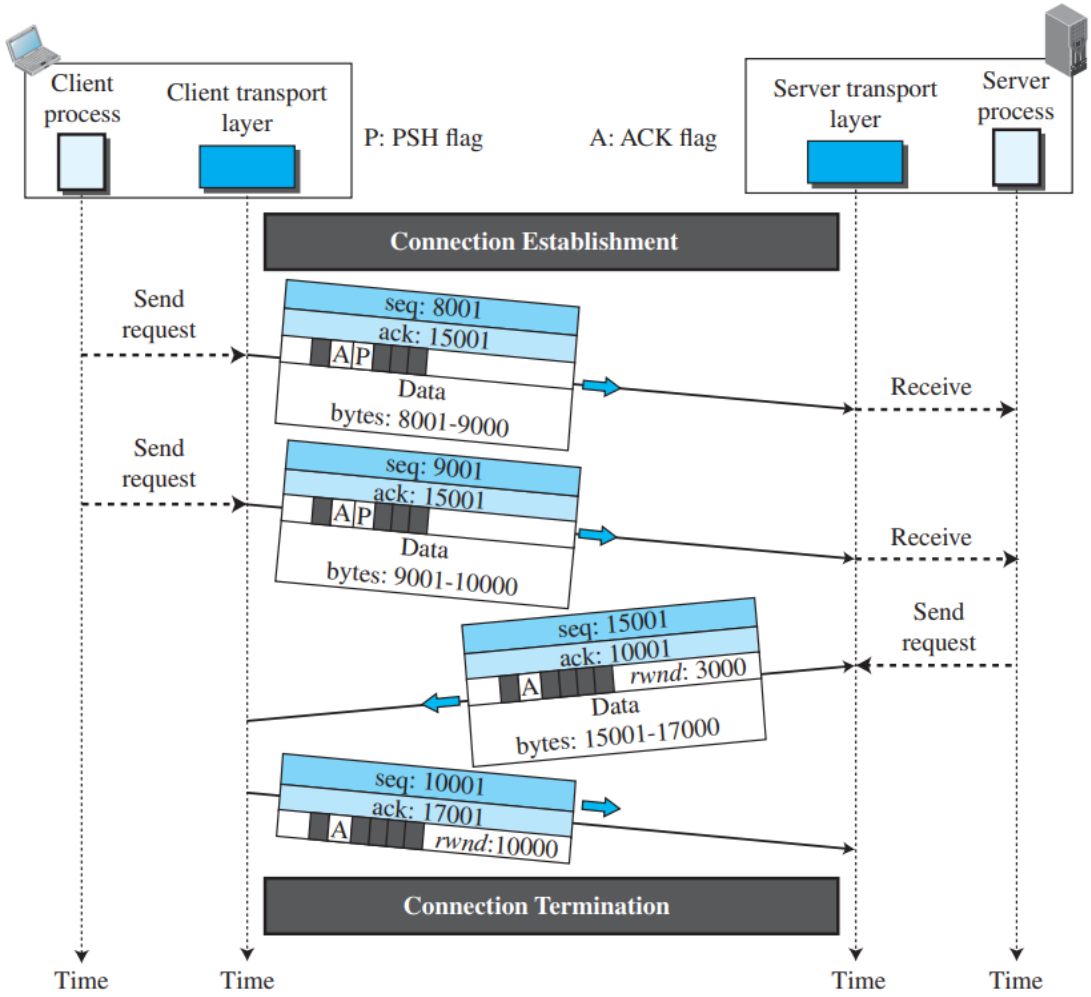
- Malicious attackers send **many SYN segments with faked source IP addresses**. The server allocates resources (transfer control block tables, timers) for each and sends SYN + ACK to the fake clients, which are lost.
- The server waits for a third segment that never comes, and resources are wasted. Finally it **runs out of resources** and cannot accept valid requests.
- It is a **denial of service attack**.
- **Remedies:** limit connection requests in a period; filter datagrams from unwanted source addresses; **postpone resource allocation** until the request is verified as coming from a valid IP address, using a **cookie** (used by SCTP).

MODEL QUESTION (6 Marks): Explain the establishment of a TCP connection using the three-way handshake. Justify the statement that a SYN segment consumes one sequence number.

Data Transfer

After connection, data and acknowledgments flow in **both directions**. Data going in the same direction as an ACK are carried in the **same segment** (**piggybacking**).

Example: the client sends 2,000 bytes in two segments (bytes 8001–9000 and 9001–10000) with the **PSH** flag set. The server sends 2,000 bytes in one segment (15001–17000), also carrying ack 10001. The client sends a final ACK-only segment (ack 17001) because it has no more data.



About Figure: After connection establishment, client and server exchange data segments with seq and ack numbers and the PSH and ACK flags. The last segment carries only an acknowledgment.

Pushing data

- Normally the sending TCP buffers data and chooses the segment size, and the receiving TCP delivers when convenient. This flexibility improves efficiency.
- **Interactive** applications cannot accept such delay. The application can request a **push**: the sending TCP **must not wait** for the window to fill. It sends a segment immediately with the **PSH** bit set, telling the receiving TCP to **deliver to the application as soon as possible**.

Urgent data

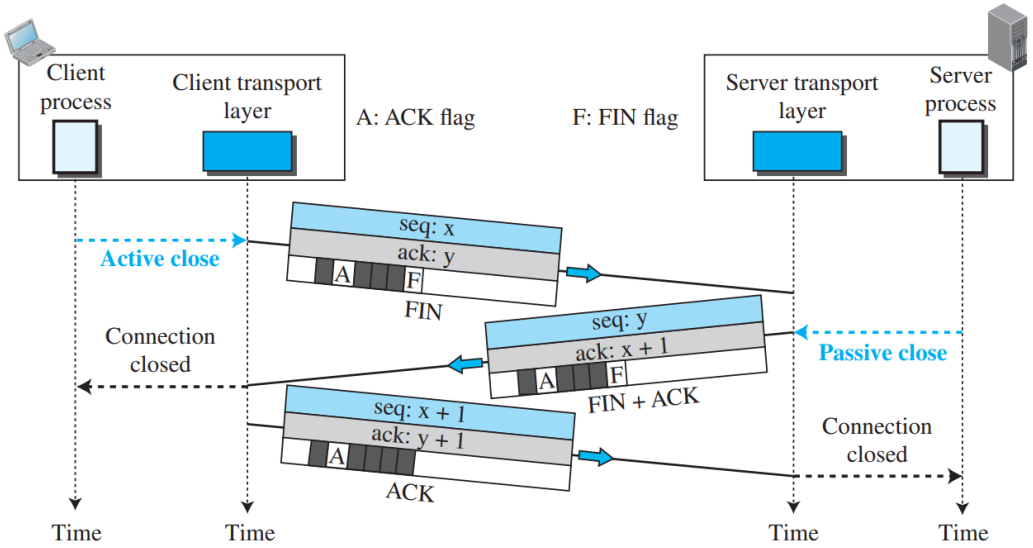
- TCP is stream-oriented, but sometimes some bytes need **special treatment** by the receiving application.
- The segment is sent with the **URG** bit set. The urgent data is placed at the **beginning** of the segment. The **urgent pointer** gives the end of the urgent data (last urgent byte).
- *Example:* sequence number 15000 and urgent pointer 200 means urgent bytes are 15000 to 15200.
- TCP urgent data is **neither a priority service nor out-of-band data**. The receiving TCP delivers bytes in order, but informs the application where the urgent data begins and ends. The application decides what to do.

Connection Termination

Either party can close the connection, usually the client. Two options are available.

Three-way handshaking

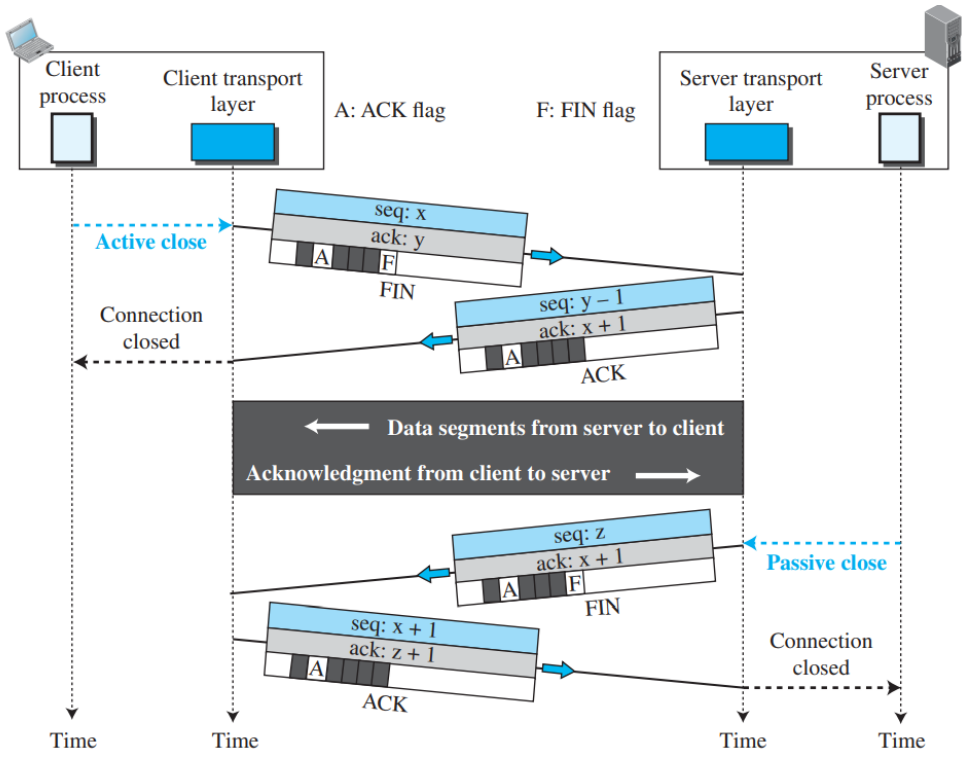
1. **FIN segment** from the client (FIN flag set). It may carry the last data. If it carries no data, it consumes **one sequence number**.
2. **FIN + ACK segment** from the server: acknowledges the client's FIN and announces closing in the other direction. It may carry last data. With no data it consumes one sequence number.
3. **ACK segment** from the client, acknowledging the server's FIN (ackNo = server's seq + 1). It carries no data and consumes **no sequence number**.



About Figure: A time line of FIN, then FIN + ACK, then ACK between client and server, with active close at the client, passive close at the server, and both ends closed.

Half-close

- One end can **stop sending data while still receiving**.
- Example: a client sends data to a server to be **sorted**. The server needs all the data before sorting. So the client, after sending all data, **half-closes** (sends FIN) in the client-to-server direction. The server ACKs. The server-to-client direction stays open to return the **sorted data**. When the server finishes, it sends its FIN, and the client sends the final ACK.
- After the half-close, data travels only from server to client, and ACKs from client to server.

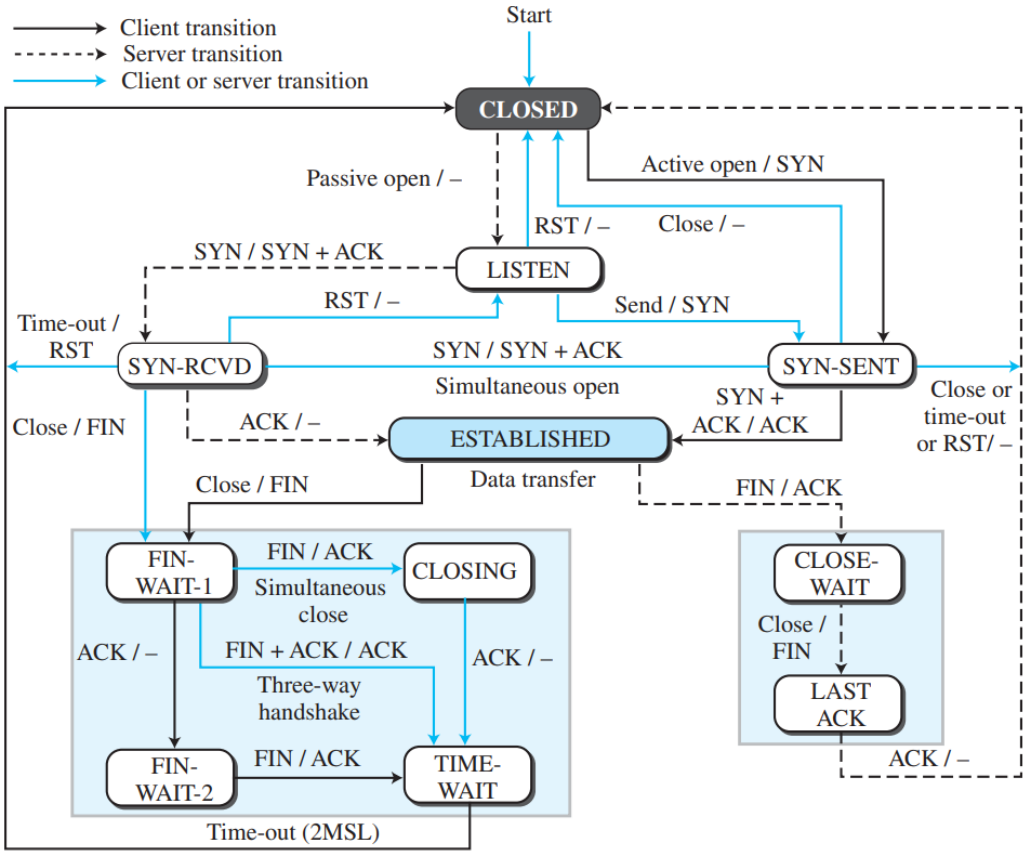


About Figure: The client sends FIN and the server replies with ACK, then the server continues sending data segments to the client (acknowledged by the client). Finally the server sends FIN and the client sends ACK.

Connection reset: TCP at one end may **deny a connection request, abort an existing connection or terminate an idle connection**. All are done with the **RST** flag.

State Transition Diagram

TCP is specified as a **finite state machine (FSM)**. Each arrow carries input / output (what TCP receives and what it sends). The ESTABLISHED state is really two sets of states (client and server) used for flow and error control.



About Figure: States are rounded rectangles joined by directed lines labelled input/output. Solid lines show the client's usual path, dotted lines the server's, and coloured lines special cases such as simultaneous open and close.

State	Description
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN + ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for the application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

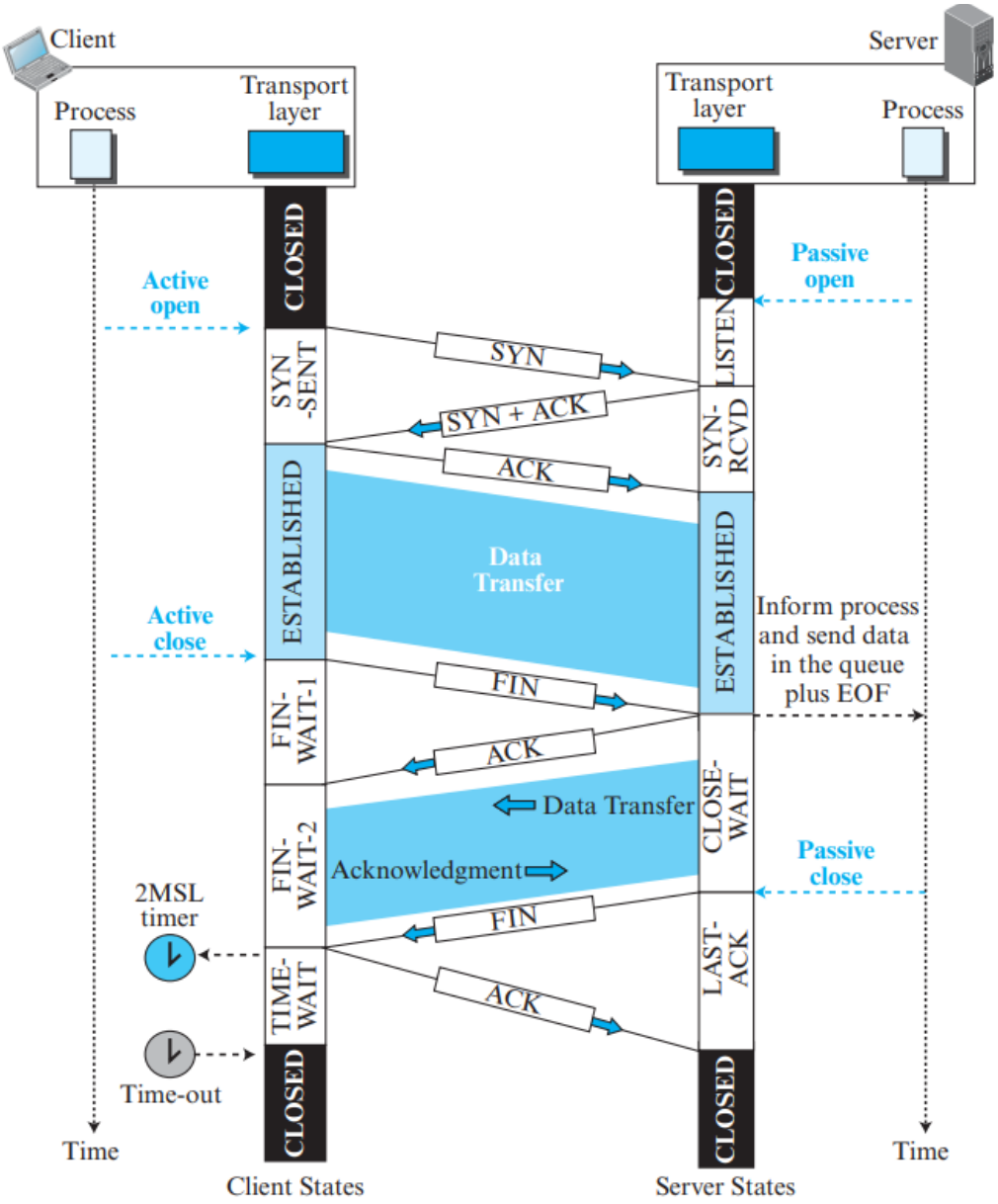
Common scenario with half-close

Client

1. Application issues an **active open**: send SYN → **SYN-SENT**.
2. Receives SYN + ACK, sends ACK → **ESTABLISHED**. Data is transferred and acknowledged.
3. No more data: **active close**, send FIN → **FIN-WAIT-1**.
4. Receives ACK → **FIN-WAIT-2**.
5. Receives FIN, sends ACK → **TIME-WAIT**. Stays for **2 MSL** seconds, then → **CLOSED**.

Server

1. Application issues a **passive open** → **LISTEN**, waiting for a SYN.
2. Receives SYN, sends SYN + ACK → **SYN-RCVD**.
3. Receives ACK → **ESTABLISHED**. Data transfer takes place.
4. Receives the client's FIN. It sends all queued data plus an EOF marker to its process, sends ACK → **CLOSE-WAIT** (it postpones its own FIN until the process issues a **passive close**).
5. Process closes: sends FIN → **LAST-ACK**.
6. Receives the final ACK → **CLOSED**.



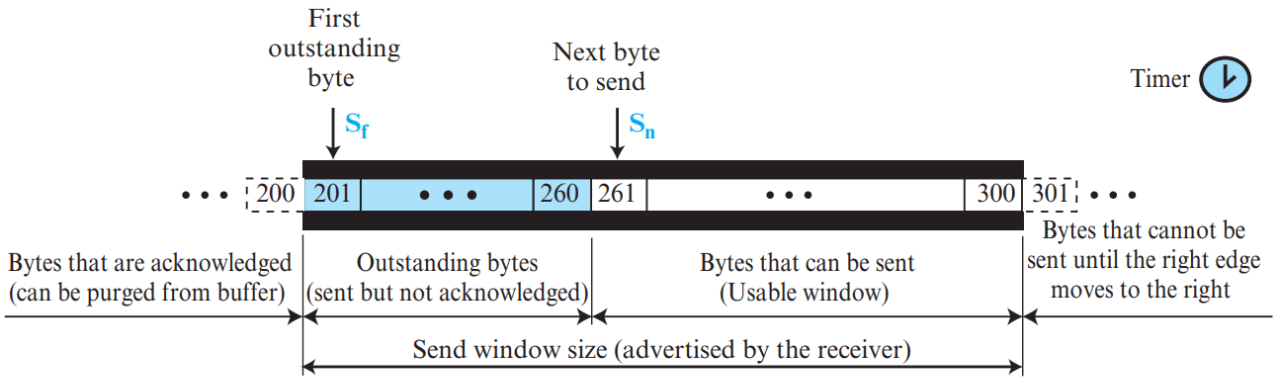
About Figure: A time line of client states (CLOSED, SYN-SENT, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, TIME-WAIT, CLOSED) against server states (CLOSED, LISTEN, SYN-RCVD, ESTABLISHED, CLOSE-WAIT, LAST-ACK, CLOSED) with the SYN, ACK and FIN segments between them.

Windows in TCP

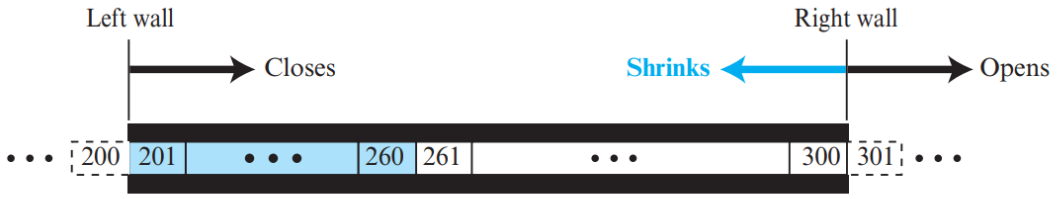
TCP uses **two windows (send and receive) for each direction** of data transfer. For simplicity, communication is treated as unidirectional.

Send window

- Window size is in **bytes** (in SR it is in packets).
- The size is dictated by the **receiver** (flow control) and by **network congestion** (congestion control).
- TCP uses **one timer** (SR theoretically uses one per packet).
- It **opens, closes or shrinks**.



a. Send window



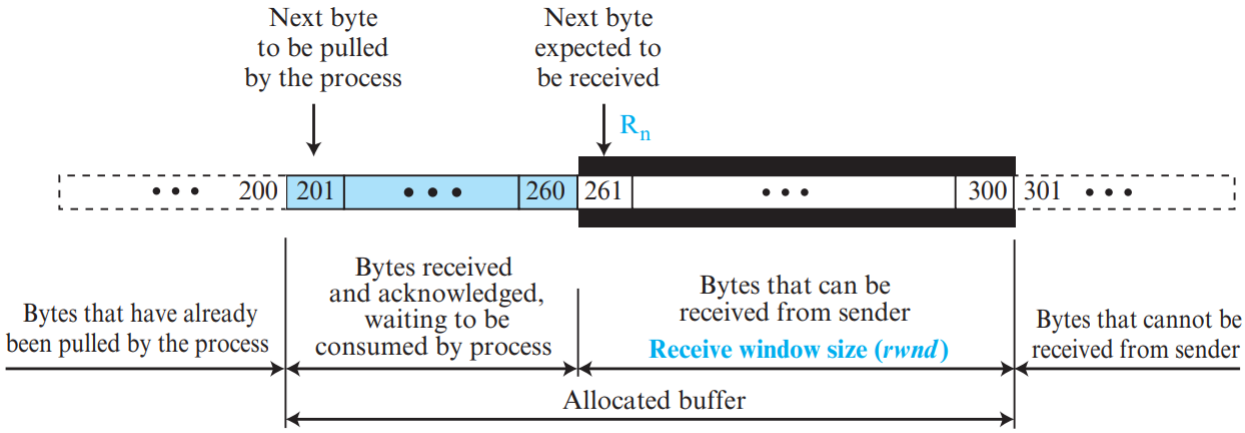
b. Opening, closing, and shrinking send window

About Figure: The window has a left wall and a right wall. Bytes to the left are acknowledged and purged. Inside the window are outstanding bytes and the usable window of bytes that can be sent. Bytes beyond the right wall cannot be sent until the wall moves right.

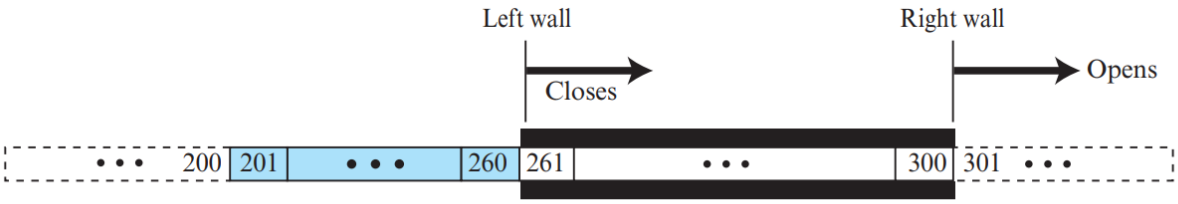
Receive window

- Size is **rwnd = buffer size − number of bytes waiting to be pulled by the process**.
- TCP lets the process **pull data at its own pace**, so some of the buffer holds received, acknowledged bytes waiting to be read. Hence rwnd ≤ buffer size.
- The receive window should **never shrink**.
- The main acknowledgment is **cumulative** (next expected byte), so TCP looks like GBN. Newer versions also use **selective** acknowledgments (SACK).

Receive window size, rwnd = buffer size − number of bytes waiting to be pulled



a. Receive window and allocated buffer



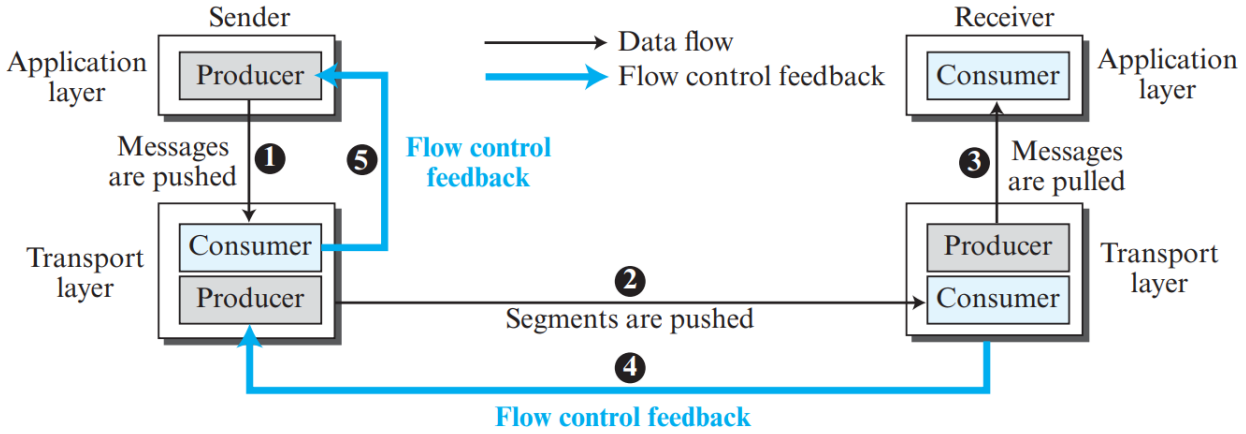
b. Opening and closing of receive window

About Figure: The allocated buffer holds bytes already pulled, bytes received and waiting for the process, and the receive window (rwnd) of bytes that can still be received. Its left wall closes as bytes arrive and its right wall opens as bytes are pulled.

Flow Control

TCP **separates flow control from error control**. Here the channel is assumed error-free.

- Data travel: sending process → sending TCP → receiving TCP → receiving process (paths 1, 2, 3).
- Feedback travels: receiving TCP → sending TCP (path 4), and sending TCP → sending process (path 5).
- Path 5 is achieved by the sending TCP simply **rejecting data when its window is full**.
- There is normally no feedback from the receiving process to the receiving TCP; the process pulls when ready.
- So the discussion concentrates on **path 4: rwnd advertised by the receiving TCP**.



About Figure: Numbered arrows 1 to 3 show data flow downward at the sender, across, and upward at the receiver. Arrows 4 and 5 show flow-control feedback from the receiving TCP to the sending TCP and from the sending TCP to the sending process.

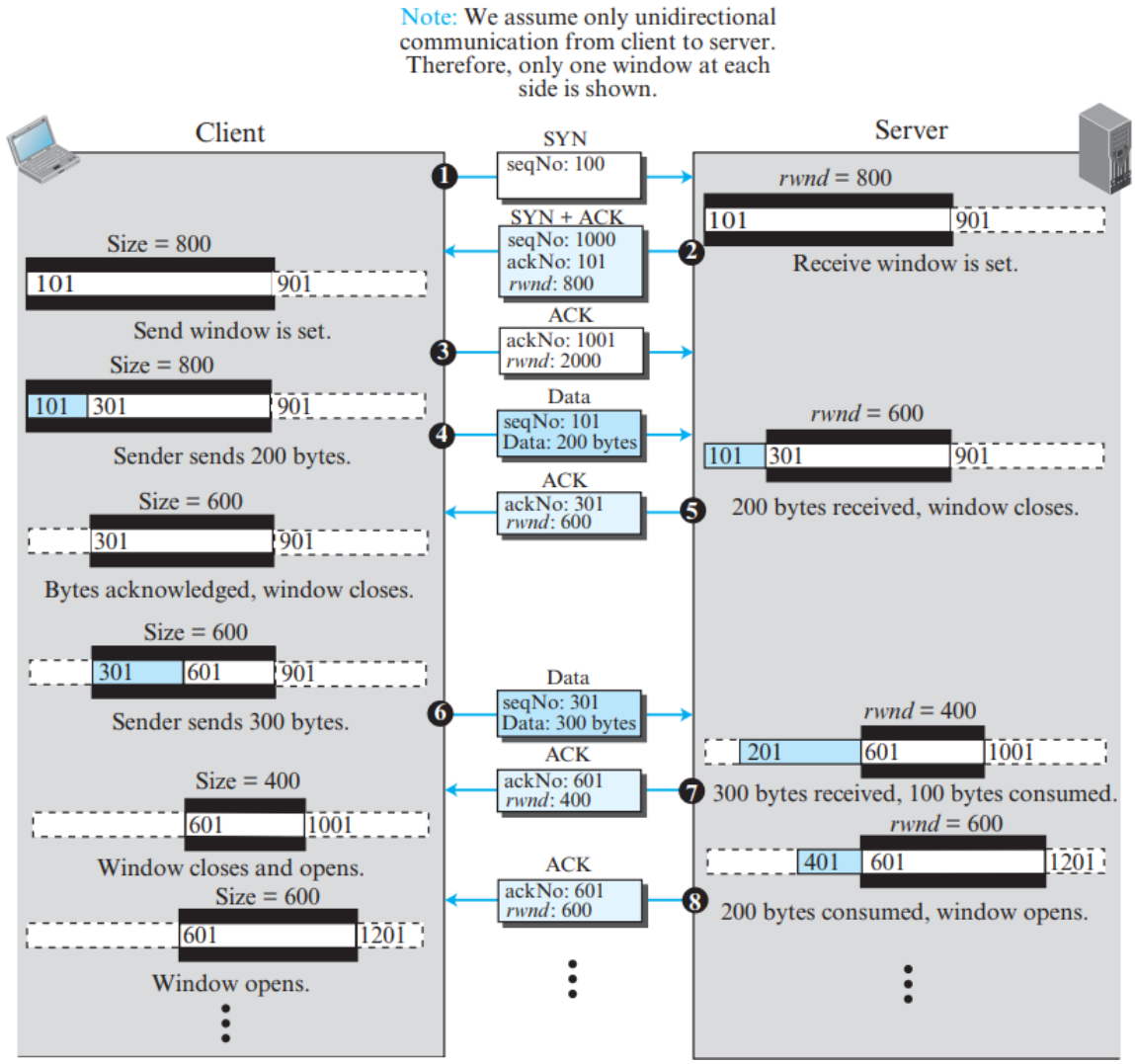
Opening and closing windows

- **Receive window closes** (left wall moves right) when more bytes arrive. It **opens** (right wall moves right) when the process pulls bytes.
- **Send window closes** (left wall moves right) when a new ACK arrives. It **opens** (right wall moves right) when the advertised rwnd allows it, that is when **new ackNo + new rwnd > last ackNo + last rwnd**.
- The send window **shrinks** if this condition does not hold.

A scenario (unidirectional, client to server)

Segment	Direction	Content	Effect
1	Client → server	SYN, seqNo 100	Server allocates a buffer of 800, sets rwnd = 800. Next byte expected: 101
2	Server → client	SYN + ACK, ackNo 101, rwnd 800	Client sets its send window to 800

Segment	Direction	Content	Effect
3	Client → server	ACK (client's rwnd 2000 not shown as communication is one-way)	Connection open
4	Client → server	Data seqNo 101, 200 bytes	Server stores them; receive window closes; next expected 301
5	Server → client	ackNo 301, rwnd 600	Client purges acknowledged bytes; window size falls to 600 (the buffer holds 800 but the receiver does not let the window open)
6	Client → server	Data seqNo 301, 300 bytes	Server stores them. Process pulls 100 bytes, so window closes 300 and opens 100: size = 400
7	Server → client	ackNo 601, rwnd 400	Client send window: closes 300, opens 100; size 400
8	Server → client	ackNo 601, rwnd 600 (process pulled 200 more)	Client window opens by 200; size 600



About Figure: Eight segments between client and server, with the client's send window and the server's receive window drawn after each step. Window sizes change with the rwnd values 800, 600, 400 and 600.

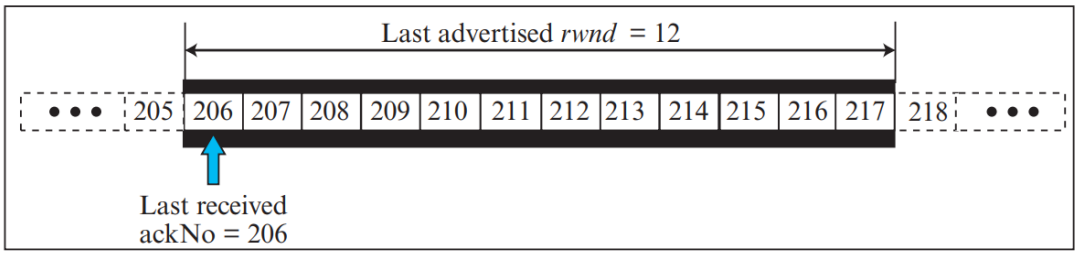
Shrinking of windows

- The receive window **cannot shrink**. The send window can shrink only if the receiver's rwnd moves its right wall left, which some implementations do not allow.
- To prevent shrinking, the receiver must keep:

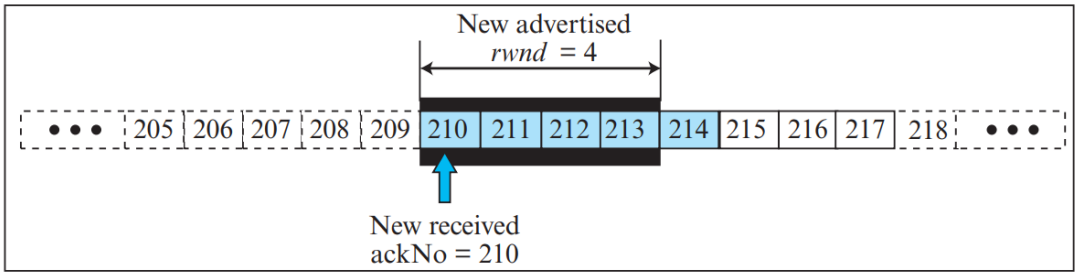
Condition to prevent shrinking of the send window, new ackNo + new rwnd ≥ last ackNo + last rwnd

The left side is the new position of the right wall; the right side is the old position. The right wall must not move left. All calculations are modulo 2^32, and the rule is valid if Sf < Sn.

Example: Last advertisement: ackNo = 206, rwnd = 12 (right wall at 218). The sender has sent bytes 206 to 214. Bytes 206–209 are acknowledged. New advertisement: ackNo = 210, rwnd = 4, so 210 + 4 = 214 < 206 + 12 = 218. The window shrinks and byte 214, **already sent**, falls outside the window. The receiver therefore must **postpone its feedback** until enough bytes are consumed to meet the rule.



a. The window after the last advertisement



b. The window after the new advertisement; window has shrunk

About Figure: (a) The window after the last advertisement (ackNo 206, rwnd 12). (b) The window after the new advertisement (ackNo 210, rwnd 4), where it has shrunk and byte 214 lies outside.

Window shutdown

The receiver may temporarily shut down the window by sending **rwnd = 0**. The sender stops sending until a new advertisement arrives. Even then it can send a **1-byte probe** segment to prevent deadlock (see the persistence timer).

Silly window syndrome

- If TCP sends **very small segments**, efficiency is poor. For example, 1 byte of data in a 41-byte datagram (20 bytes TCP + 20 bytes IP) has overhead 41/1.
- It occurs when the **sending application creates data slowly**, the **receiving application consumes data slowly**, or both.

Created by	Cause	Solution
Sender	Application writes 1 byte at a time	**Nagle's algorithm**
Receiver	Application reads 1 byte at a time, so the receiver advertises tiny windows	**Clark's solution** or **delayed acknowledgment**

Nagle's algorithm

1. The sending TCP sends the **first piece of data** it gets, even 1 byte.
2. After that, it **accumulates data** in the output buffer and waits until **either an ACK arrives or enough data fills a maximum-size segment**, then sends.
3. Step 2 is repeated for the rest of the transmission.

- It adapts to both speeds: if the application is faster than the network, segments are large; if slower, segments are smaller.

Clark's solution: send an ACK as soon as data arrives, but **announce a window of zero** until there is room for a **maximum-size segment** or **at least half of the receive buffer is empty**.

Delayed acknowledgment: the receiver does not acknowledge immediately; it waits for a decent amount of buffer space. This also **reduces traffic**. Drawback: the sender may unnecessarily retransmit. So the delay must be **not more than 500 ms**.

Error Control

TCP is reliable. The stream is delivered **in order, without error, with nothing lost or duplicated**. Error control detects and resends corrupted segments, resends lost segments, stores out-of-order segments until missing ones arrive, and discards duplicates. It uses three tools: **checksum, acknowledgment and time-out**.

Checksum

A 16-bit checksum is **mandatory**. A segment with an invalid checksum is discarded and treated as lost.

Acknowledgment

TCP acknowledges data segments. Control segments with no data but consuming a sequence number are also acknowledged. **ACK segments are never acknowledged** and consume no sequence number.

Type	Meaning
Cumulative ACK	Original type. The receiver advertises the **next byte expected**, ignoring out-of-order segments. "Positive": no feedback for discarded, lost or duplicate segments. Uses the 32-bit ACK field, valid when ACK = 1
Selective ACK (SACK)	Does not replace ACK but reports **extra information**: blocks of bytes received out of order and blocks received more than once. Implemented as a **TCP option**

Rules for generating acknowledgments

1. **Piggyback:** when end A sends data to B, it includes the ACK for the next expected byte. This reduces segments.
2. **Delay:** if the receiver has no data to send and an in-order segment arrives with the previous one already acknowledged, delay the ACK until another segment arrives or **500 ms** pass.
3. **No more than two unacknowledged in-order segments:** when an expected segment arrives and the previous in-order segment is still unacknowledged, send an ACK **immediately**. This avoids unnecessary retransmission.
4. **Out of order, higher than expected:** send an ACK **immediately** announcing the next expected segment. This leads to **fast retransmission**.
5. **Missing segment arrives:** send an ACK announcing the next expected number.
6. **Duplicate segment:** discard it, but send an ACK immediately with the next in-order segment expected. This helps if an ACK was lost.

Retransmission

Sent segments are kept in a queue until acknowledged. A segment is retransmitted in two cases.

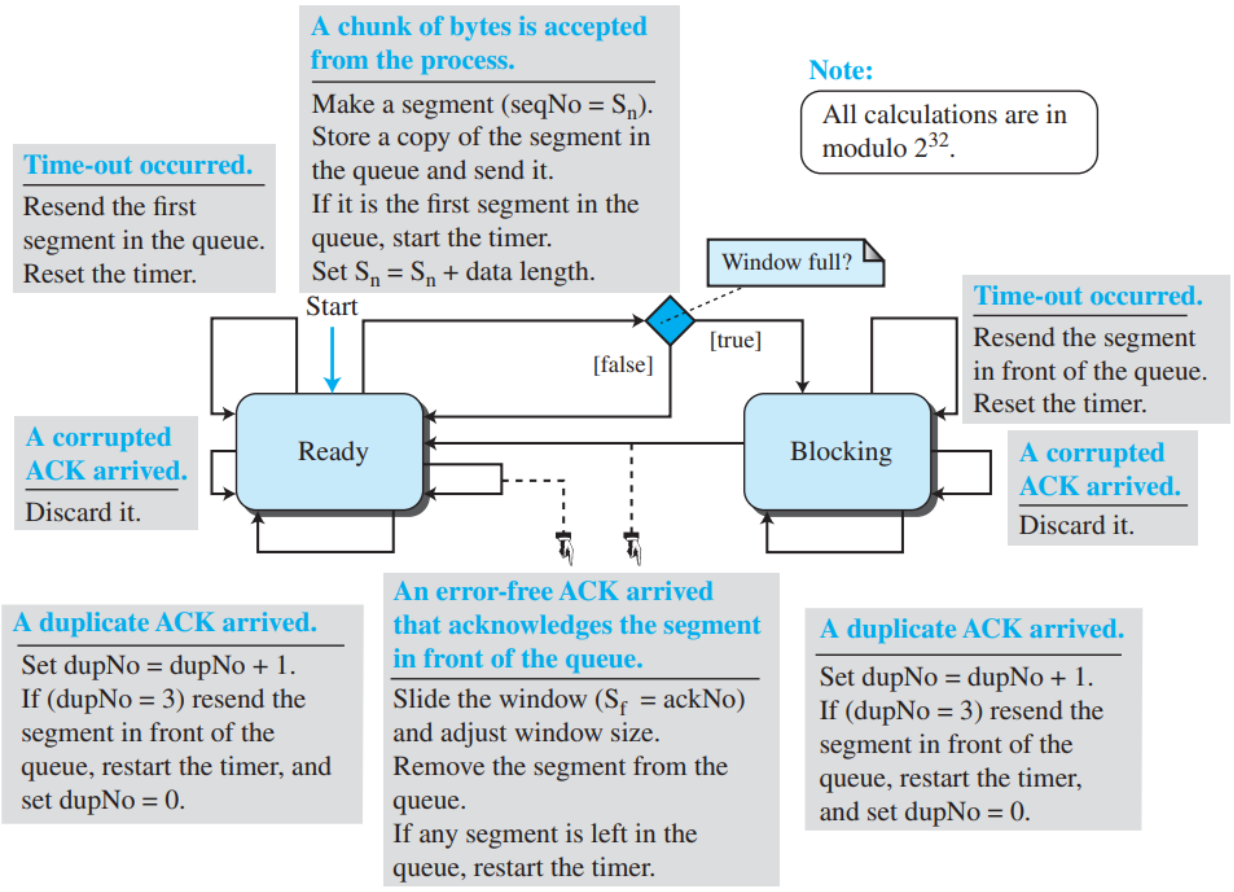
Case	Rule
After RTO	TCP keeps **one retransmission time-out (RTO)** per connection. When it expires, TCP resends the segment at the **front of the queue** (smallest sequence number) and restarts the timer. RTO is **dynamic**, based on the round-trip time
After three duplicate ACKs (fast retransmission)	If an original ACK plus **three identical duplicates** arrive for a segment, the next (missing) segment is resent **without waiting for time-out**

Out-of-order segments

TCP **stores** them temporarily and flags them. They are **never delivered** to the process out of order. TCP is best modelled as **Selective-Repeat** (and with SACK it is closest to SR; with plain cumulative ACKs it looks like GBN).

Sender-side FSM (simplified, unidirectional)

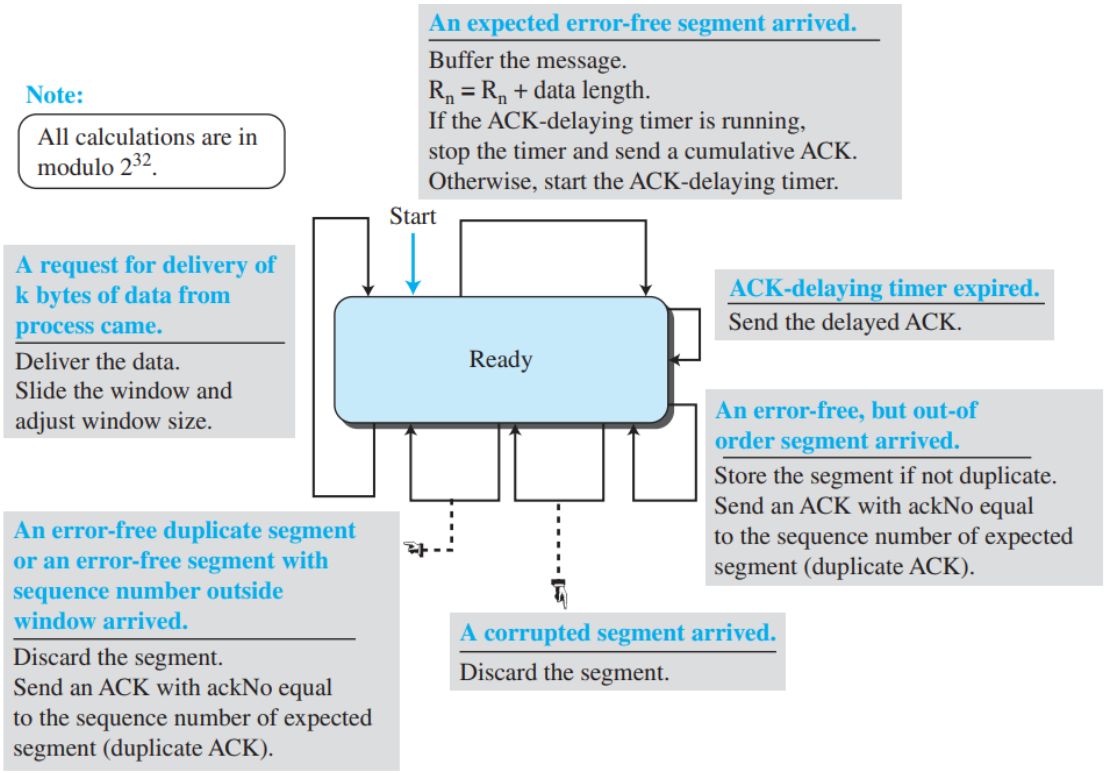
- **Ready:** a chunk of bytes accepted from the process → make segment (seqNo = Sn), store a copy in the queue, send it, start the timer if it is the first in the queue, Sn = Sn + data length. If the window is full, go to **Blocking**.
- **Error-free ACK** acknowledging the segment at the front → slide the window (Sf = ackNo), adjust window size, remove the segment from the queue, restart the timer if segments remain.
- **Duplicate ACK** → dupNo = dupNo + 1. If dupNo = 3, resend the front segment, restart the timer, dupNo = 0.
- **Corrupted ACK** → discard.
- **Time-out** → resend the first segment in the queue, reset the timer.



About Figure: Ready and blocking states with events: a chunk accepted, error-free ACK, duplicate ACK (counting to three), corrupted ACK and time-out, each with its action.

Receiver-side FSM (simplified)

- **Expected error-free segment arrives** → buffer the message, Rn = Rn + data length. If the ACK-delaying timer is running, stop it and send a cumulative ACK; otherwise start the ACK-delaying timer.
- **ACK-delaying timer expires** → send the delayed ACK.
- **Error-free but out-of-order segment** → store it if not a duplicate; send an ACK with ackNo = expected sequence number (**duplicate ACK**).
- **Duplicate or out-of-window segment** → discard, send ACK with expected number.
- **Corrupted segment** → discard.
- **Request from the process to deliver k bytes** → deliver, slide the window, adjust size.

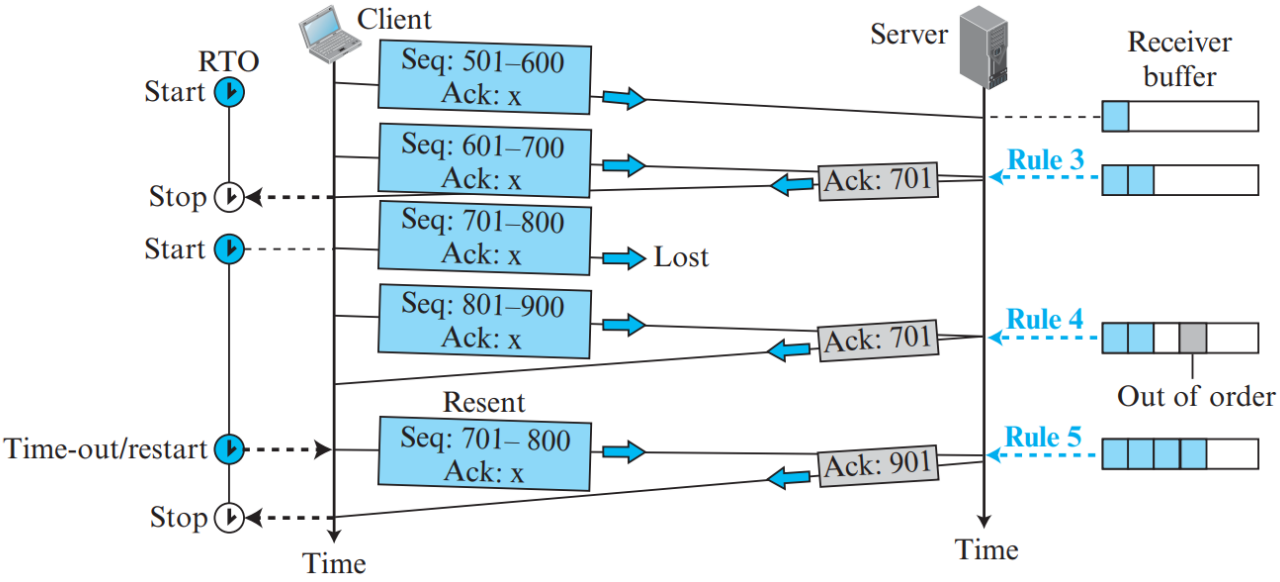


About Figure: A single ready state with the events: expected segment, delayed-ACK timer expiry, out-of-order segment, duplicate segment, corrupted segment, and delivery request from the process.

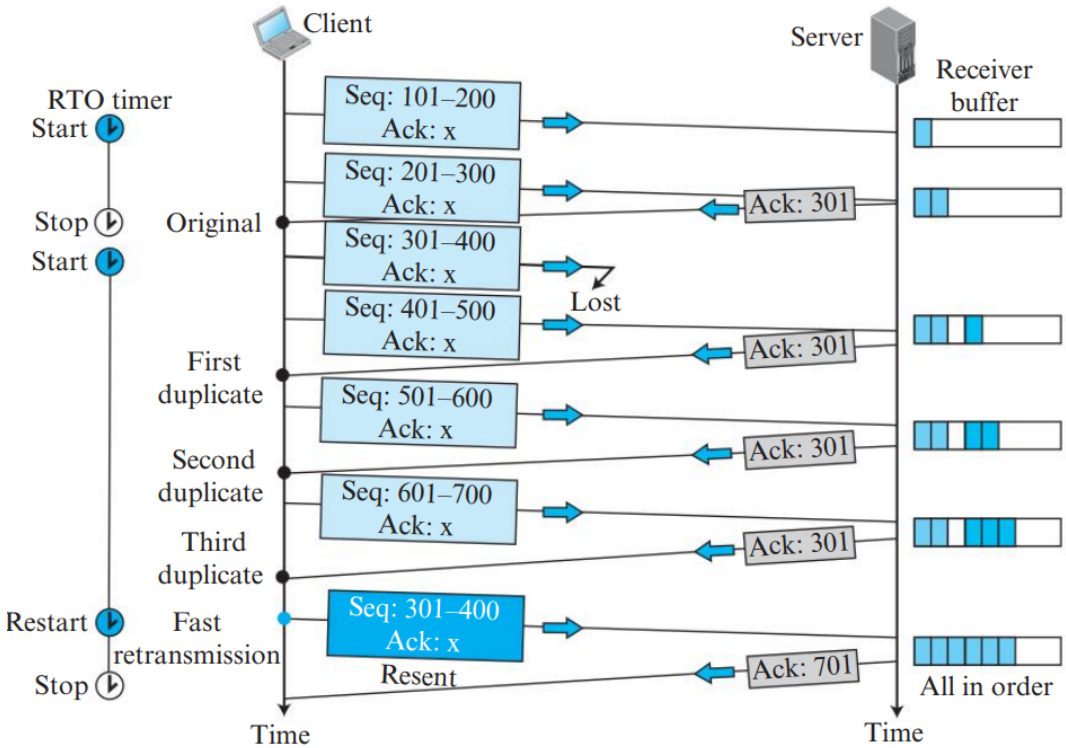
Scenarios

Scenario	What happens
Normal operation	Bidirectional transfer. Data segments piggyback ACKs (rule 1). With nothing to send, the client delays its ACK 500 ms (rule 2). A third segment triggers an immediate ACK (rule 3)
Lost segment	Segments 1 and 2 are acknowledged. **Segment 3 is lost.** Segment 4 arrives out of order: it is stored, leaving a gap, and an ACK with the next expected byte is sent at once (rule 4). It is never delivered until the gap is filled. On RTO the sender resends segment 3, and an ACK follows (rule 5). A corrupted segment is treated like a lost one
Fast retransmission	Same as above but with a **large RTO**. Each later segment triggers an ACK (rule 4). The sender gets the original ACK plus **three duplicates**, so it resends the missing segment immediately and restarts the timer
Delayed segment	IP segments take different routes. A delayed segment may time out and be resent. If the delayed copy arrives later, it is a **duplicate** and is discarded
Duplicate segment	The destination discards a segment whose sequence number matches one already stored and sends an ACK with the expected number

Scenario	What happens
Lost ACK corrected automatically	Cumulative ACKs mean the **next ACK carries the lost ACK's information**, so the loss may not even be noticed
Lost ACK corrected by resending a segment	If there is no next ACK (or it is delayed), the RTO fires and a segment is resent. The receiver **discards the duplicate and resends the last ACK** (rule 6). Only one segment is resent, although two were unacknowledged, because the ACK is cumulative
Deadlock by lost ACK	The receiver sends an ACK with **rwnd = 0**, later sends a nonzero rwnd in an ACK-only segment. If this is lost, the sender waits for it and the receiver waits for data. No retransmission timer applies to an ACK-only segment. The **persistence timer** solves this



About Figure (ABOVE): The sender sends segments 501–600 and 601–700 (ACKed), then 701–800 is lost. Segment 801–900 arrives out of order and triggers ACK 701. After RTO the lost segment is resent and ACK 901 is returned.



About Figure (ABOVE): After one segment is lost, later segments produce an original ACK and three duplicate ACKs (all 301). The sender resends segment 301–400 before the RTO expires.

TCP Congestion Control

The receive window prevents **end** congestion (no overflow of the receiver). But **routers in the middle** may be overwhelmed by data from several senders. Lost segments are resent, which worsens congestion and can collapse communication. IP has no congestion control, so **TCP must handle it**. TCP must not be too aggressive (it would hurt itself) or too conservative (wasted bandwidth).

Congestion window (cwnd)

A variable whose size depends on congestion in the network. The actual send window is the minimum of rwnd and cwnd.

$$\text{Actual window size, window size} = \min(\text{rwnd}, \text{cwnd})$$

Where rwnd (set by the receiver) relates to congestion at the **end**, and cwnd relates to congestion in the **middle** (network).

Congestion detection

The sender uses only the ACK feedback. Two events signal congestion:

Event	Meaning
Time-out	No ACK before the timer expires: segments assumed lost due to congestion. **Strong congestion**
Three duplicate ACKs (four ACKs with the same number)	One segment is missing but later ones arrived. **Weak (light) congestion**

Taho TCP treats both events the same. Reno TCP treats them differently.

Congestion policies

Algorithm	Growth	Rule
Slow start	**Exponential**	cwnd starts at **1 MSS**; each ACK adds 1 MSS. Doubles every RTT. Stops at **ssthresh**
Congestion avoidance	**Additive**	Starts when cwnd reaches ssthresh. cwnd increases by 1 MSS after the **whole window is acknowledged** (each ACK adds 1/cwnd)
Fast recovery	Additive (optional)	Starts on three duplicate ACKs. Each duplicate ACK adds to cwnd

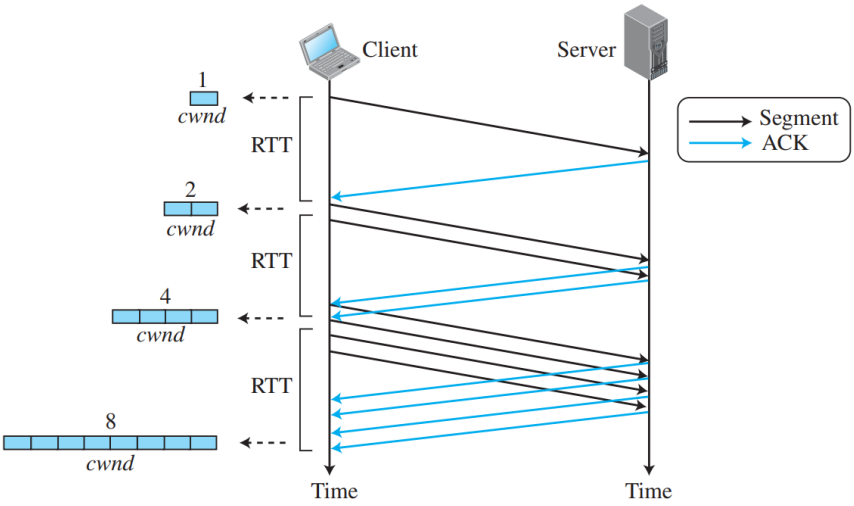
$$\text{Slow start (per ACK), cwnd} = \text{cwnd} + 1$$

$$\text{Slow start after i round-trip times, cwnd} = 2^{i}$$

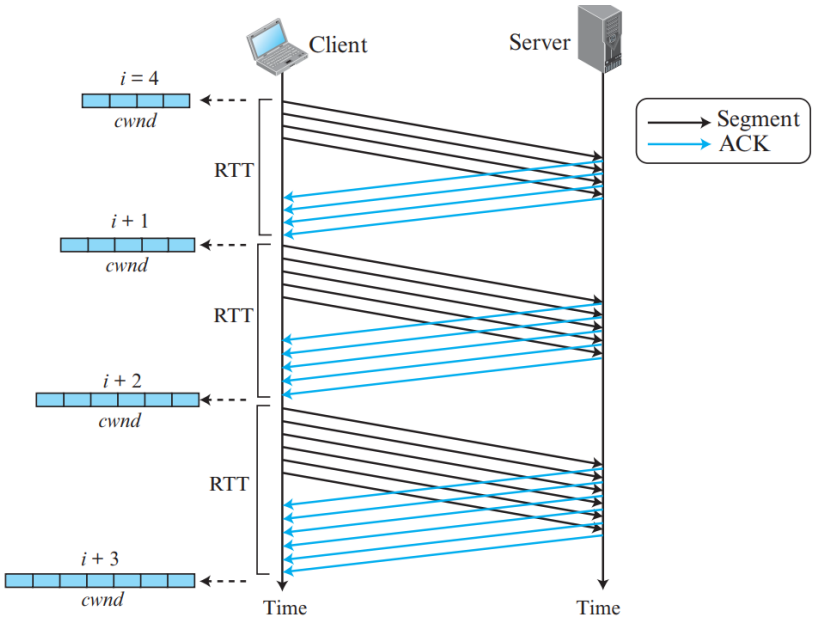
Where i = number of RTTs since the start (cwnd = 1 initially): 1 → 2 → 4 → 8. With delayed ACKs the growth is slower (power of 1.5 instead of 2).

$$\text{Congestion avoidance (per ACK), cwnd} = \text{cwnd} + \frac{1}{\text{cwnd}}$$

$$\text{Congestion avoidance after i RTTs , cwnd} = \text{cwnd}_0 + i$$



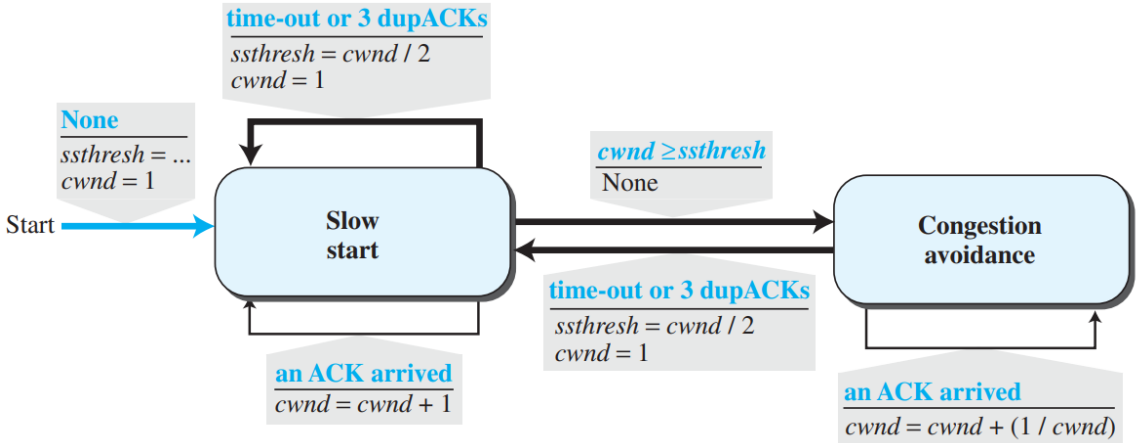
About Figure (ABOVE): Each RTT, the congestion window doubles: 1 segment, then 2, 4 and 8, as ACKs return to the client.



About Figure (ABOVE): Starting from cwnd = i, the window grows by one segment per RTT: i, i + 1, i + 2, i + 3.

Taho TCP (early version; slow start + congestion avoidance only)

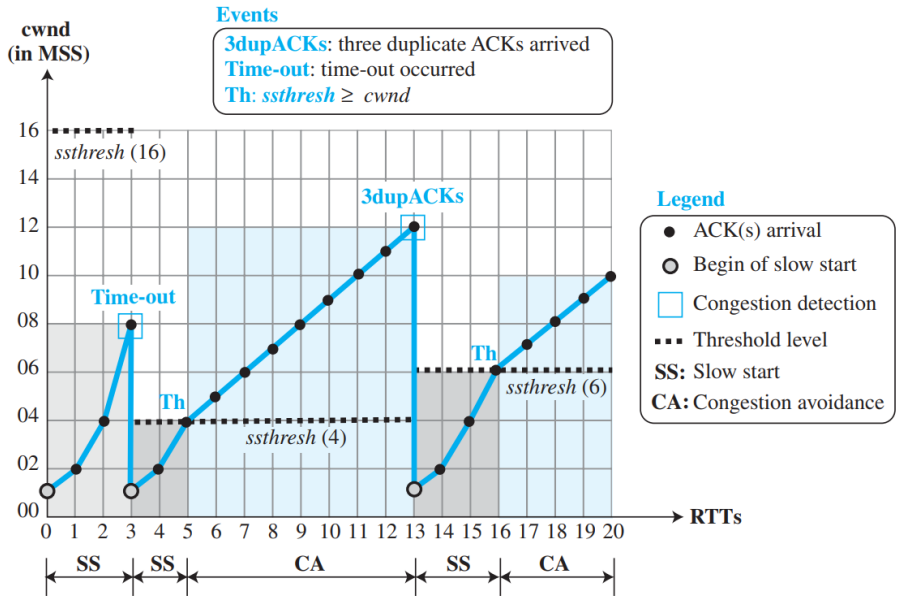
- Starts with slow start: **cwnd = 1 MSS**, ssthresh set to an agreed value.
- When cwnd ≥ ssthresh, it moves to **congestion avoidance**.
- On **time-out or three duplicate ACKs** (treated the same): **ssthresh = cwnd / 2, cwnd = 1**, and **restart slow start**.
- ssthresh is adjusted every time (not necessarily lower; for example ssthresh 8 and cwnd 20 gives new ssthresh 10).



About Figure: Two states, slow start and congestion avoidance. cwnd ≥ ssthresh moves from slow start to congestion avoidance. A time-out or 3 dupACKs in either state sets ssthresh = cwnd/2 and cwnd = 1 and returns to slow start.

***Example (Taho).* ssthresh = 16 MSS, start in slow start.**

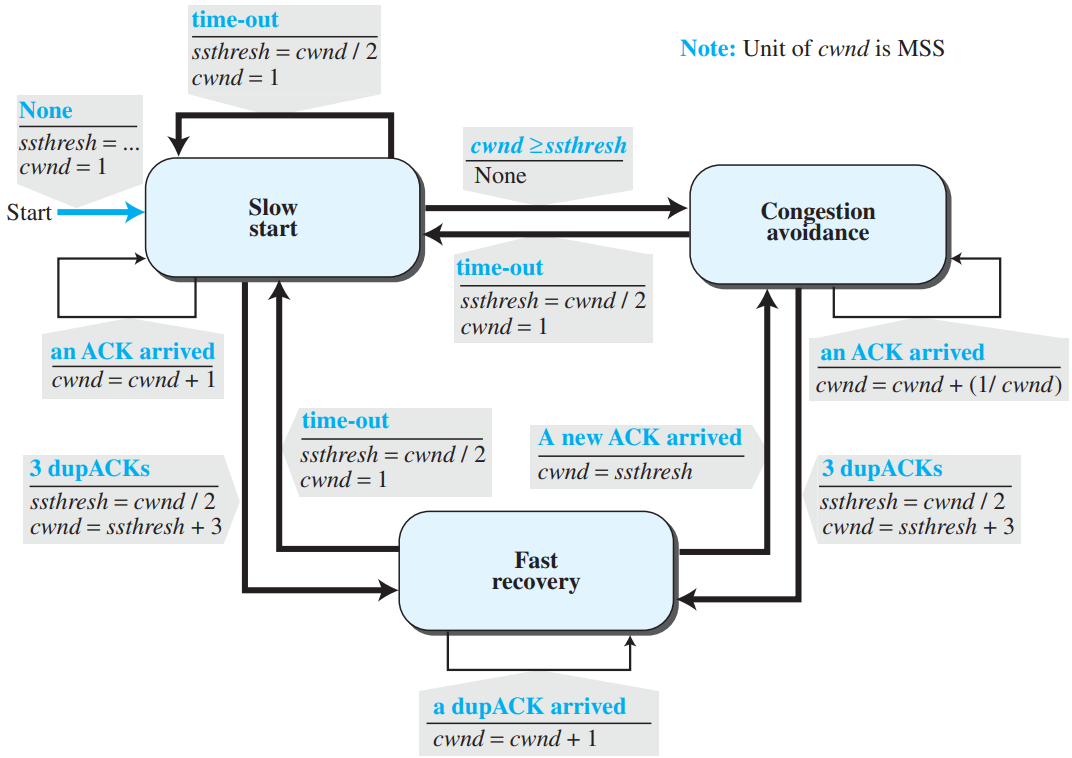
- A **time-out after RTT 3** (cwnd = 8): ssthresh = 4, new slow start from cwnd = 1.
- cwnd reaches 4, so **congestion avoidance** begins; cwnd grows to 12.
- **Three duplicate ACKs** at cwnd = 12: ssthresh = 6, new slow start.
- cwnd reaches 6, so congestion avoidance continues to the end (RTT 20).



About Figure: cwnd (in MSS) plotted against RTTs, with slow-start and congestion-avoidance phases, ssthresh levels (16, 4, 6) and the time-out and 3dupACKs events marked.

Reno TCP (adds fast recovery)

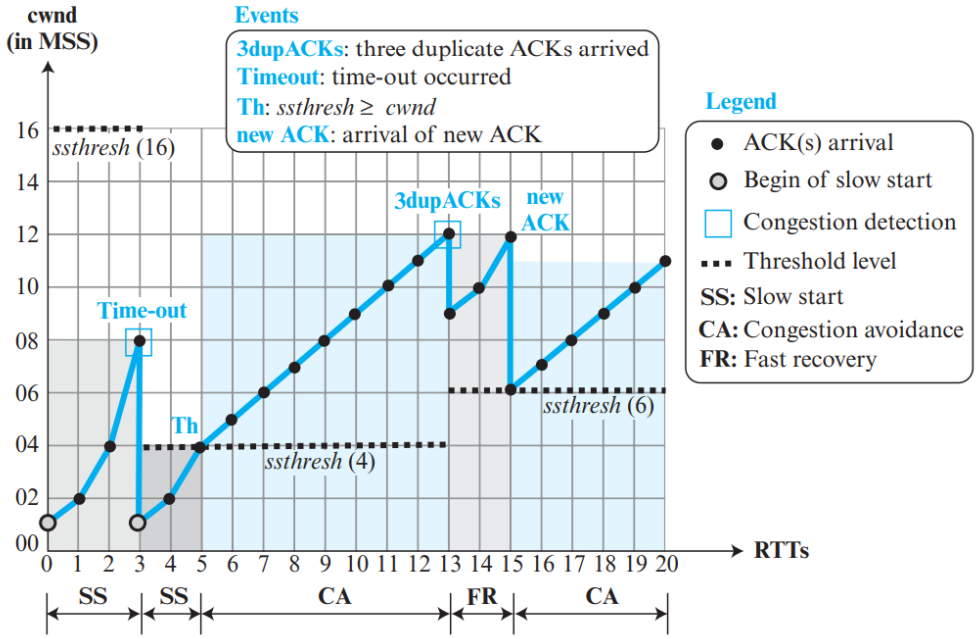
- **Time-out:** go to slow start (ssthresh = cwnd/2, cwnd = 1).
- **Three duplicate ACKs:** go to **fast recovery**: ssthresh = cwnd/2, **cwnd = ssthresh + 3**. In this state:
  - more duplicate ACKs arrive → stay, cwnd grows;
  - a **time-out** → slow start (real congestion);
  - a **new (non-duplicate) ACK → congestion avoidance**, with cwnd deflated to ssthresh.



About Figure: Three states, slow start, congestion avoidance and fast recovery, with the transitions on cwnd ≥ ssthresh, time-out, three dupACKs, a duplicate ACK in fast recovery, and a new ACK.

Example (Reno).

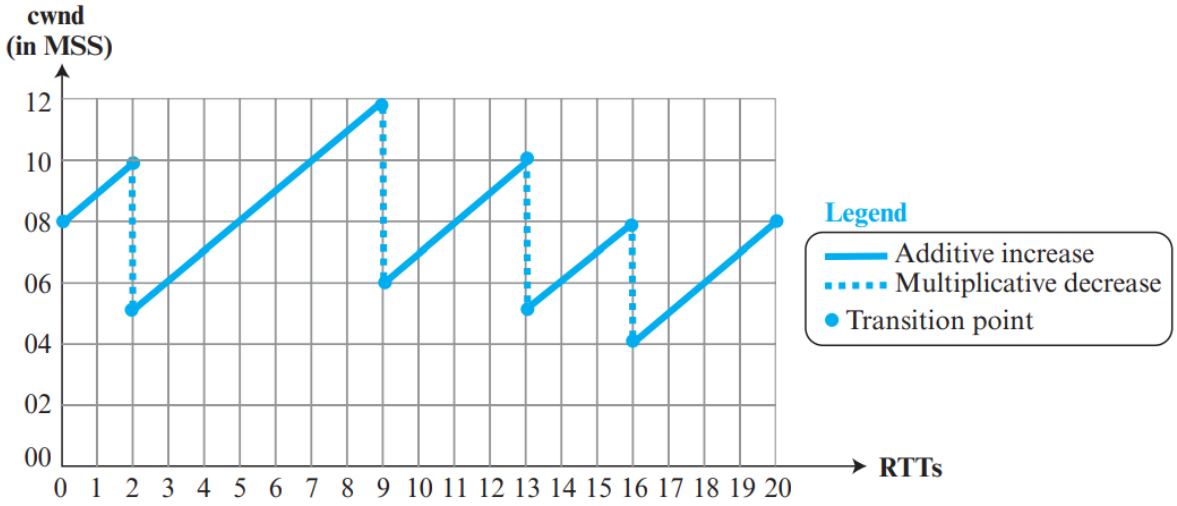
Same as previous Example until RTT 13, when **three duplicate ACKs** arrive. ssthresh = 6 (as in Taho), but cwnd = **ssthresh + 3 = 9** (not 1). Two more duplicate ACKs arrive (fast recovery). A **new ACK** arrives, so TCP deflates cwnd to 6 and moves to congestion avoidance.



About Figure: The same time line as for Taho, but after the 3dupACKs event cwnd jumps to 9 (fast recovery), then drops to 6 on the new ACK and continues additively.

NewReno TCP: an extra optimisation. On three duplicate ACKs it checks whether **more than one segment** in the window was lost. If a new ACK covers only part of the window, the next lost segment is also retransmitted.

Additive Increase, Multiplicative Decrease (AIMD): Reno is the most common version. Ignoring slow start and the short fast-recovery, cwnd increases by 1/cwnd per ACK (**additive increase**) and is **halved** at each congestion detection (**multiplicative decrease**). cwnd follows a **saw-tooth pattern**.



About Figure: cwnd rises linearly between congestion events and drops to half at each one, forming a saw-tooth.

TCP throughput

- With a flat cwnd: throughput = cwnd / RTT.
- With saw-teeth (maximum = twice the minimum), the average window is 0.75 Wmax.

$$\text{TCP throughput, Throughput} = \frac{0.75 \times W_{max}}{RTT}$$

Where W_max = average of the window sizes when congestion occurs and RTT = round-trip time.

Example: MSS = 10 KB, RTT = 100 ms, window sizes at congestion 10, 12, 10, 8, 8 MSS.

- $W_{max} = 9.6 \times 10 \times 1000 \times 8 = 768$ kbits
- $\text{Throughput} = \frac{0.75 \times 768}{0.1} = 5.76$ Mbps

TCP Timers

TCP uses at least four timers: **retransmission, persistence, keepalive and TIME-WAIT**.

Retransmission timer (one per connection) rules:

1. When the segment at the front of the queue is sent, start the timer.
2. When the timer expires, resend the front segment and restart the timer.
3. Cumulatively acknowledged segments are purged from the queue.
4. If the queue is empty, stop the timer; otherwise restart it.

Round-trip time (RTT)

- **Measured RTT (RTT_M):** time from sending a segment to its ACK (the ACK may cover several segments). **Only one RTT measurement can be in progress at a time.**
- **Smoothed RTT (RTT_S):** a weighted average, because RTT_M fluctuates.
- **RTT deviation (RTT_D):** the variation in RTT.

$$\text{Smoothed RTT, } RTT_S = (1-\alpha) \times RTT_S + \alpha \times RTT_M$$

Initially no value. After the first measurement $RTT_S = RTT_M$. Normally alpha = 1/8 (new RTT_S = 7/8 old + 1/8 measured).

$$\text{RTT deviation, } RTT_D = (1-\beta) \times RTT_D + \beta \times |RTT_S - RTT_M|$$

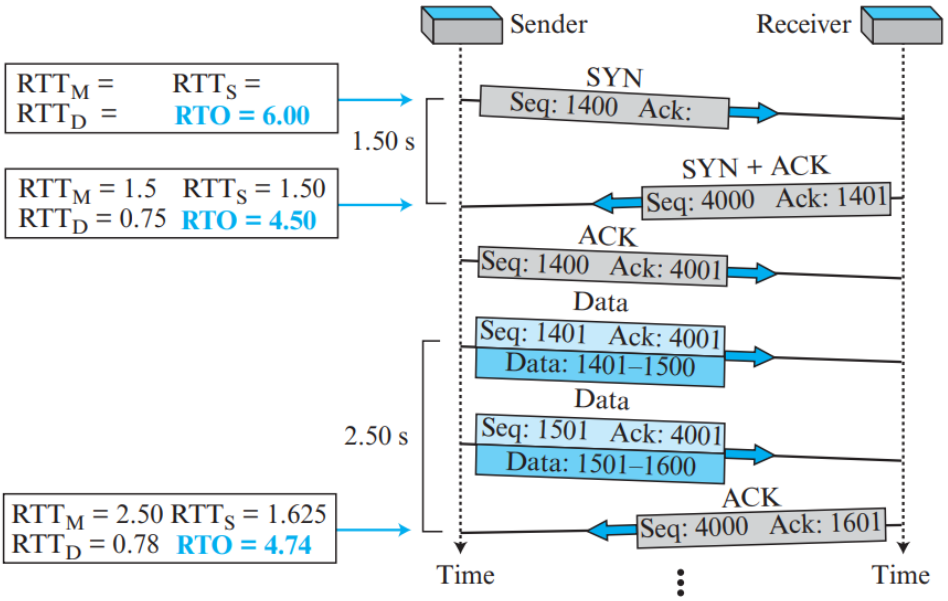
After the first measurement $RTT_D = RTT_M / 2$. Normally beta = 1/4.

$$\text{Retransmission time-out, } RTO = RTT_S + 4 \times RTT_D$$

The initial RTO is a fixed value (6 s in the example).

Example:

1. SYN sent: no RTT values; **RTO = 6.00 s**.
2. SYN + ACK arrives, RTT_M = 1.5 s: RTT_S = 1.5, RTT_D = 1.5/2 = 0.75, RTO = 1.5 + 4 × 0.75 = **4.5 s**.
3. First data segment starts a new measurement. The last ACK (cumulative, for both data segments) gives RTT_M = 2.5 s:
    - RTT_S = (7/8)(1.5) + (1/8)(2.5) = **1.625**
    - RTT_D = (3/4)(0.75) + (1/4)|1.625 − 2.5| = **0.78**
    - RTO = 1.625 + 4 × 0.78 = **4.74 s**



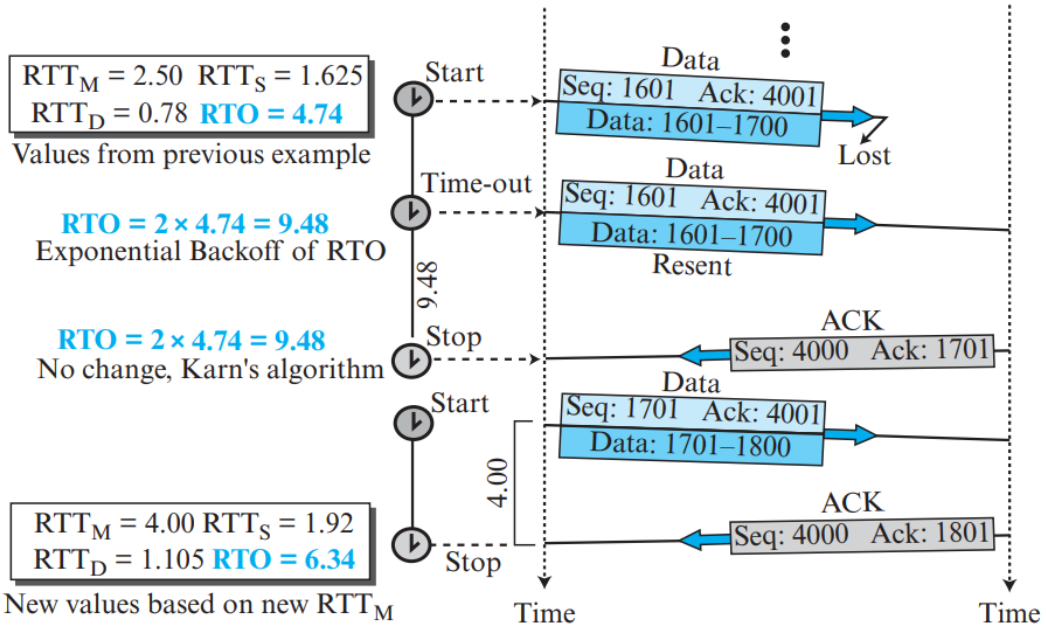
About Figure: A time line with the SYN, SYN + ACK, ACK and two data segments, listing RTT_M, RTT_S, RTT_D and RTO after each measurement (6.00, 4.50 and 4.74).

Karn's algorithm: If a segment is retransmitted, an arriving ACK is ambiguous (original or resent?). Karn's rule: **do not use the RTT of a retransmitted segment** to update RTT_S, and update only when a segment is sent and acknowledged **without retransmission**.

Exponential backoff: After a retransmission, the **RTO is doubled** each time (twice RTO for the first retransmission, four times for the second, and so on).

Example:

The first segment is lost; RTO (4.74 s) expires and the segment is resent with RTO = 2 × 4.74 = **9.48 s**. The ACK arrives before time-out. By Karn's algorithm, RTT_S and RTT_D are not recalculated now. For the next segment RTT_M = 4.00: RTT_S = (7/8)(1.625) + (1/8)(4.00) = **1.92**, RTT_D = (3/4)(0.78) + (1/4)|1.92 − 4.00| = **1.105**, RTO = 1.92 + 4 × 1.105 = **6.34 s**.



About Figure: A lost data segment, resend after time-out with RTO doubled to 9.48 s, no change by Karn's algorithm, then new RTT values (RTT_S 1.92, RTT_D 1.105, RTO 6.34) after the next segment.

Persistence timer (prevents the zero-window deadlock)

- When the sender receives an ACK with **window size 0**, it starts a persistence timer.
- When it expires, the sender sends a **probe** segment with **1 byte** of new data. The probe has a sequence number, but it is **never acknowledged** and is ignored in sequence numbering. It makes the receiver resend its ACK with the current window.
- The timer value starts at the retransmission time. If no response, another probe is sent and the timer is **doubled**, up to a threshold (usually **60 s**). After that one probe is sent every 60 s until the window reopens.

Keepalive timer (prevents long idle connections)

- Used in some implementations, usually at the server. It is reset every time the server hears from the client.
- Time-out is usually **2 hours**. After that, the server sends a **probe**. If there is no response after **10 probes, 75 seconds apart**, it assumes the client is down and terminates the connection.

TIME-WAIT (2MSL) timer (connection termination)

- **MSL** (maximum segment lifetime) is the time any segment can exist in the network (common values 30 s, 1 min, 2 min).
- Used after the **active close** sends the final ACK. The connection stays for **2 MSL** so that the final ACK can be **resent** if it was lost (the other end's RTO would then resend FIN).

UDP versus TCP

Parameter	UDP	TCP
Service	Connectionless	Connection-oriented (three phases)
Reliability	Unreliable, optional checksum only	Reliable: checksum, ACKs, retransmission, timers
Data unit	Message-oriented: **user datagram**, boundaries preserved	Stream-oriented: **byte stream** grouped into **segments**
Header	Fixed **8 bytes**	**20 to 60 bytes**
Numbering	None	Byte sequence and acknowledgment numbers
Flow control	None	Yes (rwnd, sliding window)
Error control	None (checksum optional)	Yes
Congestion control	None	Yes (slow start, congestion avoidance, fast recovery)
Checksum	Optional; pseudoheader protocol = 17	**Mandatory**; pseudoheader protocol = 6
Communication	Supports multicasting; no connection setup	Full duplex between two parties; unicast only
Overhead and delay	Very small; fast (2 packets for request-response)	Higher (handshake and termination, at least 9 packets)
Suitable for	Short request-response (DNS), real-time audio and video, SNMP, RIP, TFTP, multicast	Long messages needing reliability: file transfer, e-mail, Web

Choosing between them

- If the application needs **reliable, ordered delivery** and can tolerate delay, choose **TCP**.
- If the application needs **low delay**, sends **short messages**, tolerates loss or has its own error control, choose **UDP**.

MODEL QUESTION (3 Marks): A company runs three services over the Internet: (i) an online fund-transfer application, (ii) a live video-conferencing application, and (iii) a DNS query from a host to its local DNS server. For each service, choose the suitable transport protocol (TCP or UDP) and justify your choice.

NETWORK LAYER

Introduction to the Network Layer

The network layer is responsible for **host-to-host delivery**. At the source host it takes a packet from the transport layer, encapsulates it in a **datagram** and passes it to the data-link layer. At the destination it decapsulates and delivers the payload. **Routers** in the path use only the first three layers (they may use the transport and application layers for control only).

Network-Layer Services

Packetizing

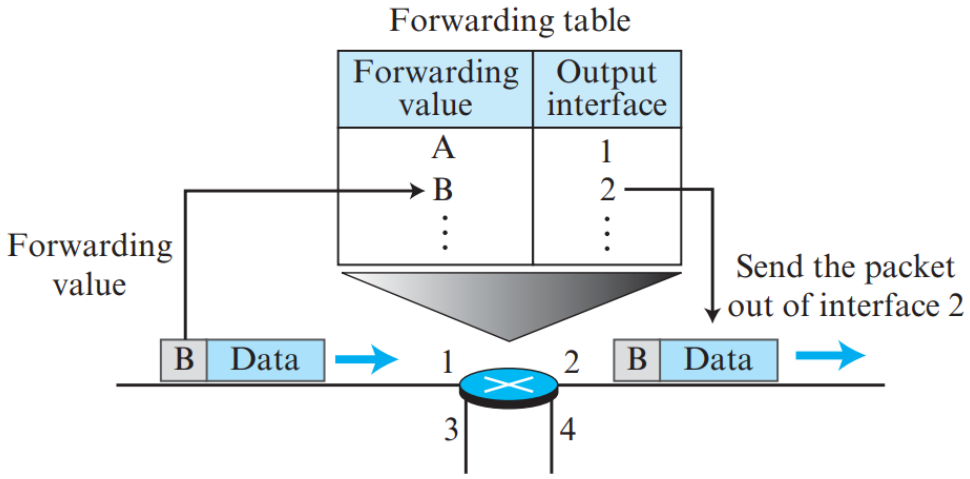
- Encapsulating the payload (data from the upper layer) in a network-layer packet at the source, and decapsulating it at the destination.
- The network layer works as a **carrier** (like the post office): it carries the payload **without changing or using it**.
- The source adds a header with source and destination addresses. The payload is changed only if it is too large and must be **fragmented**.
- The destination waits for all fragments, **reassembles** them and delivers the payload.
- Routers must not change addresses. They only inspect them to forward the packet.

Routing

- Finding the **best route** from source to destination among the many possible routes.
- Done by **routing protocols** that help routers build consistent tables **before** communication occurs.

Forwarding

- The **action** each router performs when a packet arrives at one of its interfaces.
- The router uses information in the packet header (destination address or a label) to find the **output interface** in the **forwarding table** (also called routing table).



About Figure: A packet for B arrives at a router with four interfaces. The forwarding table maps the forwarding value (B) to output interface 2, so the packet is sent out of interface 2.

Other services

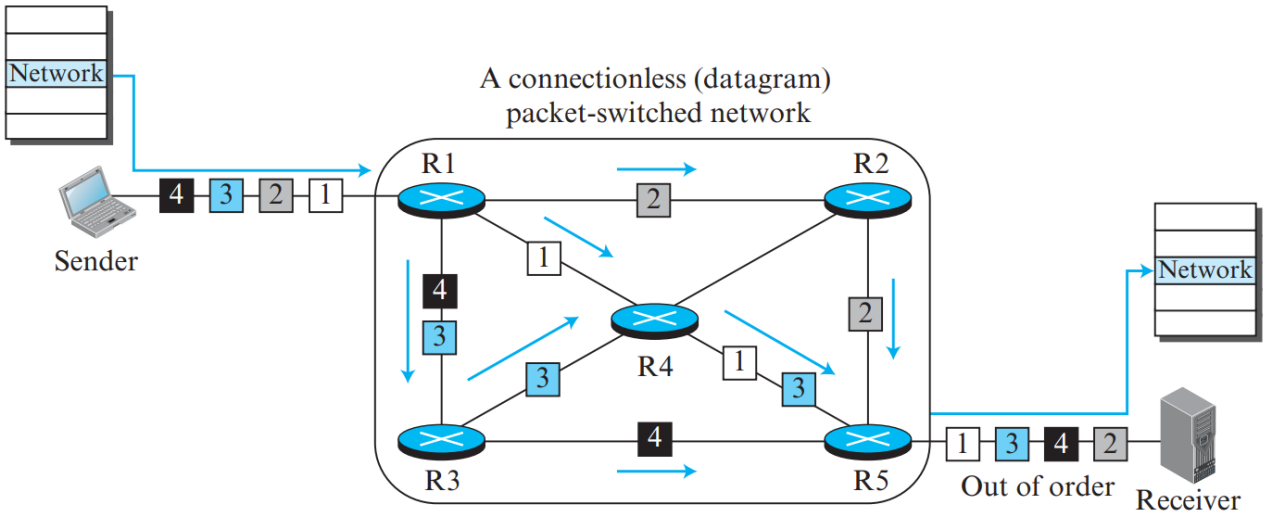
Service	Position in the Internet network layer
Error control	**Not provided** for the data. Reason: packets may be fragmented at each router, so checking would be inefficient. A **header checksum** protects the header only. **ICMP** reports some errors
Flow control	**Not provided.** Reasons: (1) no error control, so the receiving job is simple; (2) upper layers can buffer; (3) upper layers already provide flow control
Congestion control	Too many datagrams in an area. Not implemented in the Internet (discussed below)
Quality of service (QoS)	Important for multimedia. Mostly implemented in the **upper layers** to keep the network layer untouched
Security	Not provided originally. Provided by **IPSec**, a virtual layer that makes the connectionless service connection-oriented

Packet Switching

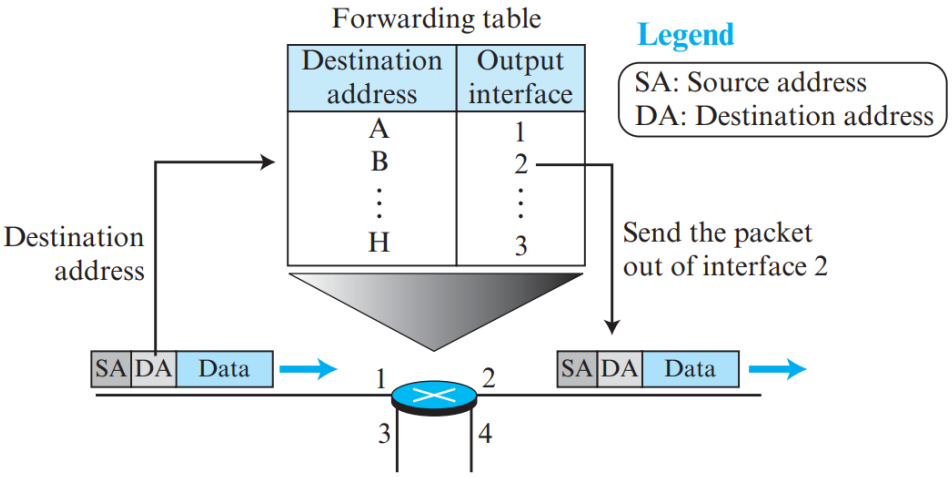
A router is a **switch** connecting an input port to an output port. Only **packet switching** is used at the network layer (circuit switching is mostly physical-layer). A message is divided into packets, sent one by one, and the destination waits for all packets before delivering the message. Two approaches are used: **datagram** and **virtual circuit**.

Datagram Approach: Connectionless Service

- The network layer treats **each packet independently**, with no relationship to any other packet.
- Packets of a message **may take different paths** and may arrive **out of order**.
- Switches are called **routers**.
- Forwarding is based on the **destination address** in the header. The source address is used only to report errors.



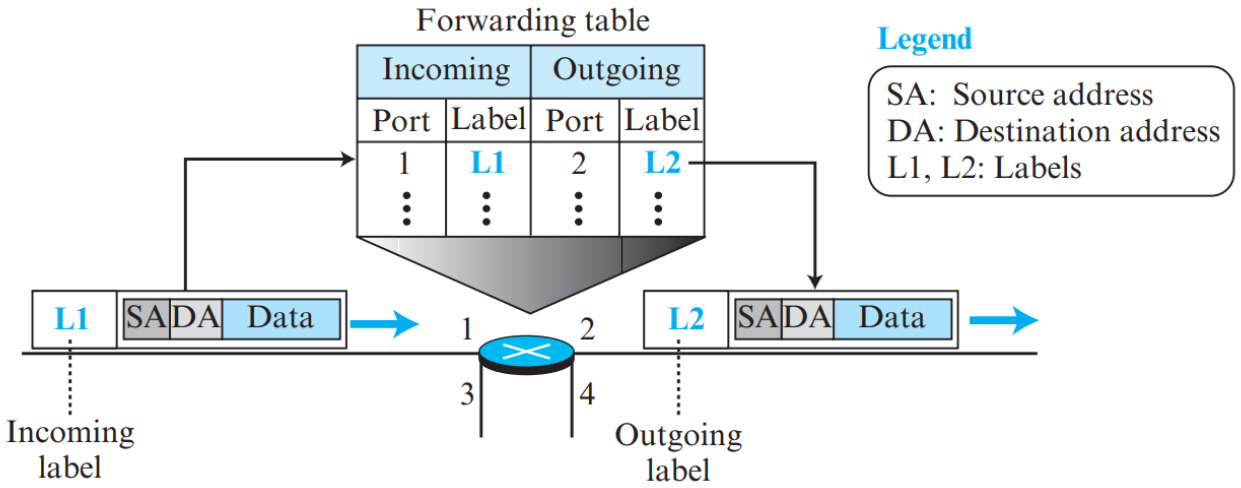
About Figure (ABOVE): Four packets of one message leave the sender, are routed independently through routers R1 to R5, and arrive at the receiver out of order (1, 3, 4, 2).



About Figure: The router looks up the destination address (SA = source, DA = destination) in the forwarding table and sends the packet out of the matching interface.

Virtual-Circuit Approach: Connection-Oriented Service

- All packets of a message are **related**. A **virtual connection** is set up first to define the path.
- Each packet carries the source and destination addresses and also a **label (virtual-circuit identifier)**.
- Forwarding is based on the **label**, using a table with incoming port/label and outgoing port/label.
- Three phases: **setup, data transfer, teardown**.



About Figure: A packet arrives with label L1 on port 1. The forwarding table maps (incoming port 1, label L1) to (outgoing port 2, label L2). The router changes the label and forwards the packet.

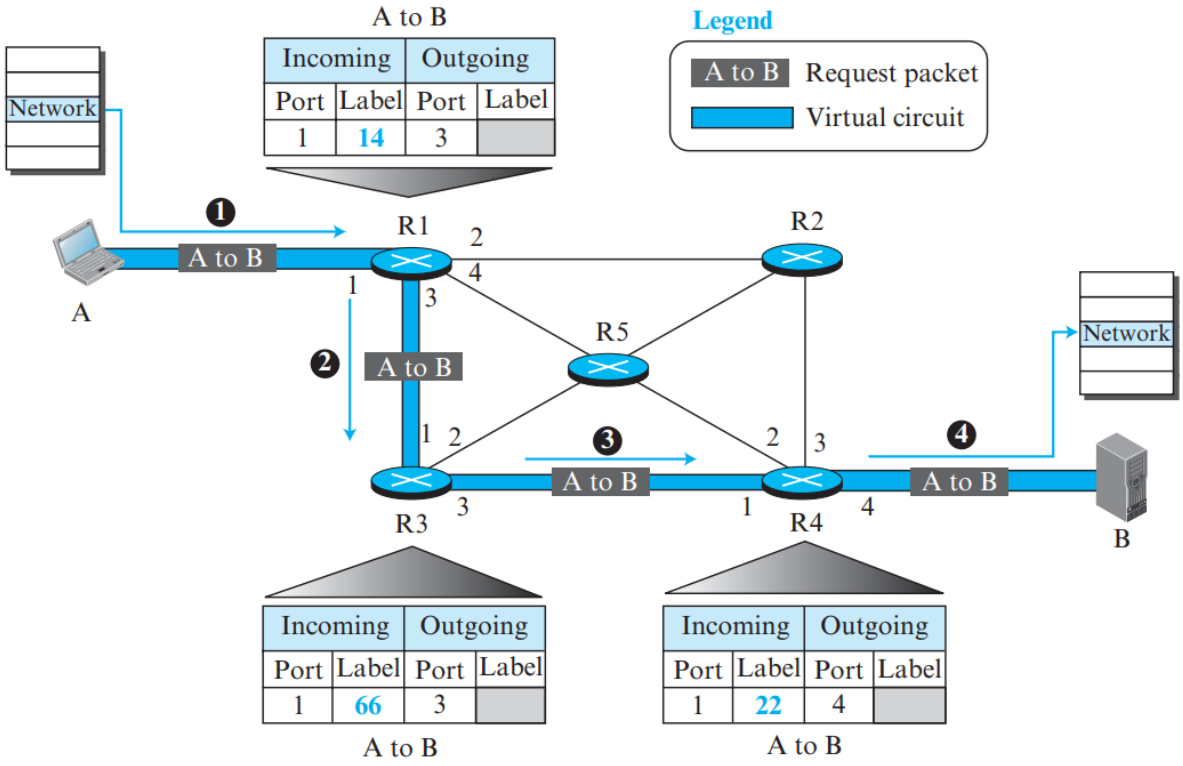
Setup phase (two auxiliary packets: request and acknowledgment)

***Request packet* (source A to destination B)**

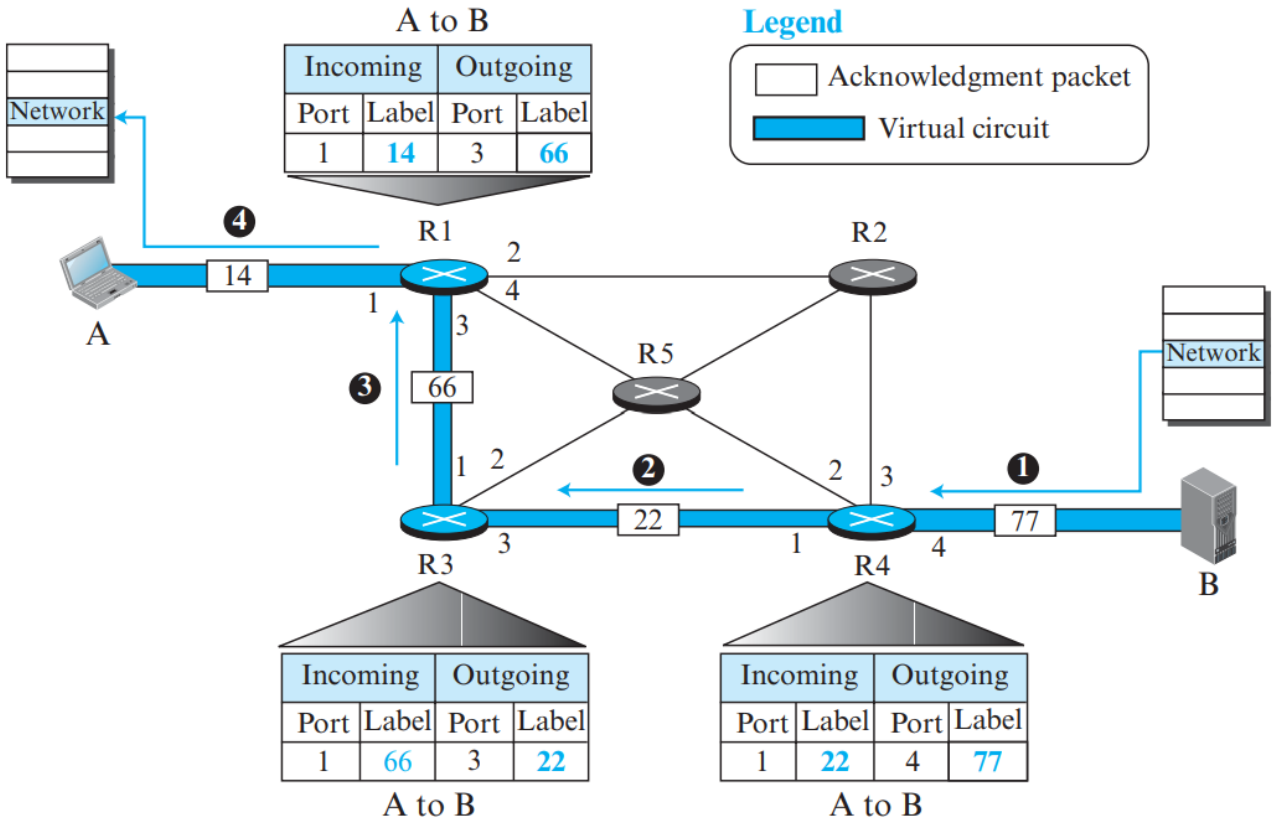
1. A sends the request to router R1.
2. R1 knows A to B leaves through port 3. It creates an entry with incoming port (1), a chosen **incoming label** (14) and outgoing port (3). The outgoing label is not yet known. It forwards the request to R3.
3. R3 and R4 do the same, filling three of four columns.
4. Destination B, if ready, assigns a label (77) to incoming packets from A.

Acknowledgment packet

1. B sends an acknowledgment to R4 carrying label 77. R4 uses it as its **outgoing label**.
2. R4 sends its incoming label (22) to R3, which uses it as outgoing label. R3 sends its incoming label (66) to R1.
3. R1 sends its incoming label (14) to source A, which uses it as the outgoing label for data packets.



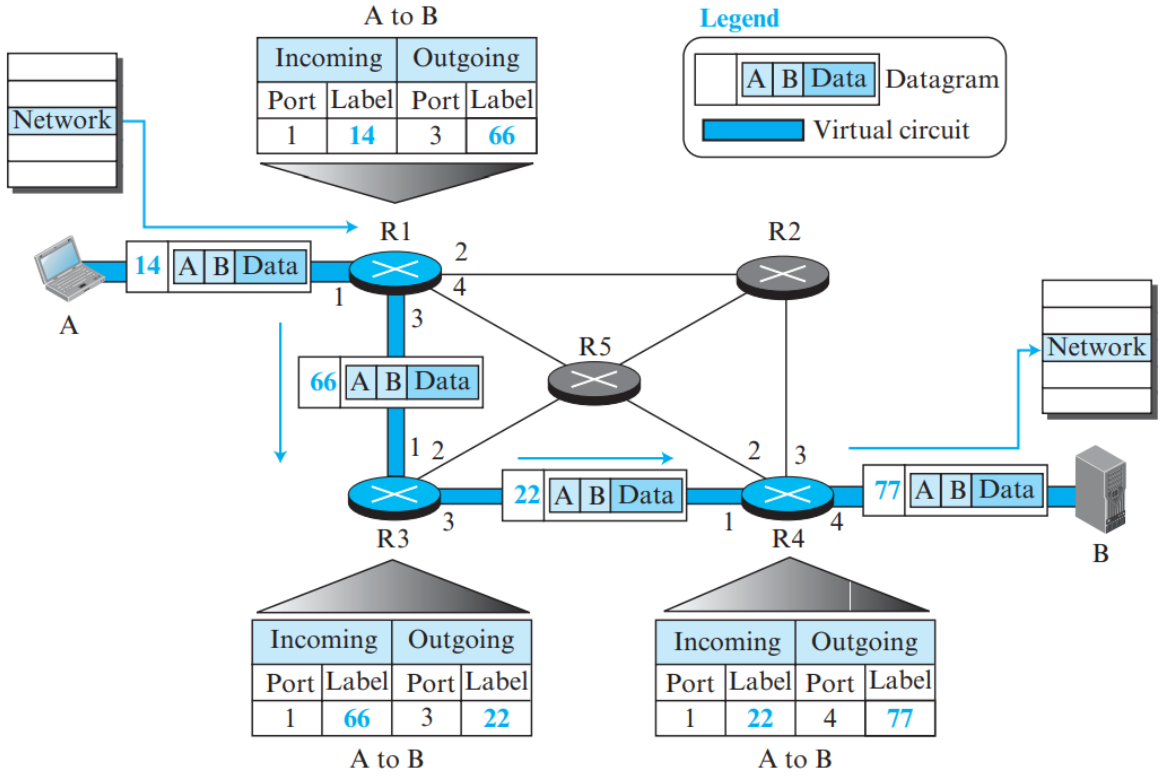
About Figure (ABOVE): The request packet travels from A through R1, R3 and R4 to B. Each router creates a table entry with incoming port, incoming label and outgoing port.



About Figure: The acknowledgment travels back from B through R4, R3 and R1 to A, completing the outgoing-label column at each router.

Data-transfer phase

Packets of the message follow the same sequence of labels. Example: A uses label 14 → R1 changes it to 66 → R3 changes it to 22 → R4 delivers with label 77. Packets arrive **in order**.



About Figure: One packet moves from A to B with labels 14, 66, 22 and 77, and each router's table converts the incoming label to the outgoing label.

Teardown phase

The source sends a **teardown packet**, the destination confirms, and all routers delete the table entries.

Parameter	Datagram (connectionless)	Virtual circuit (connection-oriented)
Relation between packets	Independent	All related
Setup	None	Setup, data transfer, teardown
Forwarding based on	**Destination address**	**Label**
Path	Each packet may take a different path	All packets follow the same path
Order of arrival	May be out of order	In order
Table	Searched by address (routing involves searching)	Accessed by label as an index (switching involves accessing)

Network-Layer Performance

Performance is measured by **delay, throughput and packet loss**. **Delay** has four types.

Delay	Meaning	Formula
Transmission delay	Time to put all bits of a packet on the line	Packet length / Transmission rate
Propagation delay	Time for a bit to travel from A to B in the medium (3×10^8 m/s in vacuum, less in cable)	Distance / Propagation speed
Processing delay	Time to receive a packet, remove the header, check errors and deliver to the output port or upper layer	Average value
Queuing delay	Time a packet waits in the input and output queues of a router	Average value

$$\text{Transmission delay, } \text{Delay}_{tr} = \frac{\text{Packet length}}{\text{Transmission rate}}$$

$$\text{Propagation delay, } \text{Delay}_{pg} = \frac{\text{Distance}}{\text{Propagation speed}}$$

Examples: a 10,000-bit packet on 100 Mbps Fast Ethernet: 10,000 / 100,000,000 = **100 µs**.
A 2000 m cable with speed 2×10^8 m/s: 2000 / (2×10^8) = **10 µs**.

Total delay

With n routers there are n + 1 links.

$$\text{Total delay, Total delay} = (n+1)(\text{Delay}_{tr} + \text{Delay}_{pg} + \text{Delay}_{pr}) + n \times \text{Delay}_{qu}$$

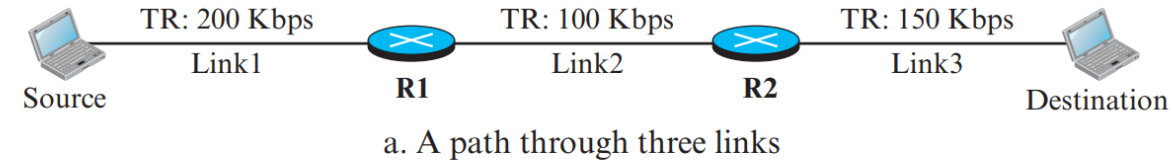
Where n = number of routers. There are (n + 1) transmission, propagation and processing delays but only n queuing delays.

Throughput

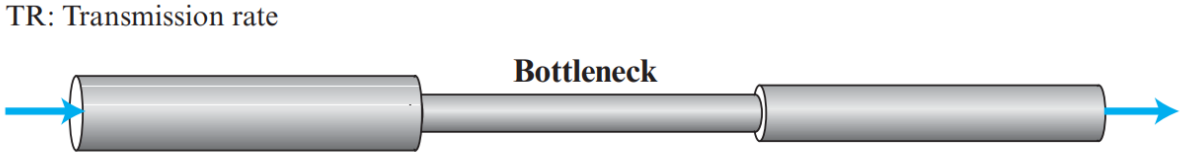
- The number of bits passing a point per second.
- For links in series, throughput is the **minimum** transmission rate (the **bottleneck**).

$$\text{Throughput of a path with n links in series, Throughput} = \min \text{TR}_1, \text{TR}_2, \ldots, \text{TR}_n$$

Example: links of 200, 100 and 150 Kbps give throughput **100 Kbps**.



a. A path through three links



b. Simulation using pipes

Explanation of the Figure: Source to destination through routers R1 and R2 with links of 200, 100 and 150 Kbps. The 100 Kbps link is the bottleneck, shown as the narrowest pipe.

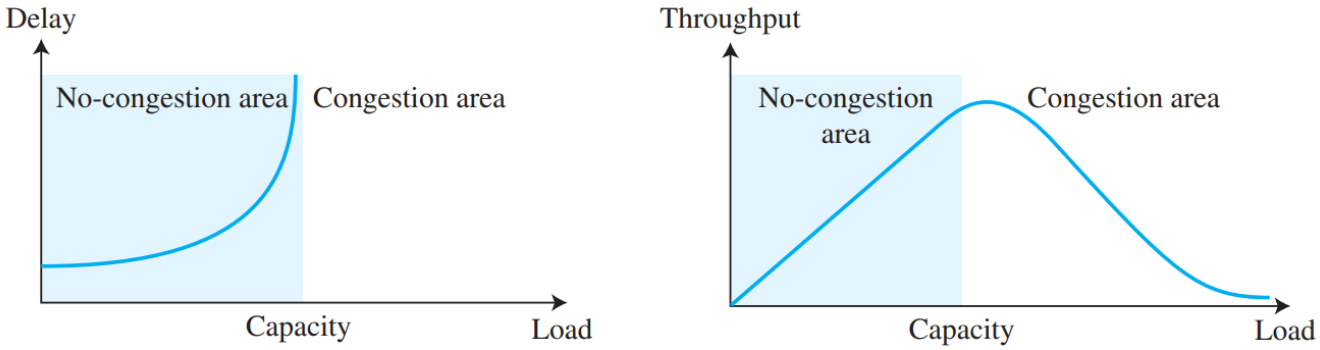
- In the Internet, the backbone is very fast, so throughput is the **minimum of the two access links** (for example a 100 Mbps server and a 40 Kbps dial-up user give 40 Kbps).
- If a link is **shared** by several flows, its rate is divided among them (a 600 Kbps main link shared by 3 paths gives 200 Kbps each).

Packet loss

A router has an input buffer of limited size. If it is full, the next packet is dropped. The lost packet must be resent, which may cause more overflow and more loss.

Network-Layer Congestion

Congestion at the network layer relates to **throughput and delay**.



a. Delay as a function of load

b. Throughput as a function of load

About Figure: (a) Delay stays minimal while load is below capacity and rises sharply (towards infinity) at and beyond capacity. (b) Throughput rises with load up to capacity, then falls sharply in the congestion area.

- **Load below capacity:** delay is minimal (propagation and processing only). Throughput rises with load.
- **Load reaches capacity:** queuing delay is added and delay increases sharply (infinite beyond capacity).
- **Throughput falls** beyond capacity because routers **discard** packets, and the sources **retransmit** them after time-out.

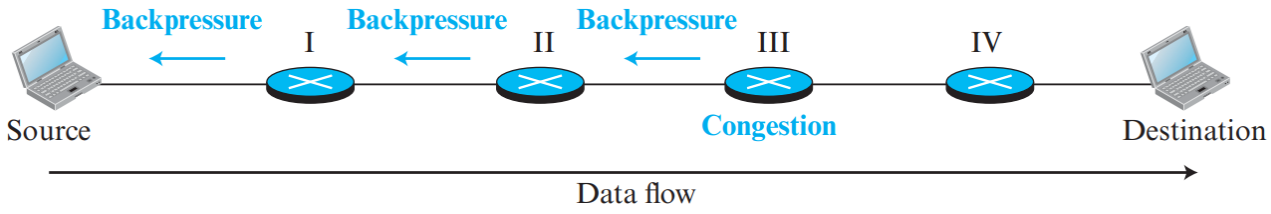
Congestion control has two categories: **open-loop** (prevention) and **closed-loop** (removal).

Open-loop congestion control (policies applied by source or destination)

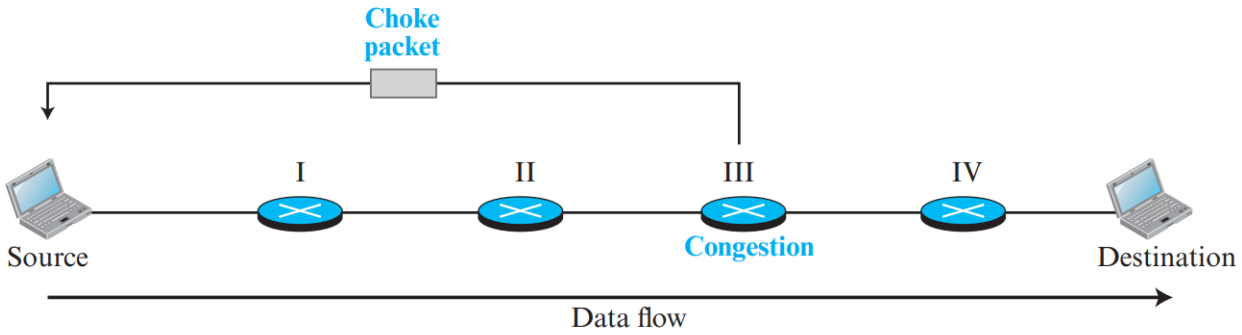
Policy	How it prevents congestion
Retransmission policy	A good policy and timers optimise efficiency and prevent congestion
Window policy	**Selective Repeat** is better than Go-Back-N, because GBN resends packets that may have arrived safely
Acknowledgment policy	Acknowledging fewer packets (only when there is data to send, a timer expires, or every N packets) reduces load, since ACKs are also load
Discarding policy	Routers discard less sensitive packets first (for example in audio) without harming quality
Admission policy	A QoS mechanism for virtual-circuit networks: a router can **deny a connection** if there is congestion or likelihood of it

Closed-loop congestion control (alleviates congestion after it happens)

Mechanism	Working
Backpressure	A congested node **stops receiving data** from its immediate upstream node, which may then become congested and stop its own upstream, and so on back to the source. Node-to-node. **Only in virtual-circuit networks**, where a node knows its upstream node
Choke packet	A congested router sends a packet **directly to the source** to warn it (intermediate nodes are not warned). Example: the **ICMP source quench** message
Implicit signaling	The source **guesses** congestion from symptoms (for example delayed ACKs). Used by TCP
Explicit signaling	The congested node signals the source or destination **inside the data packets** (forward or backward). Seen in ATM networks



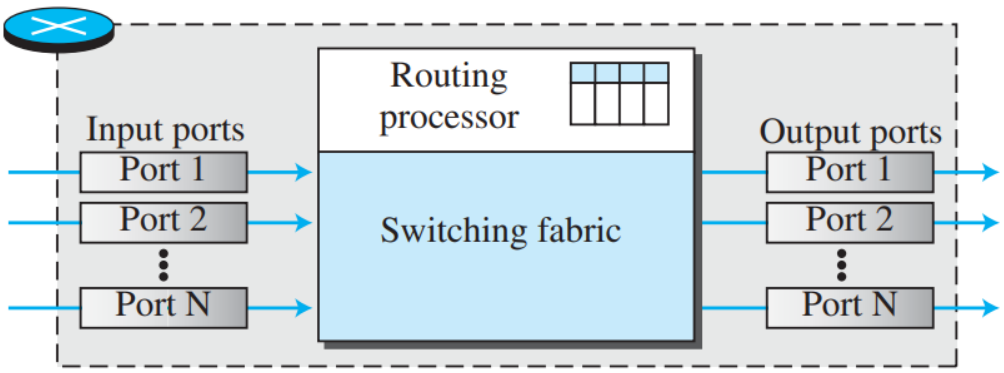
About Figure: Four nodes (I to IV) between source and destination. Congestion at node III sends backpressure to node II, then to node I and finally to the source.



About Figure: A congested node sends a choke packet directly back to the source, bypassing the intermediate nodes.

Structure of a Router

A router has **four components**: input ports, output ports, routing processor and switching fabric.



About Figure: Input ports 1 to N on one side and output ports 1 to N on the other are joined by the switching fabric, with the routing processor controlling it.

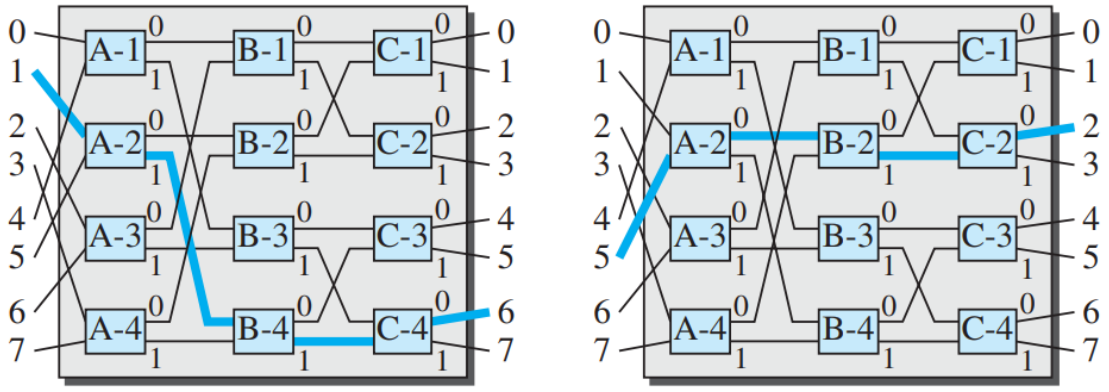
Component	Function
Input port	Performs **physical- and link-layer functions**: builds bits from the signal, decapsulates the packet from the frame, checks errors (discarding if corrupted). Has a **queue** to hold packets before the switching fabric
Output port	Same functions in **reverse order**: queue, encapsulate in a frame, physical-layer functions to create the signal
Routing processor	Performs **network-layer functions**: uses the destination address to find the next hop and output port (**table lookup**). Newer routers move this into the input ports
Switching fabric	Moves the packet from the input queue to the output queue. Its speed affects queue size and delay. Older routers used memory or a bus

Crossbar switch

The simplest fabric. It connects **n inputs to n outputs in a grid** using electronic microswitches at each crosspoint.

Banyan switch

- A **multistage** switch of microswitches that route packets by the **output port as a binary string**.
- For n inputs and n outputs: **$\log_2$ n stages**, with **n/2 microswitches** per stage.
- The first stage uses the highest-order bit, the second the next bit, and so on.
- *Example (8 × 8, three stages):* input 1 to output 6 (**110**): stage 1 routes on bit 1, stage 2 on bit 1, stage 3 on bit 0.
- **Problem:** internal collision is possible even when two packets are not heading to the same output.



a. Input 1 sending to output 6 (110) b. Input 5 sending to output 2 (010)

About Figure: An 8 × 8 banyan switch of microswitches A-1 to C-4. Part (a) routes a packet from input 1 to output 6 (110). Part (b) routes a packet from input 5 to output 2 (010).

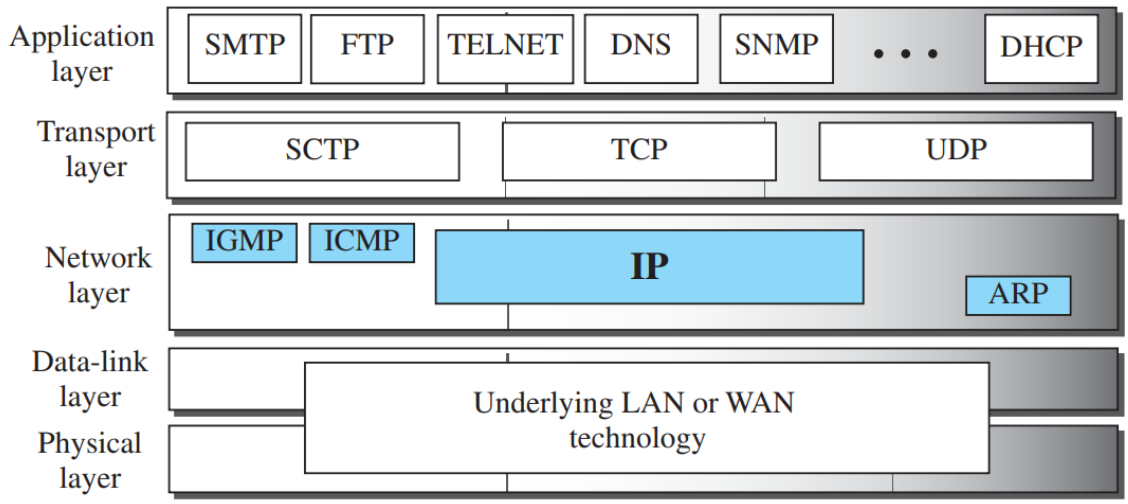
Batcher-banyan switch

A **Batcher switch** placed before the banyan **sorts** incoming packets by destination. A **trap module** between them stops packets with the same destination from entering together (only one per destination per tick). This removes internal collisions.

Network-Layer Protocols: IPv4

The network layer in version 4 has **one main protocol and three auxiliary protocols**.

Protocol	Role
IPv4	Packetizing, forwarding and delivery
ICMPv4	Helps IPv4 handle errors
IGMP	Helps IPv4 in multicasting
ARP	Maps network-layer addresses to link-layer addresses



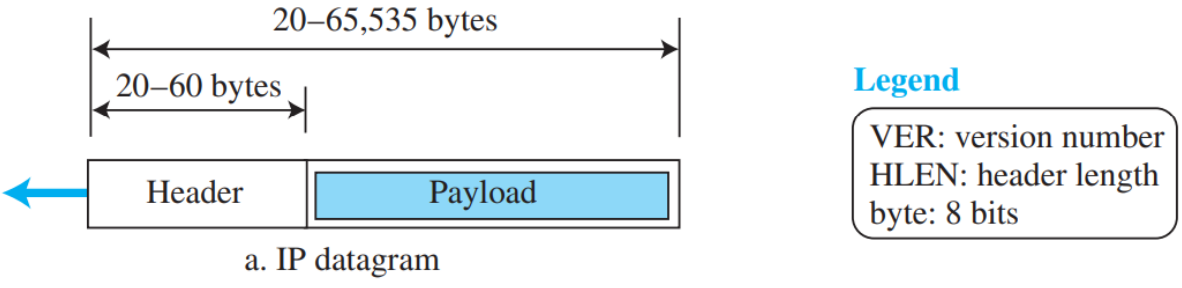
About Figure: IP sits above the data-link layer with ICMP, IGMP and ARP as helpers. TCP, UDP and SCTP in the transport layer use IP, and application protocols (SMTP, FTP, DNS and others) sit above.

IPv4 is an unreliable, connectionless, best-effort datagram protocol:

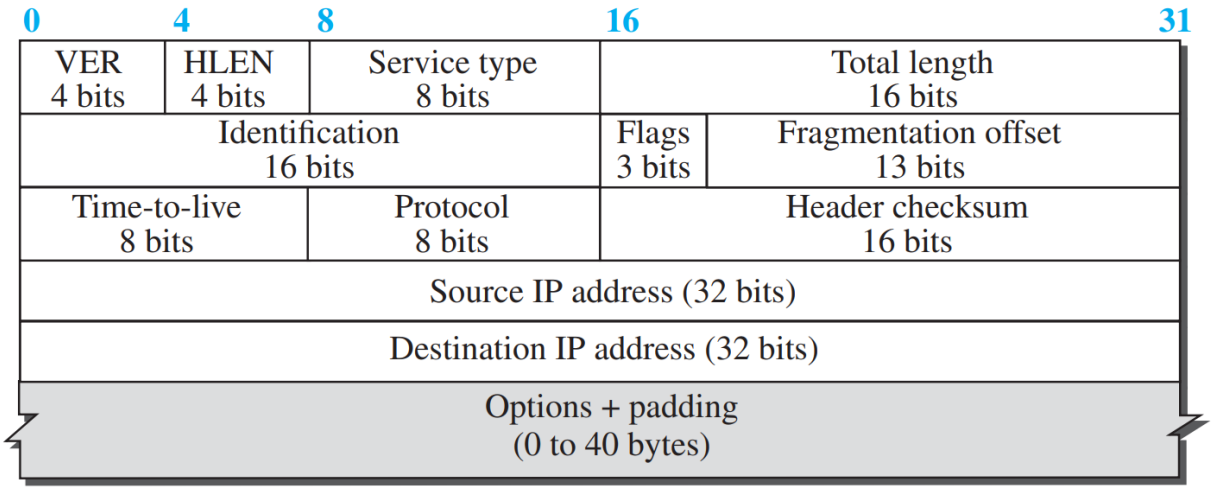
- **Best-effort:** datagrams can be corrupted, lost, out of order or delayed. If reliability is needed, a reliable transport protocol such as TCP is used (like the post office, which does its best but does not track every letter).
- **Connectionless (datagram approach):** each datagram is handled independently and may follow a different route.

IPv4 Datagram Format

A datagram is a variable-length packet of a **header (20 to 60 bytes)** and a **payload**. The header is shown in 4-byte sections.



a. IP datagram



b. Header format

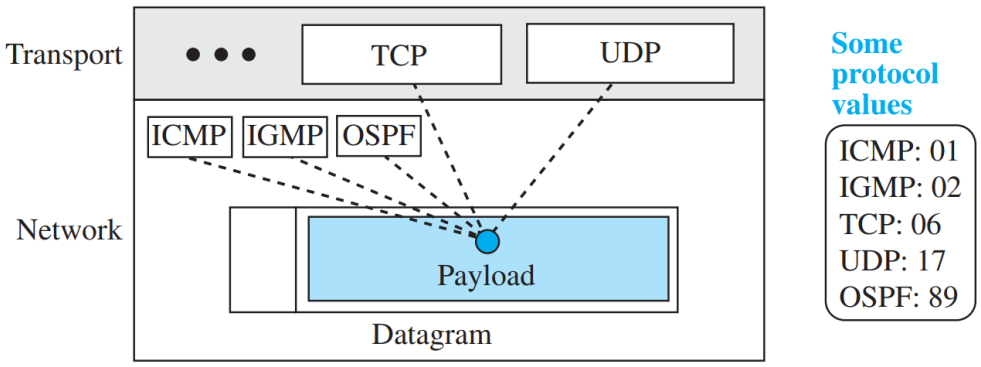
About Figure: The header holds VER, HLEN, service type, total length, identification, flags, fragmentation offset, time-to-live, protocol, header checksum, source IP address, destination IP address and options (0 to 40 bytes), followed by the payload.

Field	Size	Function
Version (VER)	4 bits	Version of IP (value 4)
Header length (HLEN)	4 bits	Header length in **4-byte words**. Multiply by 4 to get bytes
Service type	8 bits	Originally type of service (TOS). Redefined by IETF as **differentiated services (DiffServ)**
Total length	16 bits	Header plus data in bytes. Maximum **65,535** bytes
Identification	16 bits	Identifies the datagram from the source. Same in all fragments
Flags	3 bits	Reserved, **D** (do not fragment), **M** (more fragments)
Fragmentation offset	13 bits	Position of the fragment's data in the original datagram, in **8-byte units**
Time-to-live (TTL)	8 bits	Maximum number of hops. Each router **decrements** it by 1. When it reaches **zero**, the datagram is discarded (avoids endless looping)
Protocol	8 bits	Identifies the protocol of the payload: **multiplexing at the source and demultiplexing at the destination**
Header checksum	16 bits	Checks the **header only**, not the payload. **Recalculated at each router** since TTL and other fields change
Source address	32 bits	IP address of the source (unchanged during travel)
Destination address	32 bits	IP address of the destination (unchanged during travel)
Options	up to 40 bytes	Network testing and debugging. Not required, but processing is required of every IP implementation
Payload		Data from other protocols (TCP, UDP, ICMP, OSPF and so on)

$$\text{Header length, Header length (bytes)} = \text{HLEN} \times 4$$

Where HLEN is the 4-bit field (5 to 15), giving 20 to 60 bytes.

Protocol values: ICMP = 1, IGMP = 2, TCP = 6, UDP = 17, OSPF = 89.



About Figure: The IP payload is delivered to TCP (06), UDP (17), ICMP (01), IGMP (02) or OSPF (89) according to the protocol field value.

The protocol field plays the same role as the port number at the transport layer. Only one protocol field is needed (the same at source and destination), but two port numbers are needed.

Why a header checksum?

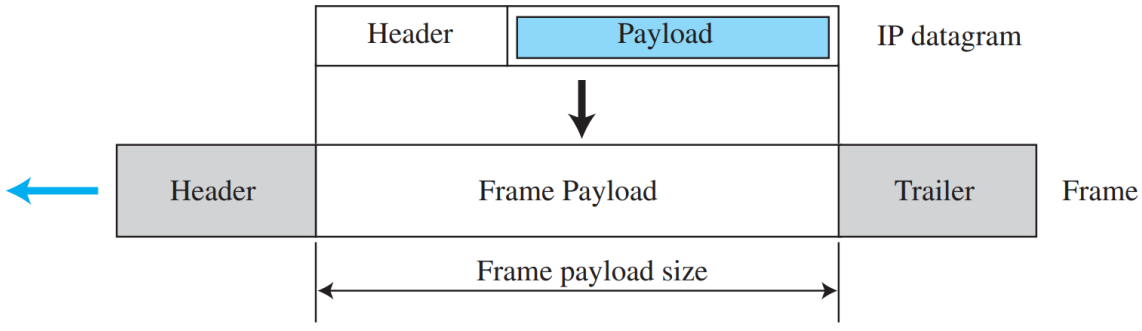
IP checks only the header because a corrupted destination address could send the packet to the wrong host, a corrupted protocol field could deliver the payload to the wrong protocol, and corrupted fragmentation fields would prevent reassembly. The payload is checked by the protocol that owns it (UDP or TCP).

Fragmentation

A datagram may pass through different networks. Each router decapsulates the datagram from the incoming frame and encapsulates it in a new frame of the next network.

Maximum Transfer Unit (MTU)

- Every link-layer protocol limits the **maximum size of the frame payload**. This is the MTU (for a LAN normally **1500 bytes**; WANs differ).
- IP's datagram can be up to 65,535 bytes (to be independent of the physical network). A larger datagram must be divided: **fragmentation**.



MTU: Maximum size of frame payload

About Figure: An IP datagram is placed in the payload of a frame, which also has a header and trailer. The MTU is the maximum size of the frame payload.

Key points

- Each fragment has its own header with most fields **repeated** and a few changed.
- A fragment may be fragmented again.
- Fragmentation can be done by the **source host or any router**.
- **Reassembly is done only by the destination host**, because each fragment is an independent datagram, fragments may take different routes, and reassembling at routers would reduce efficiency.
- The host or router that fragments must change **three fields: flags, fragmentation offset and total length**. All other fields are copied. The **checksum is recalculated** regardless.

Fields related to fragmentation

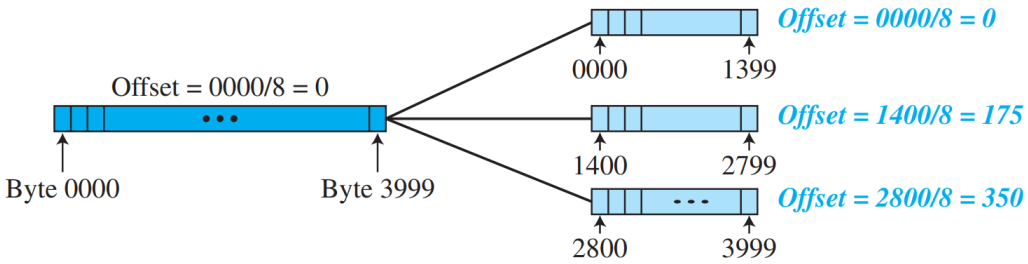
Field	Details
Identification (16 bits)	Identification + source IP address uniquely define a datagram. A counter gives each new datagram the next value. **All fragments have the same value**, which helps the destination reassemble
Flags (3 bits)	Leftmost bit **reserved**. Second bit **D (do not fragment)**: 1 means the datagram must not be fragmented, and if it cannot pass, it is discarded and an ICMP error is sent to the source. Third bit **M (more fragments)**: 1 means more fragments follow, 0 means last or only fragment
Fragmentation offset (13 bits)	Offset of the fragment's data from the start of the original data, in **units of 8 bytes** (13 bits cannot represent more than 8191, so units of 8 bytes are used). The first byte of every fragment's data must be divisible by 8

$$\text{Fragmentation offset, Offset} = \frac{\text{Number of the first data byte in the fragment}}{8}$$

$$\text{Maximum data in a fragment, } D_{max} = \frac{\text{MTU} - \text{header length}}{8} \text{ rounded down, multiplied by 8}$$

Example 4. **A datagram with 4000 bytes of data is fragmented into three:**

Fragment	Bytes	Offset
1	0 to 1399	0/8 = 0
2	1400 to 2799	1400/8 = 175
3	2800 to 3999	2800/8 = 350

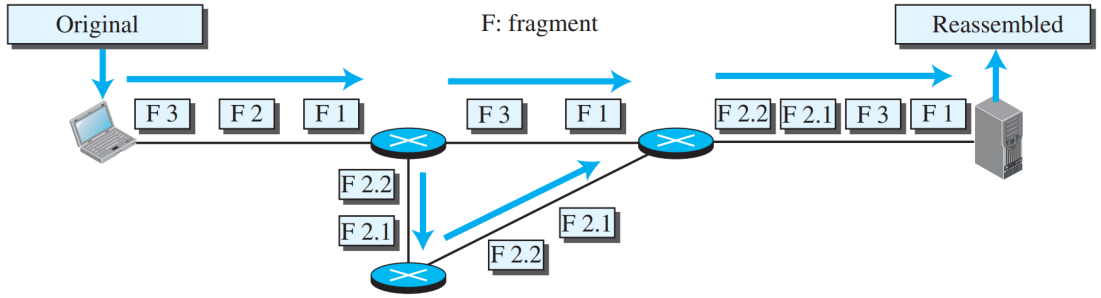


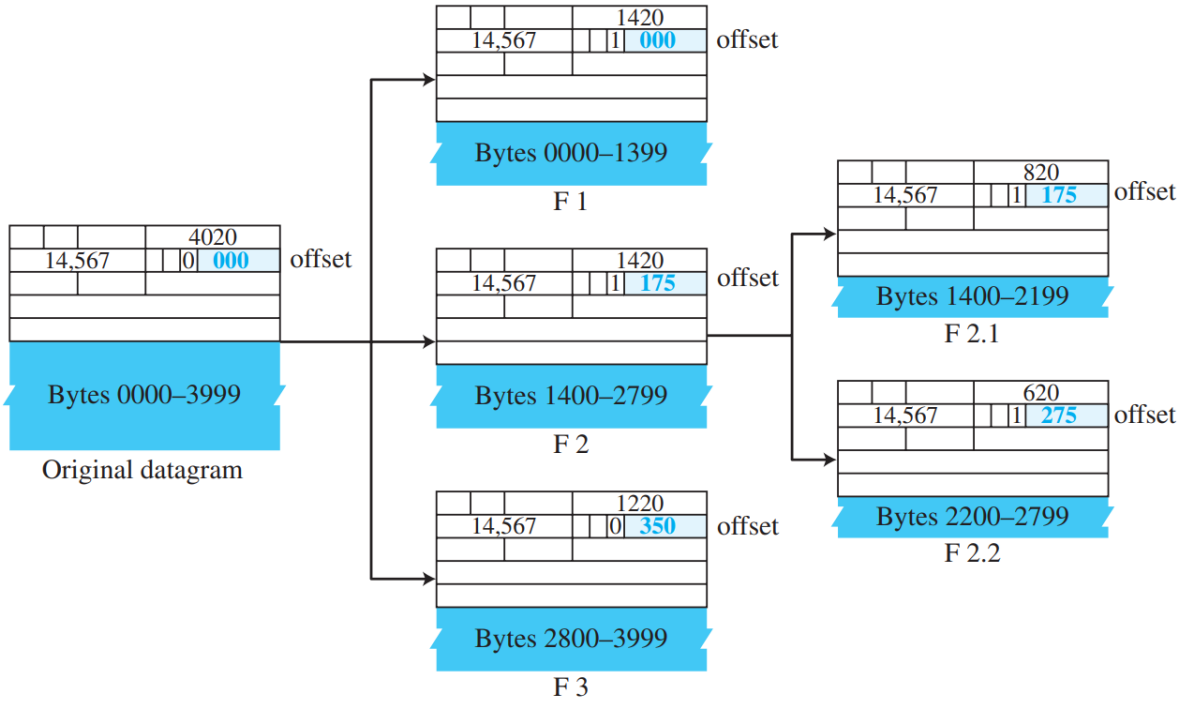
About Figure: A 4000-byte data field (bytes 0 to 3999) is split into three fragments with offsets 0, 175 and 350.

Reassembly at the destination

1. The first fragment has offset 0.
2. Divide the length of the first fragment by 8. The second fragment has offset equal to that result.
3. Divide the total length of the first two fragments by 8. The third fragment's offset is that result.
4. Continue. The **last fragment has M = 0**.

Fragments may arrive out of order and still be reassembled correctly. If a fragment is **fragmented again**, its offset is still **relative to the original datagram**.





About Figure: The original datagram (total length 4020, identification 14,567) is split into F1, F2 and F3 with offsets 0, 175 and 350 and M = 1, 1, 0. F2 is split again into F2.1 and F2.2 with offsets 175 and 275. All carry the same identification.

MODEL QUESTION (3 Marks): An IP datagram of total length 4000 bytes (including a 20-byte header) must pass through a link with an MTU of 1500 bytes. Find the number of fragments. For each fragment, give the total length, the fragment offset and the MF (more fragments) flag.

Solution

1. **Given:** total length = 4000 bytes, header = 20 bytes, MTU = 1500 bytes.
2. **Required:** number of fragments, and for each the total length, offset and MF.
3. **Data in the datagram:** Data $= 4000 - 20 = 3980$ bytes
4. **Maximum data per fragment:** MTU − header = 1500 − 20 = 1480 bytes. It must be divisible by 8: 1480 / 8 = 185, so 1480 is allowed.
5. **Number of fragments:** 3980 / 1480 = 2.69, so **3 fragments**.
6. **Data in each fragment:** 1480, 1480 and 3980 − 2 × 1480 = **1020** bytes.
7. **Offsets** (first data byte / 8):

$$\text{Offset}_2 = \frac{1480}{8} = 185$$

$$\text{Offset}_3 = \frac{2960}{8} = 370$$

Fragment	Data bytes	Total length (data + 20)	Offset	MF
1	0 to 1479	**1500**	**0**	**1**
2	1480 to 2959	**1500**	**185**	**1**
3	2960 to 3979	**1040**	**370**	**0**

8. **Final answer:** the datagram is split into **3 fragments**, with total lengths 1500, 1500 and 1040 bytes, offsets 0, 185 and 370, and MF flags 1, 1 and 0.

Security of IPv4 Datagrams

IPv4 was designed when users trusted each other. Three security issues apply:

Attack	Description	Remedy
Packet sniffing	A **passive attack**: the intruder copies the packet without changing it. Very hard to detect	**Encryption** (the packet can still be sniffed but is unreadable)
Packet modification	The intruder changes the contents and forwards it	**Data integrity** mechanism
IP spoofing	The attacker creates a packet with another computer's source address	**Origin authentication**

IPSec (IP Security) is used with IP. It creates a **connection-oriented service** between two entities and provides four services:

1. **Defining algorithms and keys** (the two parties agree on algorithms and keys).
2. **Packet encryption** (privacy; defeats sniffing).
3. **Data integrity** (modified packets are discarded).
4. **Origin authentication** (prevents spoofing).

IPv4 Addresses

An **IPv4 address** is a **32-bit** address that uniquely and universally defines the **connection** of a host or router to the Internet (not the device itself; if a device moves to another network, the address may change).

- **Unique:** each address defines one connection. A device with two connections has two addresses.
- **Universal:** the addressing system is accepted by any host connected to the Internet.

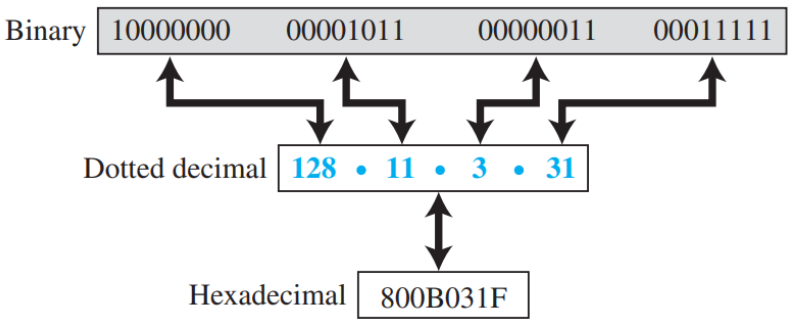
Address space is the total number of addresses a protocol can use. With *b* bits it is 2^b.

$$\text{Address space, } N = 2^b$$

For IPv4, b = 32, so $N = 2^{32}$ = **4,294,967,296** (more than four billion).

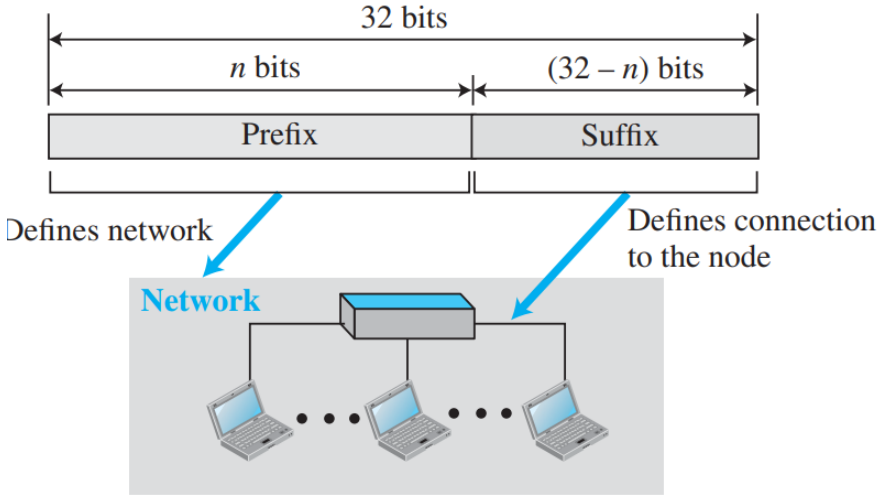
Notation

Notation	Form	Example
Binary (base 2)	32 bits, usually with a space between octets	10000000 00001011 00000011 00011111
Dotted-decimal (base 256)	Each byte as a decimal number 0 to 255	128.11.3.31
Hexadecimal (base 16)	8 hex digits (each = 4 bits); used in network programming	800B031F



About Figure: The same address shown in binary (10000000 00001011 00000011 00011111), dotted decimal (128.11.3.31) and hexadecimal (800B031F).

Hierarchy in addressing: An address has two parts: the **prefix** (n bits) defines the **network**, and the **suffix** (32 − n bits) defines the **node** (connection of a device).

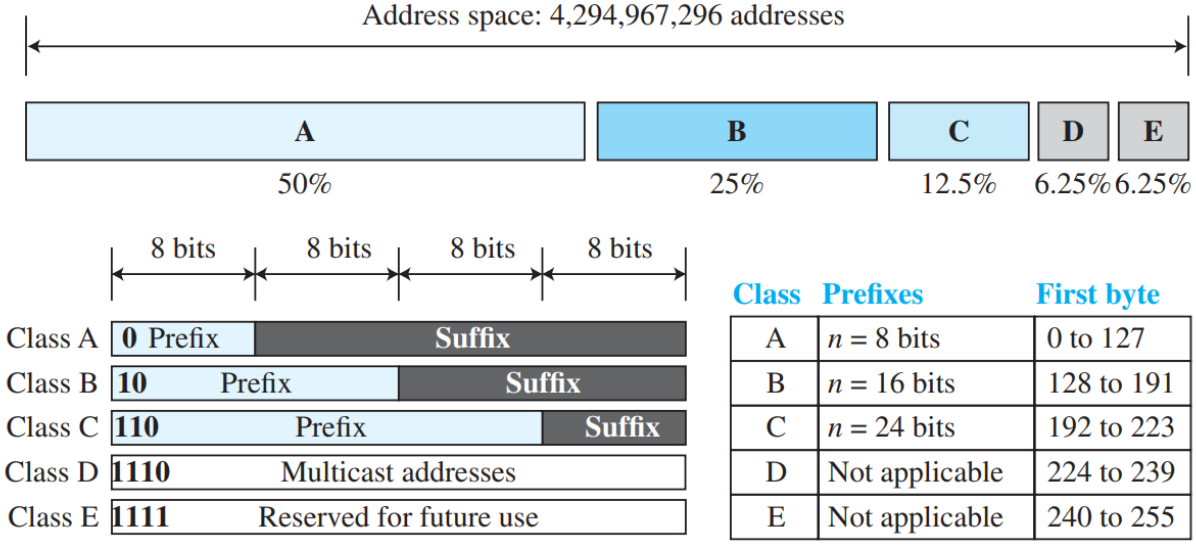


About Figure: A 32-bit address split into an n-bit prefix (defines the network) and a (32 − n)-bit suffix (defines the connection to the node).

Classful Addressing (obsolete)

Prefix length was fixed. The address space was divided into five classes.

Class	Prefix length	Leading bits	First byte range	Networks	Use
A	n = 8	0	0 to 127	$2^7 = 128$	Very large networks (16,777,216 nodes each)
B	n = 16	10	128 to 191	$2^{14} = 16{,}384$	Midsize
C	n = 24	110	192 to 223	$2^{21} = 2{,}097{,}152$	Small (256 addresses)
D	none	1110	224 to 239		**Multicast** addresses
E	none	1111	240 to 255		**Reserved** for future use



Class	Prefixes	First byte
A	*n* = 8 bits	0 to 127
B	*n* = 16 bits	128 to 191
C	*n* = 24 bits	192 to 223
D	Not applicable	224 to 239
E	Not applicable	240 to 255

About Figure: The address space is divided into classes A (50%), B (25%), C (12.5%), D (6.25%) and E (6.25%), with their leading bits and first-byte ranges.

Address depletion made classful addressing obsolete.

- **Class A:** only 128 organisations, each with one single network of 16,777,216 nodes. Most addresses wasted.
- **Class B:** many addresses unused.
- **Class C:** only 256 addresses, too few for most companies.
- **Class E:** almost never used, so wasted.
- **Subnetting** (dividing a class A or B block into subnets) and **supernetting** (combining class C blocks) did not solve it.

Advantage: given an address, the class (and so the prefix length) is found immediately with no extra information.

Classless Addressing

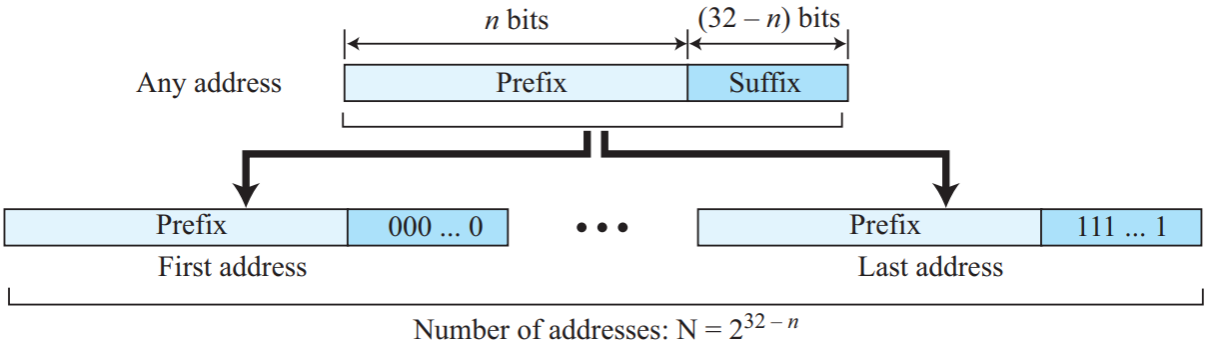
- The short-term solution to address depletion. The class privilege was removed. Announced in **1996**.
- The address space is divided into **variable-length blocks** that belong to no class. The prefix defines the block (network); the suffix defines the node.
- **ISPs** are granted large blocks and subdivide them for customers.
- The number of addresses in a block must be a **power of 2**.
- Classful addressing is a **special case** of classless (class A is prefix 8, class B is 16, and so on).
- **Smaller prefix = larger network.**

Slash notation (CIDR). The prefix length is written after the address, separated by a slash: **a.b.c.d/n**. Example: 12.24.76.8/8.

Extracting information from an address with prefix n:

Number of addresses in the block, $N = 2^{32-n}$

- **First address:** keep the n leftmost bits, set the (32 − n) rightmost bits to **0**.
- **Last address:** keep the n leftmost bits, set the (32 − n) rightmost bits to **1**.



About Figure: A block with an n-bit prefix and (32 − n)-bit suffix. The first address has the suffix all 0s, the last address all 1s, and the number of addresses is 2^(32−n).

Example: Find the block of 167.199.170.82/27.

- $N = 2^{(32-27)} = 2^5 =$ **32** addresses.
- Address: 10100111 11000111 10101010 01010010.
- First address: keep 27 bits, rest 0 → 10100111 11000111 10101010 01000000 = **167.199.170.64/27**.
- Last address: rest 1 → 10100111 11000111 10101010 01011111 = **167.199.170.95/27**.

An address alone does not define its block. For example 230.8.24.56 belongs to different blocks for different prefixes: /16 gives 230.8.0.0 to 230.8.255.255, /27 gives 230.8.24.32 to 230.8.24.63, /29 gives 230.8.24.56 to 230.8.24.63.

Address mask: A 32-bit number with the **n leftmost bits 1** and the rest **0**. Three bitwise operations give the block:

$$\text{Number of addresses using the mask, } N = \text{NOT}(\text{Mask}) + 1$$

$$\text{First address using the mask, First} = \text{Address AND Mask}$$

$$\text{Last address using the mask, Last} = \text{Address OR NOT}(\text{Mask})$$

(For /27, the mask is 255.255.255.224.)

Network address: The first address of the block identifies the network and is used by routers to route a packet to its destination network.

Block allocation: ICANN allocates large blocks to ISPs (not to individuals). Two restrictions:

1. The number of addresses N must be a **power of 2**, so $n = 32 - \log_2 N$.
2. The block must be **contiguous**, and the **first address must be divisible by N**.

$$\text{Prefix length for N addresses, } n = 32 - \log_2 N$$

Example: An ISP asks for 1000 addresses. 1000 is not a power of 2, so **1024** are granted. $n = 32 - \log_2 1024 = \mathbf{22}$. A block such as 18.14.12.0/22 is granted; its first address in decimal (302,910,464) is divisible by 1024.

Subnetting

An organisation (or ISP) can divide its block into subblocks (subnets), and subnets can be divided further.

Design rules (N = total addresses, n = prefix; N_sub, n_{sub} for a subnet):

1. The number of addresses in each subnet must be a **power of 2**.
2. The prefix length of each subnet is found from:

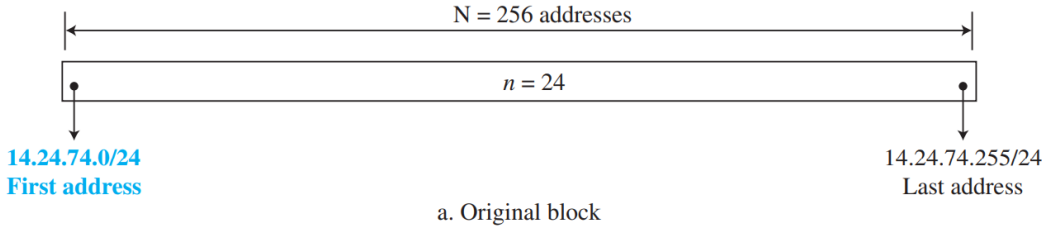
$$\text{Subnet prefix length, } n_{sub} = 32 - \log_2 n_{sub}$$

3. The **starting address** of each subnet must be **divisible by the number of addresses** in it. This is achieved by assigning **larger subnets first**.

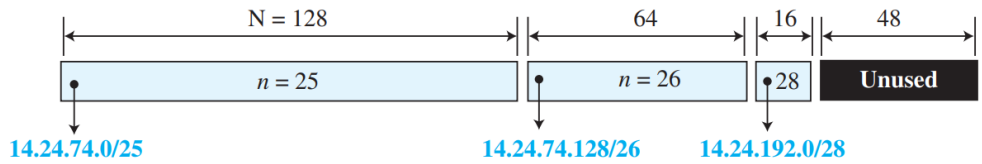
Example*:* The block 14.24.74.0/24 (256 addresses) must supply subblocks for 120, 60 and 10 addresses.

Required	Allocated (power of 2)	Prefix	Block
120	128	$n = 32 - \log_2 128 = \mathbf{25}$	14.24.74.0/25 to 14.24.74.127/25
60	64	$n = 32 - \log_2 64 = \mathbf{26}$	14.24.74.128/26 to 14.24.74.191/26
10	16	$n = 32 - \log_2 16 = \mathbf{28}$	14.24.74.192/28 to 14.24.74.207/28

The total used is 208 addresses, so **48 are left** (14.24.74.208 to 14.24.74.255).



a. Original block

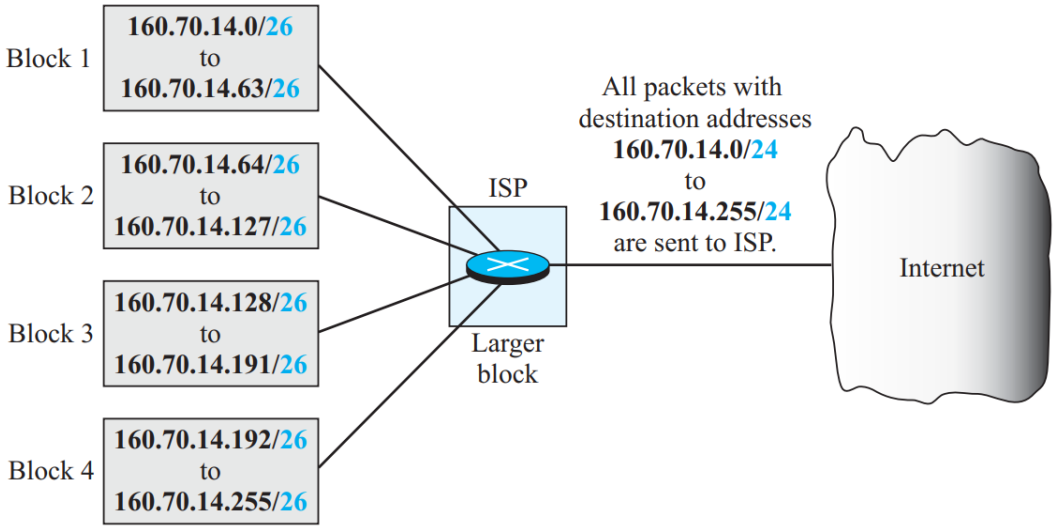


b. Subblocks

About Figure: The original /24 block is shown with its three subblocks (128, 64 and 16 addresses) and the 48 unused addresses.

Address aggregation (route summarisation): When blocks are combined into a larger block, routing is done on the **prefix of the larger block**. ICANN gives a large block to an ISP, which subdivides it for customers.

Example: An ISP assigns four /26 blocks (160.70.14.0/26, .64/26, .128/26, .192/26) to four organisations. It advertises the single larger block **160.70.14.0/24** to the rest of the Internet. All packets for this block go to the ISP, which forwards them to the right organisation.



About Figure: Four /26 blocks belonging to four organisations are combined by the ISP into one larger block, and the Internet sends all packets for the block to the ISP.

Special addresses

Address	Block	Purpose
This-host	0.0.0.0/32	Used when a host does not know its own address and must send a datagram (as source address)
Limited-broadcast	255.255.255.255/32	Send to all devices in the network. Routers block it, so it cannot leave the network
Loopback	127.0.0.0/8	Packet never leaves the host. Used for testing software on one machine
Private addresses	10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16	For internal use; used with NAT
Multicast	224.0.0.0/4	Group addresses

MODEL QUESTION (3 Marks): An organization is given the address 192.168.10.75/26 for one of its hosts. Find (a) the subnet mask (b) the network address and the broadcast address of the subnet (c) the range of usable host addresses and the number of usable hosts.

Solution

1. **Given:** address 192.168.10.75/26, so n = 26.
2. **(a) Subnet mask.** The first 26 bits are 1.
   Mask = 11111111.11111111.11111111.11000000
   Last octet: 11000000 = 128 + 64 = 192. **Mask = 255.255.255.192.**
3. **(b) Network and broadcast address.** Write the last octet of the address: 75 = 01001011.
   - Network address = Address AND Mask:
     "01001011" " AND " "11000000" = "01000000" = 64
     Network address = **192.168.10.64**.

- Number of addresses in the block: $N = 2^{32-26} = 2^6 = 64$
- Broadcast address (last address) = Address OR NOT(Mask):
  01001011 OR 00111111 = 01111111 = 127
  Broadcast address = **192.168.10.127**.

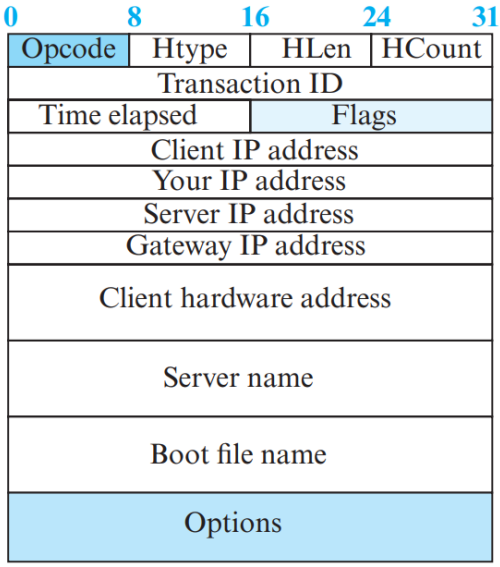
4. **(c) Usable hosts.** The first address (network) and the last (broadcast) are not given to hosts.
   - Range: **192.168.10.65 to 192.168.10.126**.
   - Number of usable hosts: $\text{Hosts} = 2^6 - 2 = 62$
5. **Final answer:** mask **255.255.255.192**; network address **192.168.10.64**; broadcast address **192.168.10.127**; usable range **192.168.10.65 to 192.168.10.126**; **62** usable hosts.

Dynamic Host Configuration Protocol (DHCP)

- A block of addresses can be assigned to hosts **manually**, or **automatically** using **DHCP**.
- DHCP is an **application-layer** client-server program that helps TCP/IP at the network layer. It is called a **plug-and-play** protocol.
- It can give **permanent** addresses to hosts and routers, or **temporary, on-demand** addresses. For example, an ISP with 1000 addresses can serve 4000 households if not more than one-fourth are online at once.
- A host needs **four pieces of information**: its **IP address**, the **network prefix (mask)**, the **address of a default router**, and the **IP address of a name server**. DHCP provides these.

DHCP message format

A request message goes from client to server, and a response comes back.



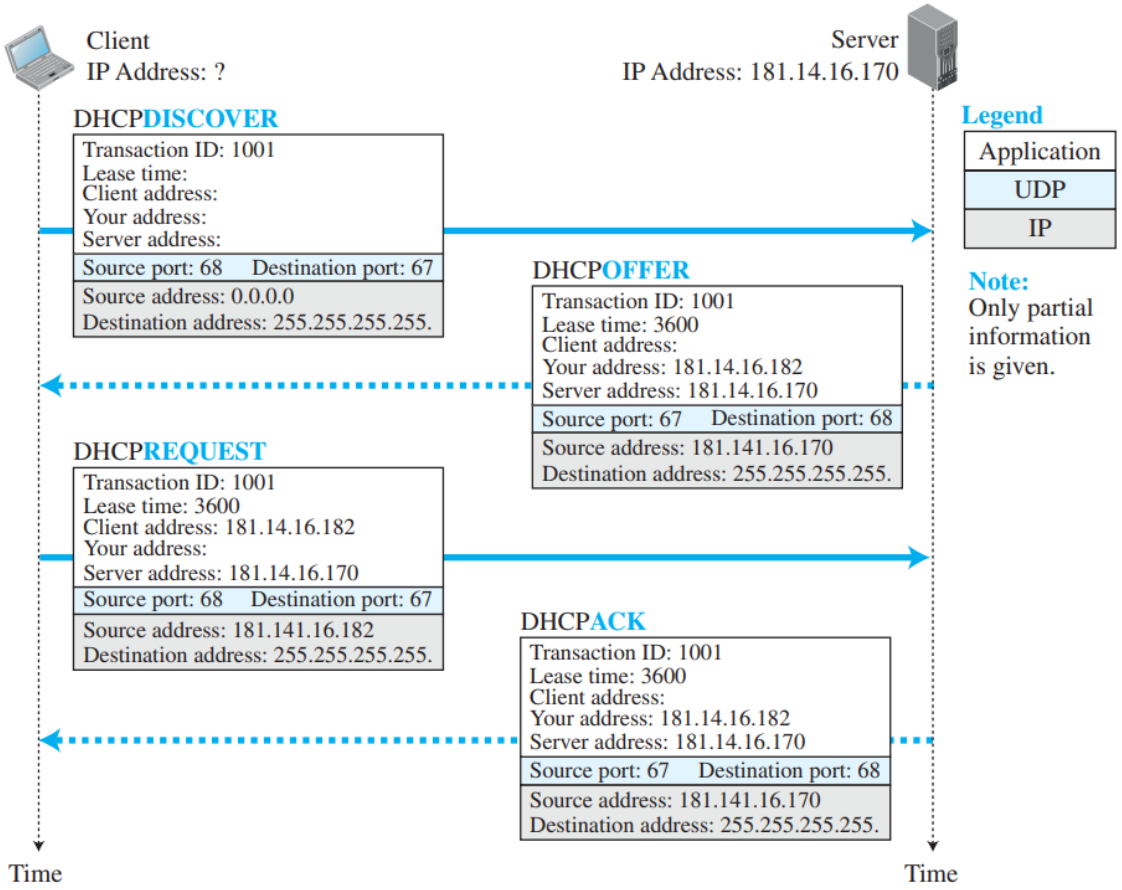
Fields:
Opcode: Operation code, request (1) or reply (2)
Htype: Hardware type (Ethernet, ...)
HLen: Length of hardware address
HCount: Maximum number of hops the packet can travel
Transaction ID: An integer set by client and repeated by the server
Time elapsed: The number of seconds since the client started to boot
Flags: First bit defines unicast (0) or multicast (1); other 15 bits not used
Client IP address: Set to 0 if the client does not know it
Your IP address: The client IP address sent by the server
Server IP address: A broadcast IP address if client does not know it
Gateway IP address: The address of default router
Server name: A 64-byte domain name of the server
Boot file name: A 128-byte file name holding extra information
Options: A 64-byte field with dual purpose described in text

- The option field is introduced by a **magic cookie** (99.130.83.99). Then each option has a 1-byte **tag**, a 1-byte **length** and a **value**.
- If the **tag is 53**, the value gives the **message type**: 1 DHCPDISCOVER, 2 DHCPOFFER, 3 DHCPREQUEST, 4 DHCPDECLINE, 5 DHCPACK, 6 DHCPNACK, 7 DHCPRELEASE, 8 DHCPINFORM.

DHCP operation (UDP client port 68, server port 67)

1. **DHCPDISCOVER:** the joining host sets only the transaction ID (random). It is sent in a UDP datagram, with source IP **0.0.0.0** (this host) and destination **255.255.255.255** (broadcast), because the host knows neither its own nor the server's address.
2. **DHCPOFFER:** each DHCP server replies with the offered IP address (your-IP-address), its own address and the **lease time**. It is **broadcast**, so other servers can make a better offer.
3. **DHCPREQUEST:** the host selects the best offer and sends a request to that server. Source address is the new client address, destination is broadcast (so other servers know their offer was rejected).

4. **DHCPACK:** the server confirms if the address is still valid. If it cannot keep its offer, it sends **DHCPNACK** and the client starts again.



About Figure (ABOVE): A time line between client and server: DHCPDISCOVER (ports 68→67), DHCPOFFER (67→68, your address 181.14.16.182, lease 3600), DHCPREQUEST and DHCPACK, with the source and destination addresses of each datagram.

Two well-known ports: The server's reply is **broadcast**. If the client used a temporary port, another host (for example a daytime client) that accidentally uses the same port number would also receive the reply and be confused. A well-known client port (68) avoids this. If two DHCP clients run together, the **transaction ID** separates the replies.

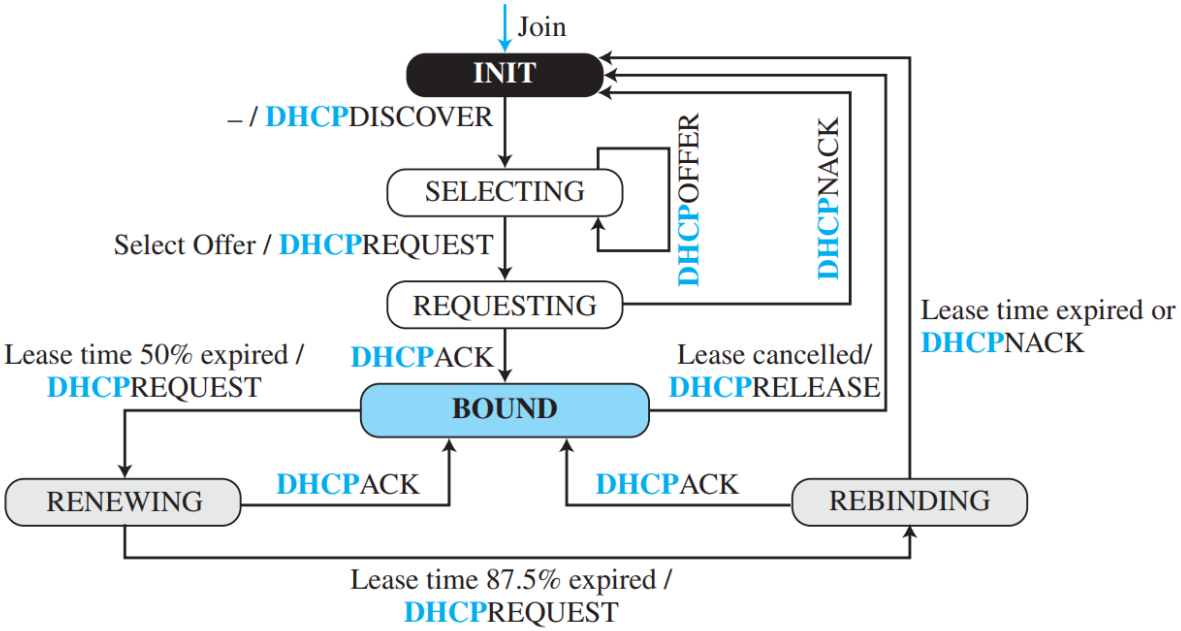
Using FTP: The server does not send everything in the ACK. It gives the **pathname of a file** from which the client can fetch the rest (for example the DNS server address) using a file transfer protocol.

Error control: DHCP uses UDP, which is unreliable. So DHCP **requires the UDP checksum** and the client uses **timers and retransmission** with a **random** timer value (to avoid a traffic jam after, for example, a power failure).

Transition states of the DHCP client

State	Event
INIT	Client starts; broadcasts DISCOVER
SELECTING	Offer received (more offers may come); selects one and sends REQUEST
REQUESTING	Waiting for ACK; on ACK goes to BOUND
BOUND	Uses the address (also usable in RENEWING and REBINDING)
RENEWING	Lease **50 %** expired: tries to renew. ACK returns it to BOUND
REBINDING	Lease **75 %** expired (as stated in the text): tries again. ACK gives BOUND; otherwise back to INIT

Three timers: **renewal timer** (50 % of lease), **rebinding timer** (87.5 %) and **expiration timer** (full lease).

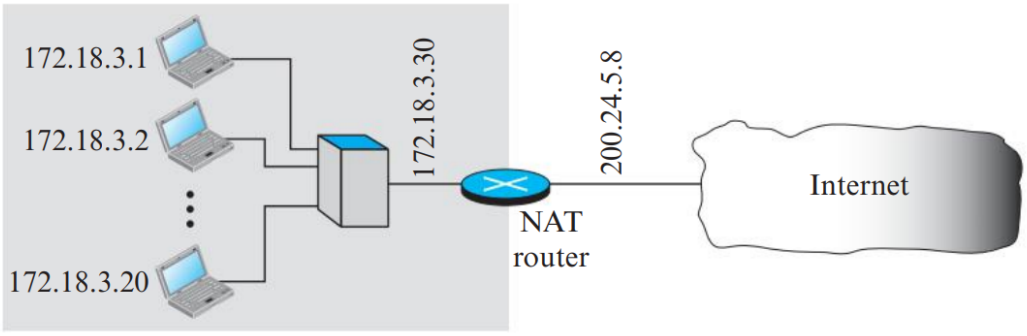


About Figure (ABOVE): States INIT, SELECTING, REQUESTING, BOUND, RENEWING and REBINDING joined by transitions labelled with the messages (DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, DHCPACK, DHCPNACK) and lease-time events.

Network Address Translation (NAT)

Problem: a small site may need only a few Internet addresses at once, but ISP blocks are fixed and cannot grow. **NAT** lets a site use **private addresses internally** and **one or more global addresses** to talk to the Internet. The site has a **single connection** through a **NAT router**.

- The private network is **invisible** to the rest of the Internet, which sees only the NAT router (for example 200.24.5.8).



Site using private addresses

About Figure: A site with private addresses (172.18.3.1, .2, .20) connects to the Internet through a NAT router with private address 172.18.3.30 and global address 200.24.5.8.

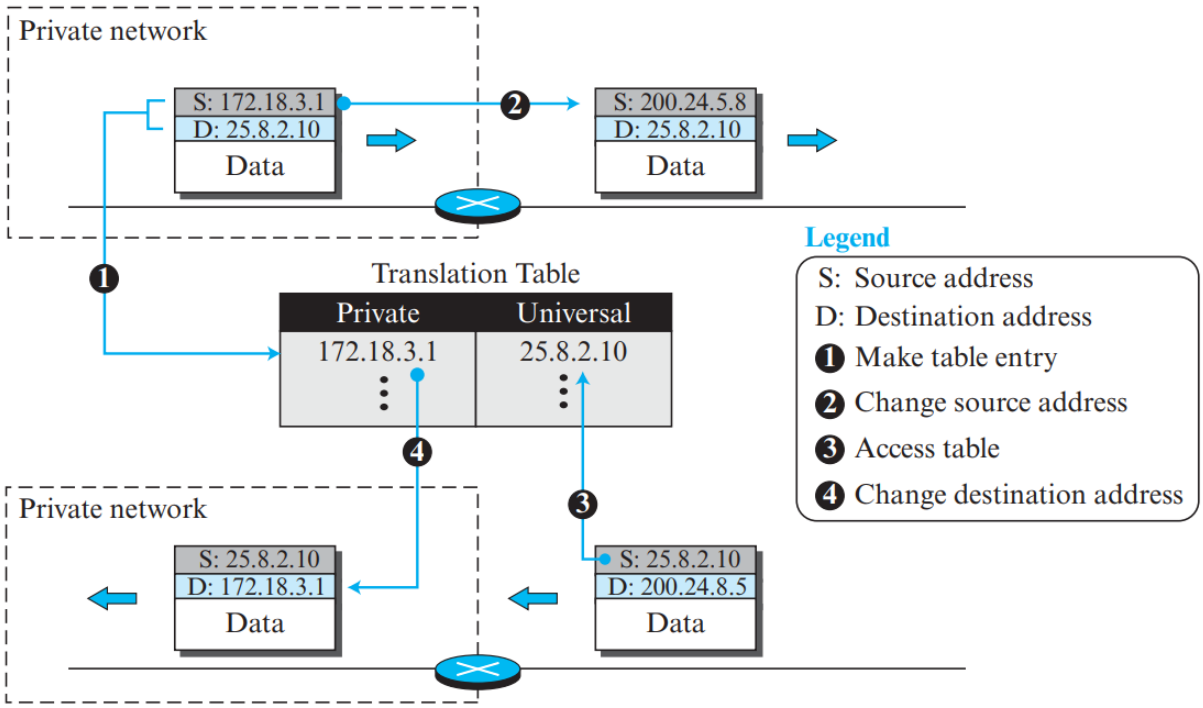
Address translation

- **Outgoing packets:** the NAT router replaces the **source address** with the global NAT address.
- **Incoming packets:** it replaces the **destination address** (the NAT global address) with the correct private address.

Translation table (how the router finds the private host for an incoming packet)

Method	How it works	Limitation
One global address	Two columns: private address, external address (destination of the outgoing packet). The reply's source address is matched as the external address	Only **one** private host can talk to a given external host at a time. Communication must be **initiated from the private network**
Pool of global addresses	Several global addresses (for example 4) are used, so four hosts can talk to the same external host	No more than as many connections as addresses; a host cannot use two server programs of the same external host together; two hosts cannot use the same server program of an external host together
IP addresses and port addresses	**Five columns**: private address, private port, external address, external port, transport protocol	Ephemeral ports (for example 1400, 1401) must be **unique**. Allows many-to-many use

Example: 172.18.3.1 (port 1400) and 172.18.3.2 (port 1401) both use HTTP (port 80, TCP) on external host 25.8.3.2. A reply from 25.8.3.2 to **port 1401** goes to 172.18.3.2.



About Figure: A packet from 172.18.3.1 to 25.8.2.10 leaves through the NAT router, which makes a table entry and changes the source to 200.24.5.8. On the reply, it finds the entry and changes the destination back to 172.18.3.1.

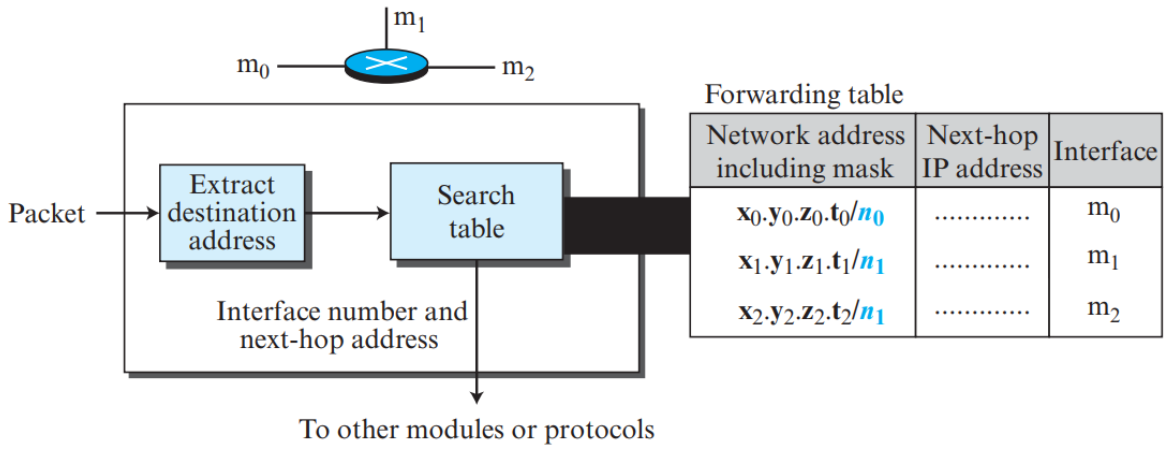
Forwarding of IP Packets

Forwarding places the packet on its route to the next hop. If IP is **connectionless**, forwarding is based on the **destination address**. If it is used **connection-oriented**, forwarding is based on a **label**.

Forwarding Based on Destination Address

A classless forwarding table needs **four items**: the **mask (/n)**, the **network address** (often written together as a.b.c.d/n), the **interface number** and the **next-router IP address**.

Forwarding module: It searches the table **row by row**. In each row, it keeps the n leftmost bits of the destination address, sets the rest to 0 and compares the result with the network address. On a match, it extracts the next hop and interface; otherwise the search continues. The last row is the **default** entry.



About Figure: The forwarding module extracts the destination address from the packet, searches the table (network address with mask, next-hop address, interface) and passes the interface number and next-hop address on.

Example: **Forwarding table of router R1:**

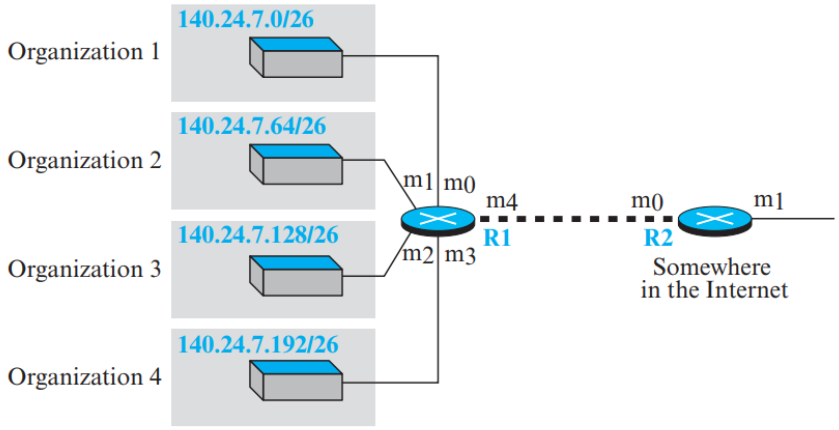
Network address/mask	Next hop	Interface
180.70.65.192/26	—	m2
180.70.65.128/25	—	m0
201.4.22.0/24	—	m3
201.4.16.0/22	—	m1
Default	180.70.65.200	m2

A packet arrives for **180.70.65.140**:

1. Apply /26: result 180.70.65.128 ≠ 180.70.65.192, **no match**.
2. Apply /25: result 180.70.65.128 = 180.70.65.128, **match**. Send out of **interface m0**.

The first row has the **longest prefix** (smallest range) and the fourth the shortest. The table can also show the **n leftmost bits** instead of address and mask (Table 4.3).

Address aggregation: With classless addressing the forwarding table can grow and slow the search. **Aggregation** replaces many entries with one. In Figure 4.47, router R1 connects four organisations (four /26 blocks). Router R2, far away, needs **one entry 140.24.7.0/24** to send everything to R1.



Forwarding table for R1

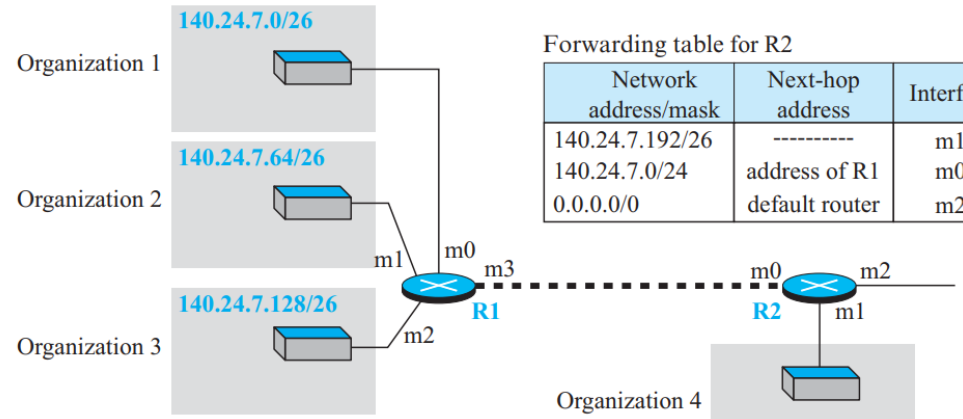
Network address/mask	Next-hop address	Interface
140.24.7.0/26	----------	m0
140.24.7.64/26	----------	m1
140.24.7.128/26	----------	m2
140.24.7.192/26	----------	m3
0.0.0.0/0	address of R2	m4

Forwarding table for R2

Network address/mask	Next-hop address	Interface
140.24.7.0/24	----------	m0
0.0.0.0/0	default router	m1

About Figure (ABOVE): R1 has four entries (one /26 per organisation) and a default entry. R2 has one aggregated entry 140.24.7.0/24 toward R1 and a default entry.

Longest mask matching. If one organisation (for example 140.24.7.192/26) is **not** connected through R1, aggregation can still be used. The table is **sorted from longest mask to shortest**, and the first match is used. A packet for 140.24.7.200 matches /26 first and goes to the right place. Without this order, the /24 mask would send it wrongly toward R1.



Forwarding table for R2

Network address/mask	Next-hop address	Interface
140.24.7.192/26	----------	m1
140.24.7.0/24	address of R1	m0
0.0.0.0/0	default router	m2

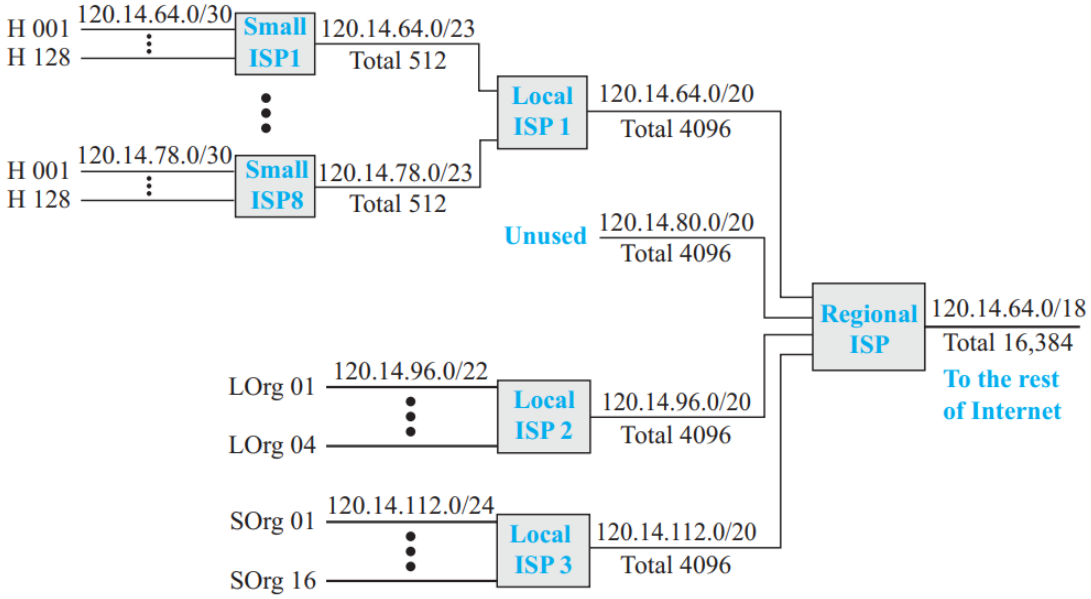
Forwarding table for R1

Network address/mask	Next-hop address	Interface
140.24.7.0/26	----------	m0
140.24.7.64/26	----------	m1
140.24.7.128/26	----------	m2
0.0.0.0/0	default router	m3

About Figure: R2's table has the /26 entry for organisation 4 placed before the /24 entry toward R1, so the packet for organisation 4 leaves by interface m1.

Hierarchical routing: The forwarding table follows the hierarchy of ISPs (backbone, national, regional, local). A local ISP with a block a.b.c.d/n divides it into smaller blocks (e.f.g.h/m, m > n). The rest of the Internet knows only a.b.c.d/n, giving **one entry for all customers**. The levels of hierarchy are unlimited.

Example: A regional ISP has **120.14.64.0/18** (16,384 addresses). It divides it into four /20 blocks (4096 each). Three go to local ISPs and one is kept unused. Local ISP 1 divides its block into 8 /23 blocks for small ISPs (each serving 128 households with /30 blocks of 4 addresses). Local ISP 2 gives four /22 blocks (1024 each) to large organisations. Local ISP 3 gives sixteen /24 blocks (256 each) to small organisations.



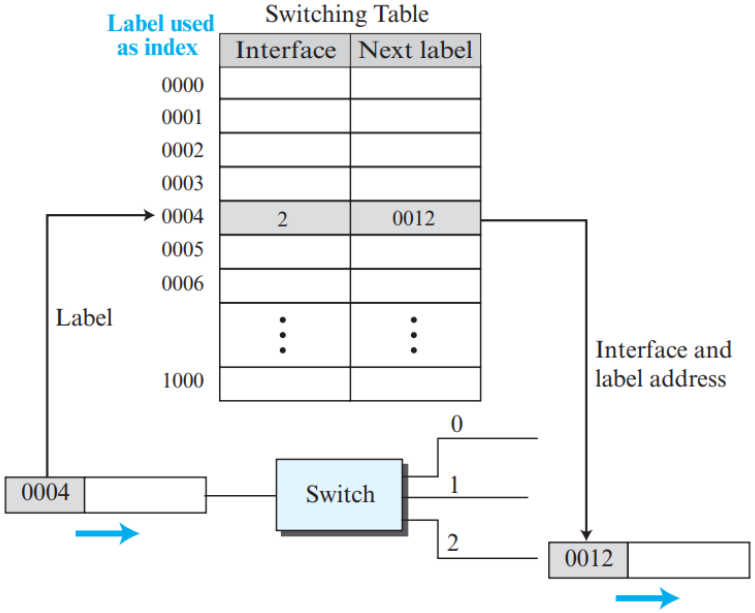
About Figure (ABOVE): A regional ISP block 120.14.64.0/18 divided into local-ISP blocks (/20), which are divided into small-ISP blocks (/23), organisation blocks (/22, /24) and household blocks (/30).

Geographical routing. The whole address space is divided into a few large blocks by continent (America, Europe, Asia, Africa). ISPs outside a region need **one entry** for that block.

Forwarding table search. The simplest method is **longest prefix match** (try the longest prefix first, then the next). It is slow. Better data structures, such as a **trie**, can be used.

Forwarding Based on Label

- In a **connectionless** network, a router **searches** the table using the destination address. In a **connection-oriented** network, a switch **accesses** the table using the **label as an index**, which is immediate.
- Routing involves **searching**; switching involves **accessing**.

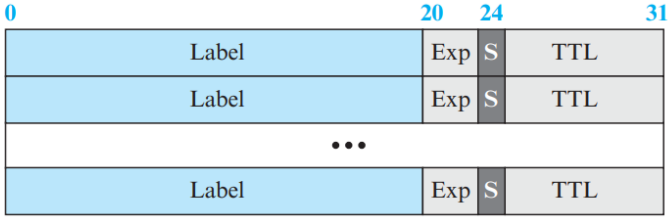


About Figure: The incoming label is used directly as the index into the switching table, giving the outgoing interface and next label.

Multi-Protocol Label Switching (MPLS)

- IETF standard in which routers can behave like both **router** and **switch**: forward by destination address, or by label.
- The IPv4 packet has no label field, so it is **encapsulated in an MPLS packet**, with an MPLS header between the data-link and network layers.
- The MPLS header is a **stack of 32-bit subheaders**, which allows **hierarchical switching**. For example, the top label routes outside an organisation and the bottom label inside it.

MPLS subheader field	Size	Function
Label	20 bits	Index into the forwarding table
Exp	3 bits	Experimental
S	1 bit	Stack bit; **1 means the last** subheader in the stack
TTL	8 bits	Decremented by each router; packet discarded at zero (prevents looping)



About Figure: Each 32-bit subheader holds Label (20 bits), Exp (3 bits), S (1 bit) and TTL (8 bits). Several subheaders are stacked.

ICMPv4

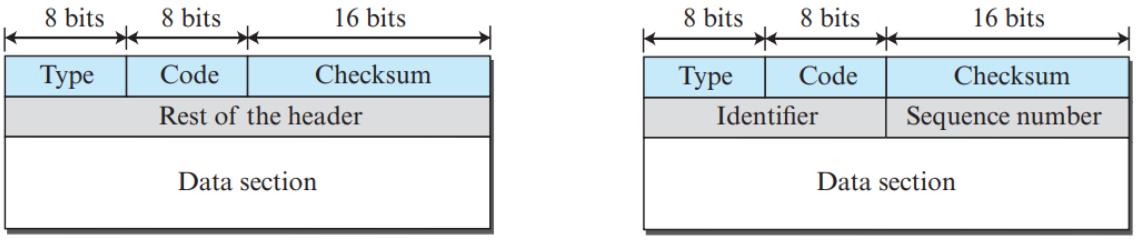
Why ICMP?

IPv4 has:

1. **no error-reporting or error-correcting mechanism** and
2. **no host and management queries**.

ICMP compensates for both. It is a **companion** of IP. Its messages are **encapsulated in IP datagrams** (protocol field = **1**).

Message format: An ICMP message has an **8-byte header** and a variable data section. The first 4 bytes are common: **type** (8 bits), **code** (8 bits, the reason) and **checksum** (16 bits). The rest of the header depends on the message type.



About Figure: Error-reporting messages have type, code, checksum, the rest of the header and a data section. Query messages have type, code, checksum, identifier and sequence number, and a data section. The type and code values are listed.

Error-reporting messages

- ICMP **reports** errors but does **not correct** them. Errors are always sent to the **original source** (using the source IP address).
- Every error message carries the **IP header of the original datagram plus the first 8 bytes of its data** (these give the port numbers for UDP/TCP and the sequence number for TCP).

Type	Message	Purpose
3	**Destination unreachable** (codes 0 to 15)	The datagram could not reach its destination. Code 0 means host unreachable (for example a Web server down)
4	**Source quench** (code 0)	The network is **congested** and the datagram was dropped; the source should slow down (a **choke packet**)
5	**Redirection** (codes 0 to 3)	The source used a wrong router; the message gives the correct default router
11	**Time exceeded** (codes 0, 1)	Code 0: TTL reached 0 and the datagram was dropped. Code 1: not all fragments arrived in time
12	**Parameter problem** (codes 0, 1)	Code 0: problem in the header. Code 1: option missing or cannot be interpreted

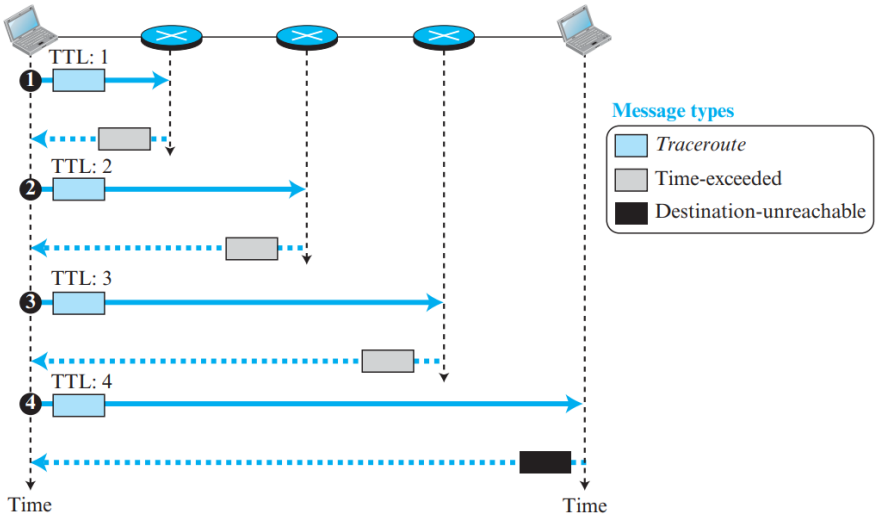
Rules: no error message is generated (1) for a datagram with a **multicast or special address**, (2) in response to a datagram that **carries an ICMP error message**, or (3) for a **fragment that is not the first**.

Query messages come in **request and reply pairs** and test liveness or get information.

Type	Message	Purpose
8 / 0	**Echo request / reply**	Test whether a host or router is alive (used by **ping**)
13 / 14	**Timestamp request / reply**	Find the round-trip time, or check clock synchronisation. The reply returns the request time and adds the receive time and send time

Debugging tools

- **ping:** uses echo request (type 8) and echo reply (type 0). The sequence number starts from 0 and increases by 1. It finds the **round-trip time** (rtt) by putting the sending time in the data section.
- **traceroute (tracert):** finds the **routers on the path**. It sends UDP datagrams with TTL = 1, 2, 3, ... to an **unused port**. Each router that drops a datagram when TTL reaches 0 sends a **time-exceeded** message, revealing its address. The datagram that reaches the destination causes a **destination-unreachable (code 3, port not found)**, so traceroute knows it has arrived. With n routers, n + 1 messages are sent. It uses **three probes** per device for a better RTT estimate. It needs **only the client** program.



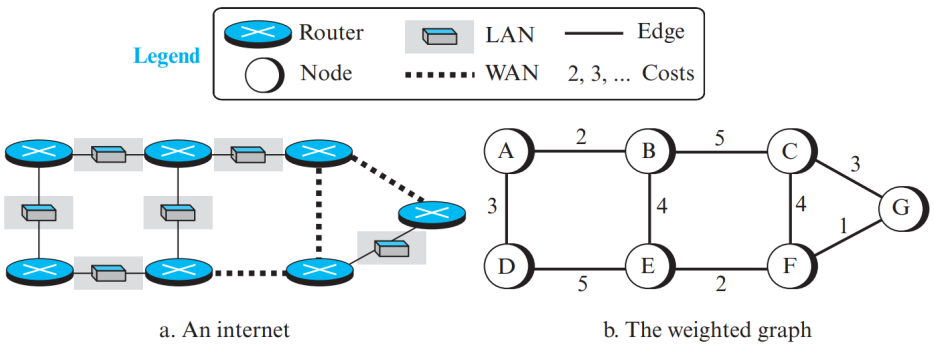
About Figure: Probes with TTL 1, 2, 3 and 4 are sent. The first three routers return time-exceeded messages, and the fourth probe reaches the destination, which returns a destination-unreachable message.

Unicast Routing

The network layer delivers a datagram to its destination. If there is **one destination**, it is **unicast routing**; **several destinations** is **multicast routing**; **all hosts** is **broadcast routing**. Unicast routing in the Internet uses **hierarchical routing** (different algorithms in different steps).

- A packet is routed **hop by hop** using **forwarding tables**.
- The **source host** and **destination host** need no forwarding table; they use their default router. Only **routers** need tables.
- So routing means routing from the **source router** (default router of the source) to the **destination router**.

An internet as a graph. Each **router is a node** and each **network between two routers is an edge**. It is a **weighted graph**: each edge has a **cost** (infinity if no edge). Cost means different things in different protocols.



a. An internet

b. The weighted graph

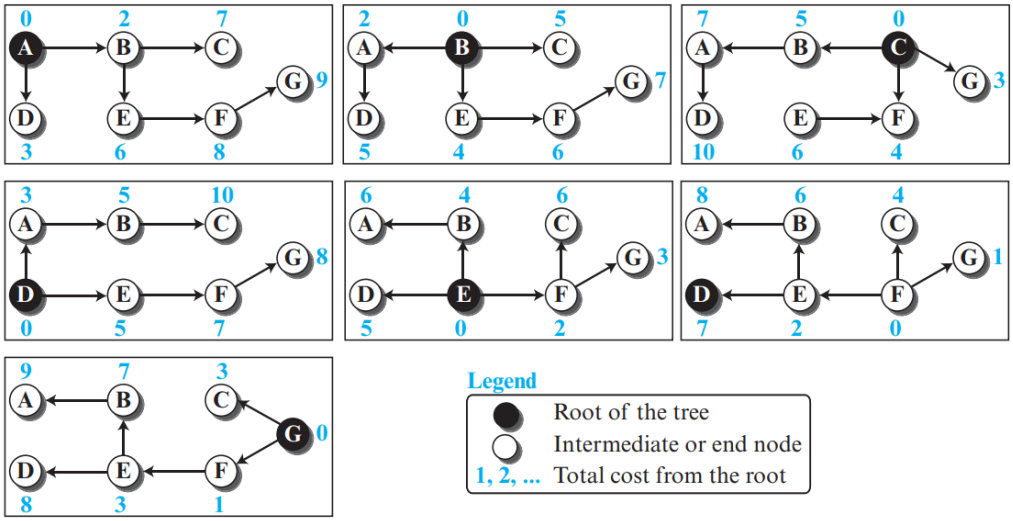
About Figure: (a) An internet of LANs, WANs and routers. (b) The same internet as a weighted graph with nodes A to G and edge costs.

Least-cost routing: The best route is the one with the **minimum total cost**. Each router must find the least-cost route to all other routers.

Least-cost trees: With N routers there are N × (N − 1) least-cost paths. They are combined into one **least-cost tree** per router: the router is the root, the tree spans all nodes, and the path from the root to any node is the shortest. So there are **N trees**.

Properties of least-cost trees (with consistent criteria)

1. The least-cost route from X to Y (in X's tree) is the **inverse** of the route from Y to X (in Y's tree), with the same cost in both directions.
2. A route from X to Z can be made by going X to Y in X's tree and then Y to Z in Y's tree. The cost is the same as the direct route in X's tree.



About Figure: Seven least-cost trees, one rooted at each of the nodes A to G, with the total cost from the root written at each node.

Routing Algorithms

Algorithms differ in how they interpret least cost and how they build the tree. Three are discussed: **distance-vector, link-state and path-vector**.

1. Distance-Vector (DV) Routing

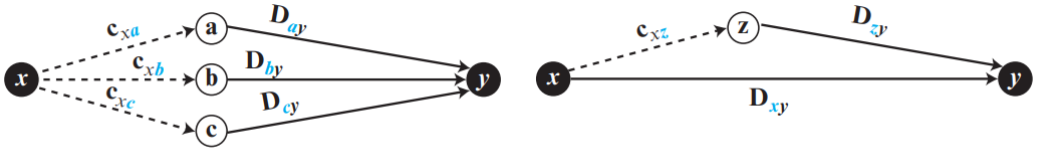
Each node first builds its own tree from information about its **immediate neighbours**. These incomplete trees are **exchanged between neighbours**, so they become more and more complete. A router **continuously tells its neighbours what it knows about the whole internet**.

Bellman-Ford equation: It finds the least cost between node x and node y through intermediate nodes (a, b, c, ...), when the cost from x to each intermediate (c_{xa}) and the least cost from each to y (D_{ay}) are known.

$$\text{Bellman-Ford (general case): } D_{xy} = \min\left(c_{xa} + D_{ay}\right), \left(c_{xb} + D_{by}\right), \left(c_{xc} + D_{cy}\right), \ldots$$

$$\text{Bellman-Ford (updating with one intermediate node z): } D_{xy} = \min D_{xy}, \left(c_{xz} + D_{zy}\right)$$

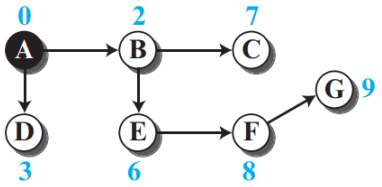
Where D_{ij} = shortest distance between nodes i and j, and c_{ij} = cost of the link between i and j. It builds a new least-cost path from **previously established** least-cost paths.



a. General case with three intermediate nodes

b. Updating a path with a new route

Distance vector: The tree is "unglued" into a **one-dimensional array**. The vector's **name is the root**, the **indexes are the destinations**, and each **cell is the least cost** from the root to that destination. It gives only costs, not paths.



a. Tree for node A

	A
A	0
B	2
C	7
D	3
E	6
F	8
G	9

b. Distance vector for node A

About Figure: The least-cost tree of node A and its distance vector: A = 0, B = 2, C = 5? (as shown in the book), with one cell for each destination.

Initial vectors: When booted, a node sends greeting messages, learns the **identity and distance of its immediate neighbours**, puts those in its vector, and sets all other cells to **infinity**. It sends the vector to its neighbours.

Updating: On receiving a neighbour's vector, a node updates **every cell** with the Bellman-Ford equation (second form). If its vector changes, it sends the new vector to all neighbours.

***Example**:* B receives A's vector, so B's cost to D changes from infinity to **5** (via A, cost 2 + 3). Then B receives E's vector, so B's cost to F changes from infinity to **6** (via E).

New B	Old B	A
A 2	A 2	A 0
B 0	B 0	B 2
C 5	C 5	C ∞
D **5**	D ∞	D 3
E 4	E 4	E ∞
F ∞	F ∞	F ∞
G ∞	G ∞	G ∞

B[ ] = min (B[ ] , 2 + A[ ])

a. First event: B receives a copy of A's vector.

New B	Old B	E
A 2	A 2	A ∞
B 0	B 0	B 4
C 5	C 5	C ∞
D **5**	D **5**	D 5
E 4	E 4	E 0
F **6**	F ∞	F 2
G ∞	G ∞	G ∞

B[ ] = min (B[ ] , 4 + E[ ])

b. Second event: B receives a copy of E's vector.

Note:
X[ ]: the whole vector

About Figure: Two asynchronous events at node B. First, B[ ] = min(B[ ], 2 + A[ ]) after receiving A's vector. Second, B[ ] = min(B[ ], 4 + E[ ]) after receiving E's vector. The old and new vectors of B are shown.

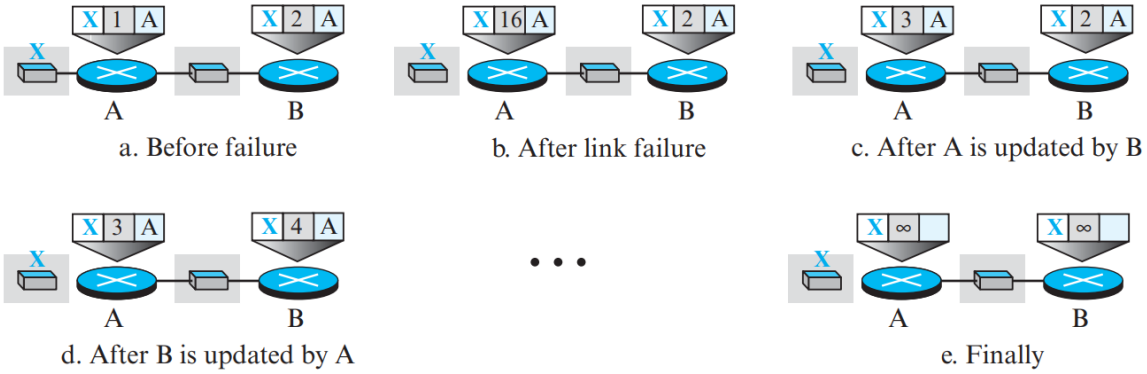
Distance-vector algorithm (pseudocode):

```
Distance_Vector_Routing ( )
{
  // Initialize
  D[myself] = 0
  for (y = 1 to N)
  {
    if (y is a neighbor) D[y] = c[myself][y]
    else                 D[y] = infinity
  }
  send vector {D[1], D[2], ..., D[N]} to all neighbors
  // Update
  repeat (forever)
  {
```

```
    wait (for a vector Dw from a neighbor w or any change in the link)
    for (y = 1 to N)
      D[y] = min [ D[y], (c[myself][w] + Dw[y]) ]    // Bellman-Ford
    if (any change in the vector)
      send vector {D[1], D[2], ..., D[N]} to all neighbors
  }
}
```

Count to infinity: Good news (cost decrease) spreads quickly; **bad news** (cost increase, broken link) spreads **slowly**. Several updates are needed before a broken link is recorded as infinity everywhere.

Two-node loop: A and B both reach X, with A's route direct. The link A–X fails. If B sends its table to A **before** A's update reaches B, A thinks B can reach X and updates (cost 3 via B). Then B updates (cost 4 via A), and so on. The cost increases gradually to infinity. Meanwhile packets for X **bounce between A and B**.



About Figure: Tables of A and B for destination X before failure, after the link fails, and after successive wrong updates (costs 2, 3, 4 and so on) until both show infinity.

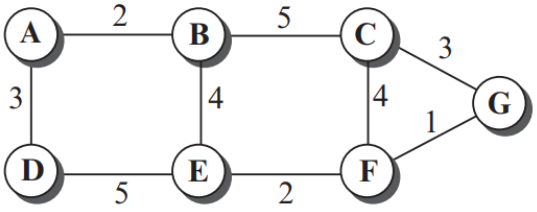
Solution	Working
Split horizon	A node sends through an interface **only the part of its table not learned from that interface**. B does not advertise to A the route to X that came from A. So A keeps infinity
Poisoned reverse	B **still advertises** the route to X to A, but with **infinity** (a warning: "what I know comes from you"). This removes the doubt of whether the route is missing because of split horizon or because B has no news

Three-node instability: split horizon with poisoned reverse prevents two-node loops, but **stability cannot be guaranteed for three nodes**.

2. Link-State (LS) Routing

This algorithm follows the least-cost-tree idea directly. The cost of an edge defines the **state of the link** (infinity means the link is absent or broken).

Link-state database (LSDB). The collection of the states of **all links**, represented as a **matrix** where each cell is the cost of a link. There is **one LSDB** for the whole internet, and **each node keeps a copy**.



a. The weighted graph

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

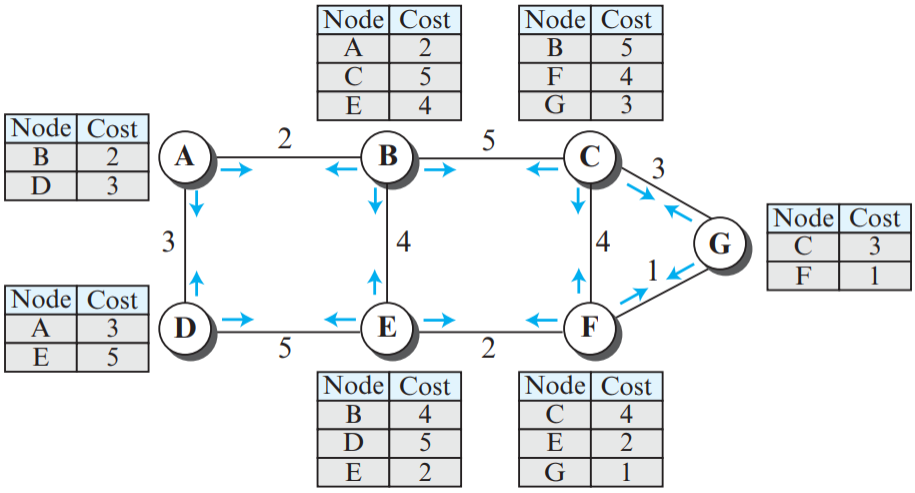
b. Link state database

About Figure: (a) The weighted graph. (b) The LSDB as a 7 × 7 matrix with the cost of each link, 0 on the diagonal and infinity where no link exists.

Building the LSDB by flooding

1. Each node sends greeting messages to its immediate neighbours and learns each neighbour's **identity** and the **link cost**. This pair of information is the **LS packet (LSP)**.
2. The LSP is sent out of **every interface**.
3. A node receiving an LSP compares it with its stored copy using the **sequence number**. An **older** LSP is **discarded**. A **newer or first** LSP replaces the old one and is sent out of every interface **except the one it arrived on**. This ensures flooding stops.
4. At the end, every node has the **same, complete LSDB**.

DV versus LS: in DV each router tells its **neighbours** what it knows about the **whole internet**; in LS each router tells the **whole internet** what it knows about its **neighbours**.



About Figure: For each node A to G, the LSP lists its neighbours and the link costs (for example A: B 2, D 3, E? as shown), forming the data from which the LSDB is built.

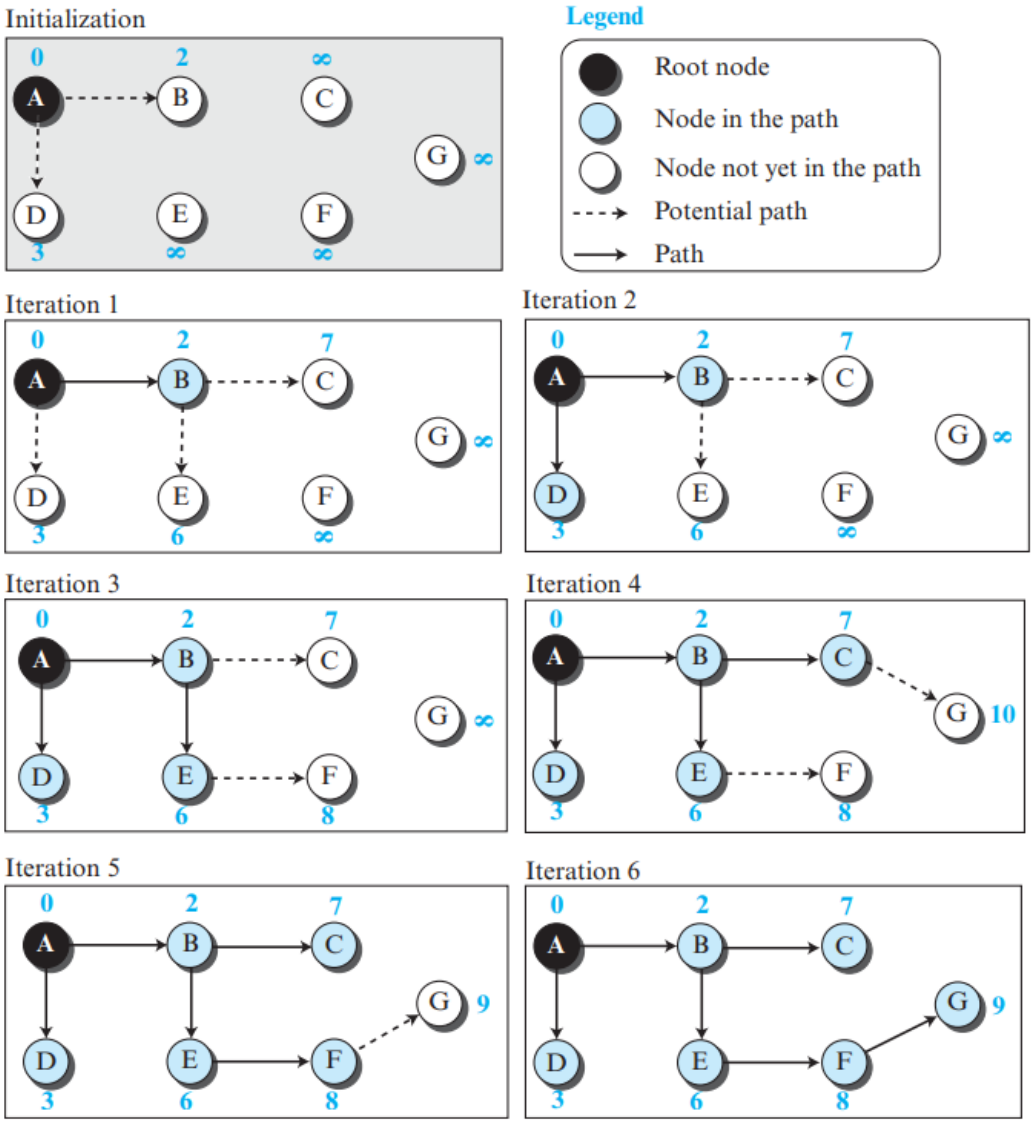
Formation of the least-cost tree: Dijkstra's algorithm:

1. The node makes **itself the root** and sets the cost of every other node from the LSDB (neighbours get the link cost, others infinity).
2. It selects the node **not yet in the tree that is closest to the root** and adds it. Then it **updates the costs** of the remaining nodes through the new node.
3. Step 2 repeats until **all nodes are in the tree**.

```
Dijkstra's Algorithm ( )
{
  // Initialization
  Tree = {root}
  for (y = 1 to N)
  {
    if (y is the root)          D[y] = 0
    else if (y is a neighbor)   D[y] = c[root][y]
    else                        D[y] = infinity
  }
  // Calculation
  repeat
  {
    find a node w, with D[w] minimum among all nodes not in the Tree
    Tree = Tree U {w}
    for (every node x, neighbor of w and not in the Tree)
      D[x] = min { D[x], (D[w] + c[w][x]) }
  } until (all nodes are included in the Tree)    }
```

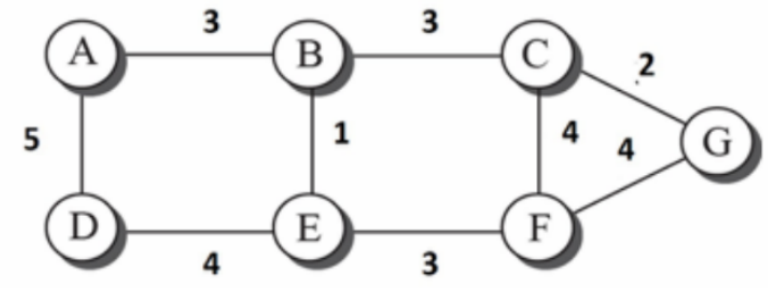
Dijkstra update step, $D[x] = \min D[x], (D[w] + c[w][x])$

Where w is the node just added to the tree and x is a neighbour of w not yet in the tree.



About Figure: The initialisation and six iterations of Dijkstra's algorithm on the graph, showing which node joins the tree each time and how the costs of the other nodes are updated.

MODEL QUESTION (6 Marks): All the routers in a network run a link state routing protocol and have flooded their link state packets, so every router holds the same link state database. The database corresponds to the network shown in the figure, with the numbers on the links denoting the link costs. Using Dijkstra's algorithm, find the least-cost path from router F to every other router, and write the routing table (destination, cost, next hop) of F.



Solution

1. **Given:** the link costs read from the figure in the model paper: A–B 3, A–D 5, B–C 3, B–E 1, C–F 4, C–G 2, D–E 4, E–F 3, F–G 4. The root is **F**. *(Please check this - method is the same for any values.)*
2. **Required:** least cost and path from F to A, B, C, D, E, G, and the routing table of F.
3. **Initialisation:** Tree = {F}. D[F] = 0. Neighbours of F: C = 4, E = 3, G = 4. Others = infinity.
4. **Iterations** (D[x] = min{D[x], D[w] + c[w][x]}):

Iteration table:

Step	Selected node	A	B	C	D	E	G
Init	F (0)	∞	∞	4	∞	3	4
1	E (3)	∞	4	4	7	3	4
2	C (4)	∞	4	4	7	3	4
3	B (4)	7	4	4	7	3	4
4	G (4)	7	4	4	7	3	4
5	A (7)	7	4	4	7	3	4
6	D (7)	7	4	4	7	3	4

Distance updates at each iteration:

- Iteration 1 – Select E (3):
- B = min(∞, 3 + 1) = 4
- D = min(∞, 3 + 4) = 7

- Iteration 2 – Select C (4):
- B = min(4, 4 + 3) = 4 (No change)
- G = min(4, 4 + 2) = 4 (No change)

- Iteration 3 – Select B (4):
- A = min(∞, 4 + 3) = 7

- Iteration 4 – Select G (4):
- No change.

- Iteration 5 – Select A (7):
- D = min(7, 7 + 5) = 7 (No change)

- Iteration 6 – Select D (7):
- All nodes have been visited. Algorithm terminates.

Note: C, B and G have equal tentative distances of 4. Any of these nodes can be selected first without changing the final shortest-path distances.

Least-cost paths from F:

Destination	Least-cost path	Total cost
A	F → E → B → A	7
B	F → E → B	4
C	F → C	4
D	F → E → D	7
E	F → E	3
G	F → G	4

Routing table of F:

The next hop is the first router encountered when travelling from source F towards the destination.

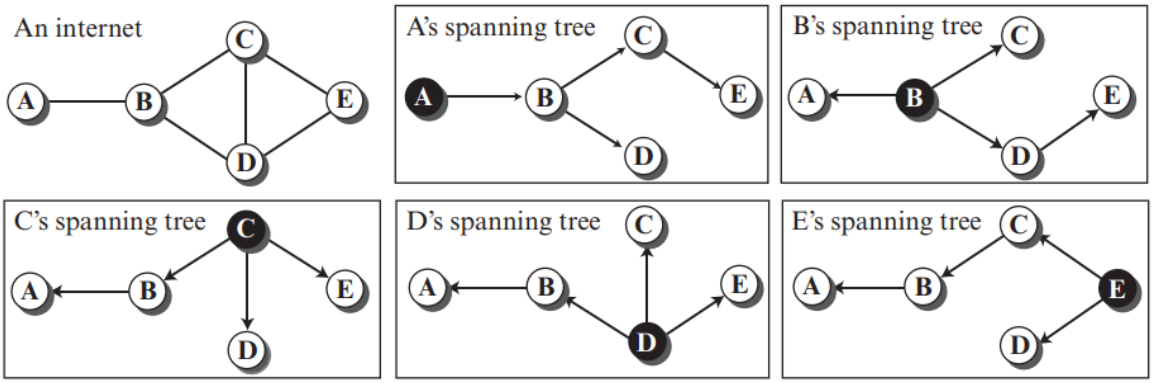
Destination	Cost	Next hop
A	7	E
B	4	E
C	4	C
D	7	E
E	3	E
G	4	G

Final Answer: The least-cost routing table of router F is obtained as shown above.

3. Path-Vector (PV) Routing

Least-cost routing (DV, LS) does not let a source apply a **policy**, for example avoiding routers of a commercial rival or an insecure organisation. Sometimes the goal is just **reachability**. **Path-vector routing** is not based on least cost: the **best route is chosen by the source using its policy**. It is mostly designed for routing between **ISPs**.

Spanning trees: The path from a source to all destinations is the **best spanning tree**, decided by the source's policy. Common policies: use the **minimum number of nodes**; **avoid some nodes** as middle nodes.



About Figure: A five-node internet and the spanning tree chosen by each of A, B, C, D and E under the policy of fewest nodes while avoiding certain middle nodes (D for A and E, C for B).

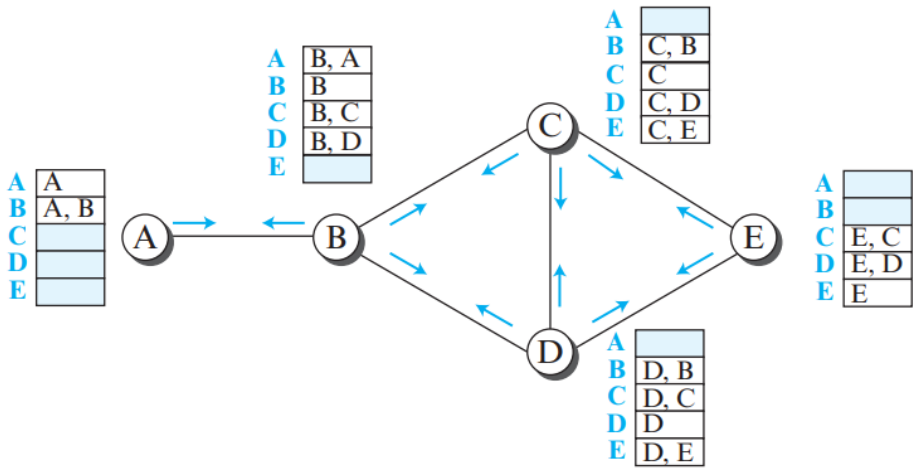
Creation of path vectors: Like DV, it is **asynchronous and distributed**.

- At boot, each node builds a **path vector** from its immediate neighbours (for example C: C, C–B, C –D, C–E) and sends it to its neighbours.
- On receiving a neighbour's vector, a node updates its own using a Bellman-Ford-like equation with **its own policy** in place of least cost.

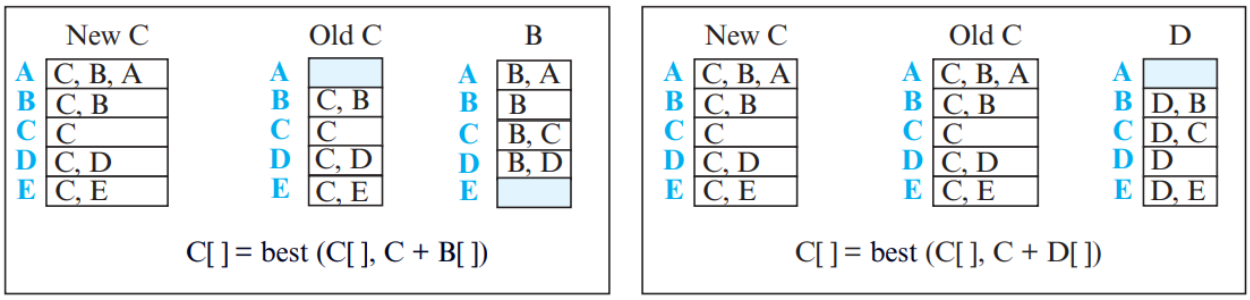
$$\text{Path-vector update, } \text{Path}(x,y) = \text{best Path}(x,y),[x + \text{Path}(v,y)]$$

Where the operator "+" means **adding x to the beginning** of the path, and v is a neighbour. A node must not add itself to an **empty** path (an empty path means no path exists).

Loop prevention: if Path(v, y) **already includes x**, the path is **discarded**, so x never visits itself.



About Figure (ABOVE FIG): The initial path vector of each node (for example A: A–B; B: B–A, B–C, B –D) and the arrows showing each is sent to its neighbours.



Event 1: C receives a copy of B's vector

Event 2: C receives a copy of D's vector

Note:
X [ ]: vector X
Y: node Y

About Figure: Node C's vector after two events. After receiving B's vector, C learns the path C–B–A to A. After receiving D's vector, no change.

Path-vector algorithm (pseudocode):

```
Path_Vector_Routing ( )
{
  // Initialization
  for (y = 1 to N)
  {
    if (y is myself)         Path[y] = myself
    else if (y is neighbor)  Path[y] = myself + neighbor node
    else                     Path[y] = empty
  }
  send vector {Path[1], ..., Path[N]} to all neighbors
  // Update
  repeat (forever)
  {
    wait (for a vector Pathw from a neighbor w)
    for (y = 1 to N)
    {
      if (Pathw includes myself) discard the path       // avoid loop
      else Path[y] = best { Path[y], (myself + Pathw[y]) }
    }
    if (there is a change in the vector)
      send vector {Path[1], ..., Path[N]} to all neighbors
  } }
```

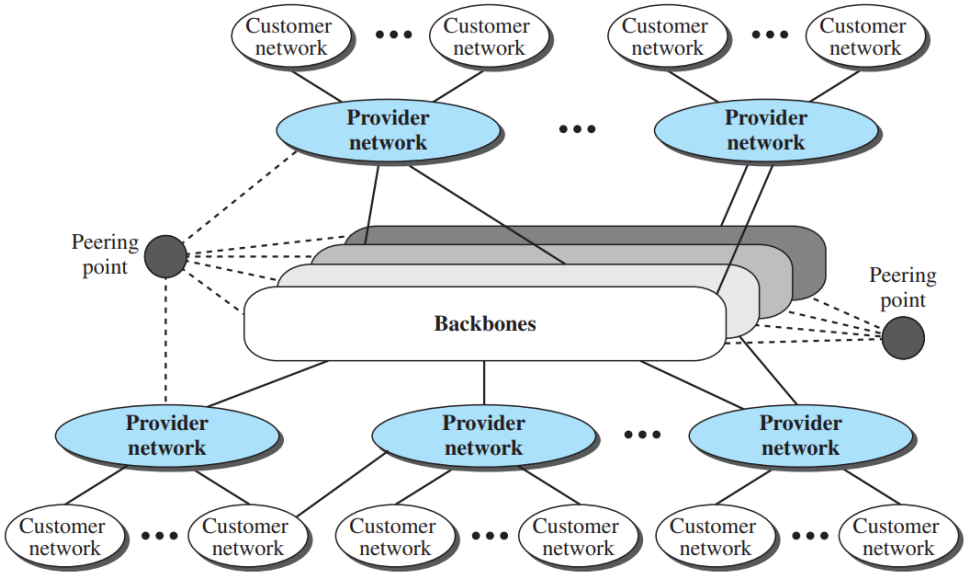
Parameter	Distance-vector	Link-state	Path-vector
Goal	Least cost	Least cost	**Policy** / reachability
Information exchanged	Distance vector (costs)	LSPs (link states of neighbours)	Path vectors (lists of nodes)
Sent to	Immediate neighbours	**All nodes** (flooding)	Immediate neighbours
Knowledge	Whole internet, but **incomplete**	**Complete** map (LSDB)	Paths to destinations
Computation	Bellman-Ford	**Dijkstra**	Bellman-Ford-like with policy
Main problem	**Count to infinity**, slow convergence	Heavy flooding traffic	Policy complexity
Loop prevention	Split horizon, poisoned reverse	Complete database	Path includes own node, so discard
Protocol	RIP	OSPF	BGP

Unicast Routing Protocols

A protocol is more than an algorithm. It defines its **domain of operation, the messages exchanged, communication between routers and interaction with protocols in other domains**. Three protocols are used: **RIP** (distance-vector), **OSPF** (link-state) and **BGP** (path-vector).

Internet Structure and Hierarchical Routing

The Internet is now **multi-backbone**: several private backbones joined at **peering points**, **provider networks** below them, and **customer networks** at the edge. Any of these can be called an **ISP**.



About Figure: Backbones are joined at peering points. Provider networks connect to the backbones, and customer networks connect to provider networks.

Why one routing protocol is not enough

- **Scalability:** forwarding tables become huge, searching is slow and updating creates a huge amount of traffic.
- **Administrative issue:** each ISP is run by an authority that wants control of its subnets, routers, routing algorithm and traffic policy.

Hierarchical routing: each ISP is an autonomous system (AS).

- **Intradomain (intra-AS) routing** or **interior gateway protocol (IGP)**: run inside each AS. Each AS may choose its own. Common: **RIP, OSPF**.
- **Interdomain (inter-AS) routing** or **exterior gateway protocol (EGP)**: runs globally to glue ASs together. Only one exists: **BGP**.

Autonomous systems: Each AS has a **16-bit autonomous system number (ASN)** given by ICANN. ASs are classified by how they connect to others.

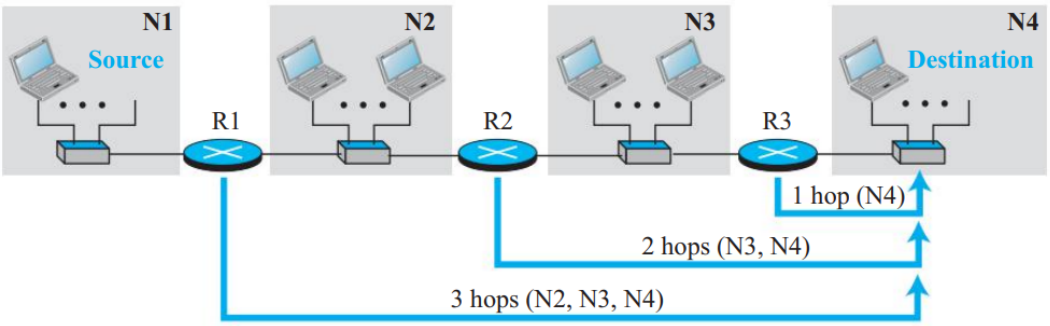
Type	Description	Example
Stub AS	**One** connection to another AS. Traffic can start or end here but **cannot pass through**	Customer network
Multihomed AS	**More than one** connection, but does **not allow** traffic to pass through	Customer AS using several providers
Transient AS	Connected to more than one AS and **allows traffic to pass through**	Provider networks, backbones

1. Routing Information Protocol (RIP)

RIP is a widely used **intradomain** protocol based on the **distance-vector** algorithm. It began in the Xerox Network System, and BSD UNIX made it widespread.

Hop count

- RIP routers advertise the cost of reaching **networks** (not routers). The cost is the **number of hops** (networks a packet crosses), independent of delay or bandwidth.
- The network of the source host is **not counted**.
- **Maximum cost is 15; 16 means infinity.** So RIP suits ASs with a **diameter not above 15 hops**.



About Figure: A source in N1 reaches N4 through routers R1, R2, R3. R1 advertises 1 hop for N4 as seen from R3, R2 advertises 2 hops (N3, N4), and R1 advertises 3 hops (N2, N3, N4).

Forwarding table: It has **three columns**: **destination network, next router, cost in hops**. The first and third columns together convey a distance vector. The next-router column gives the whole tree (by the second tree property). The cost column is not needed for forwarding but is needed for **updating**.

Forwarding table for R1

Destination network	Next router	Cost in hops
N1	——	1
N2	——	1
N3	R2	2
N4	R2	3

Forwarding table for R2

Destination network	Next router	Cost in hops
N1	R1	2
N2	——	1
N3	——	1
N4	R3	2

Forwarding table for R3

Destination network	Next router	Cost in hops
N1	R2	3
N2	R2	2
N3	——	1
N4	——	1

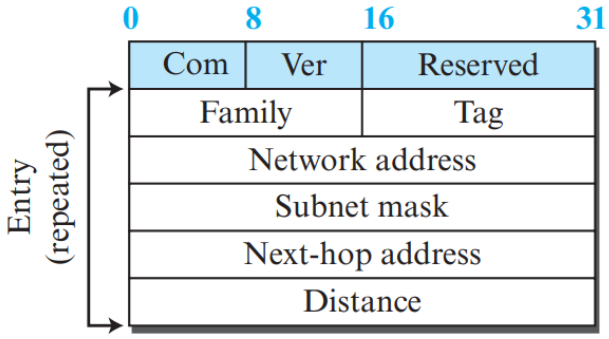
About Figure: The forwarding tables of R1, R2 and R3 for networks N1 to N4, each with destination, next router and hop count (for example R1: N3 via R2, 2 hops; N4 via R2, 3 hops).

Implementation

- RIP runs as a **process over UDP, well-known port 520**. In BSD it is a daemon named **routed**.
- RIP messages are inside UDP, inside IP. It runs at the application layer but builds forwarding tables for IP at the network layer.
- Versions: RIP-1 and **RIP-2** (backward compatible, uses extra fields; only RIP-2 is studied).

RIP messages. Two types:

- **Request:** sent by a router that has just come up or has expired entries; can ask for specific or all entries.
- **Response (update): solicited** (in reply to a request) or **unsolicited** (sent **periodically every 30 seconds** or when the table changes).



Fields

Com: Command, request (1), response (2)
Ver: Version, current version is 2
Family: Family of protocol, for TCP/IP value is 2
Tag: Information about autonomous system
Network address: Destination address
Subnet mask: Prefix length
Next-hop address: Address length
Distance: Number of hops to the destination

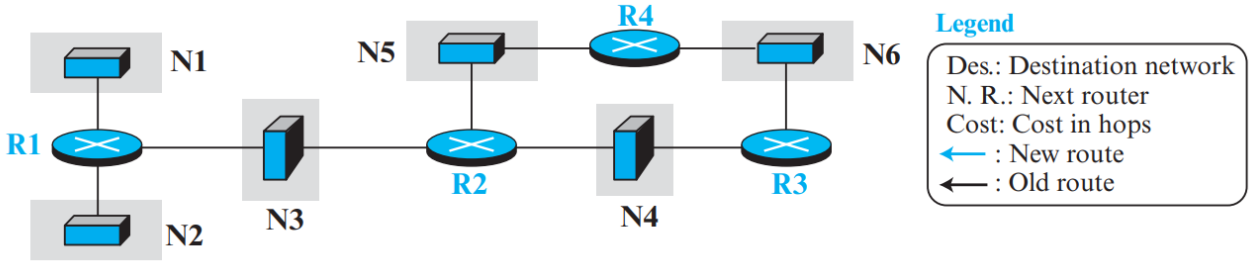
About Figure: The message has Com (command), Ver, Reserved, then a repeated entry with Family, Tag, Network address, Subnet mask, Next-hop address and Distance (number of hops).

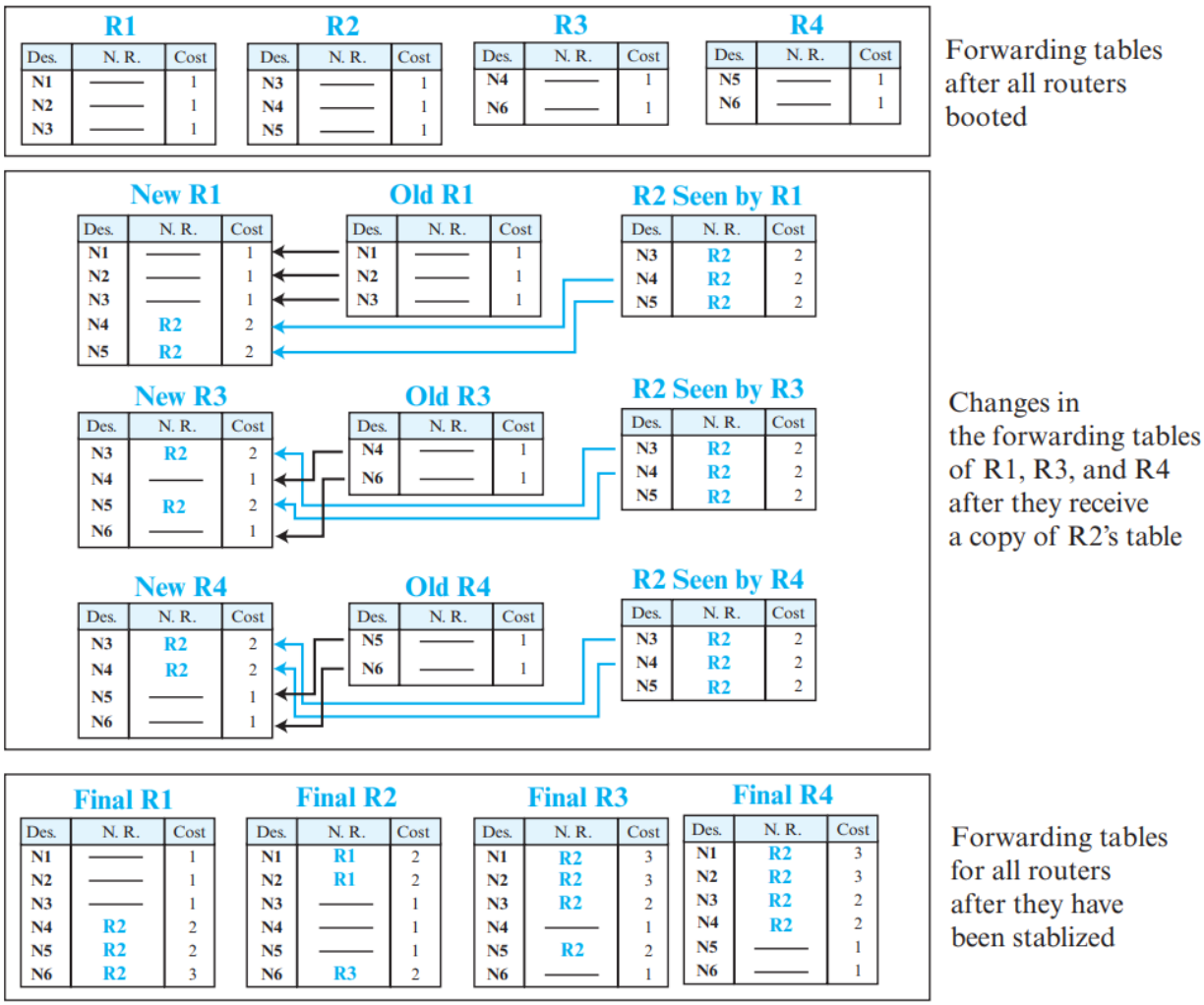
RIP algorithm

It is the DV algorithm with changes:

- A router sends the **whole contents of its forwarding table** in a response message.
- The receiver **adds one hop** to each cost and sets the **next router to the sender**. Each such route is a **received route**; each route in the old table is an **old route**.
- The old route is kept except in three cases:
  1. The received route is **not in the old table**: add it.
  2. The received route has a **lower cost**: select it.
  3. The received route has a **higher cost, but the same next router**: select it. (The neighbour previously advertised the route and the situation has changed. For example, it now advertises infinity (16) because the path is gone, and this must not be ignored.)
- The new table is **sorted by destination** (longest prefix first).

Example: all routers first hold only directly connected networks. After R2's table reaches R1, R3 and R4, they add routes through R2 (for example R1 learns N4, N5? via R2 with updated costs). After several exchanges the tables **stabilise**.





About Figure: An AS with networks N1 to N6 and routers R1 to R4 showing the tables after boot, the changes after copies of R2's table are received, and the final stabilised tables.

RIP timers

Timer	Value	Purpose
Periodic timer	Random between **25 and 35 s** (prevents all routers sending at once)	Controls regular update messages
Expiration timer	**180 s** per route, reset on each update	If no update arrives, the route is expired and its hop count set to **16**
Garbage-collection timer	**120 s**	An invalid route is still advertised with metric 16 for this time so neighbours learn it is invalid, then it is purged

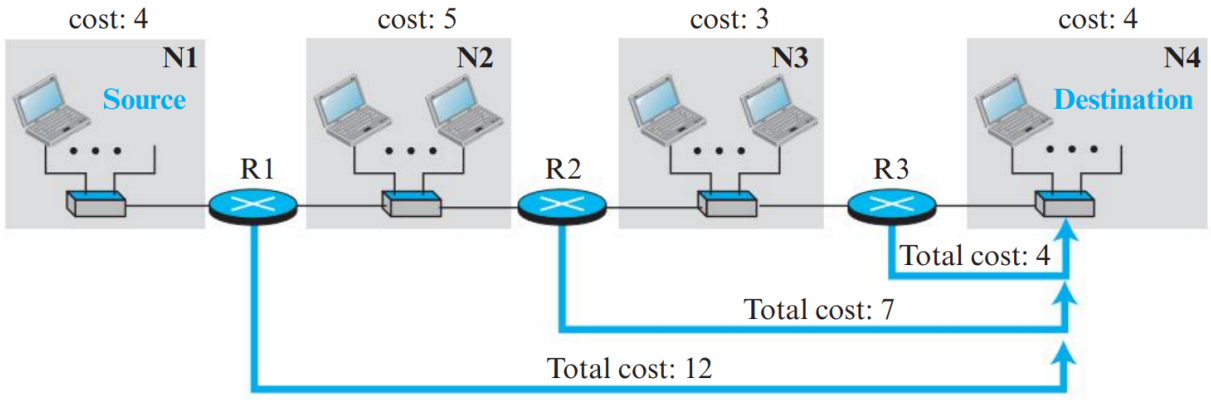
Performance

- **Updates:** simple format, sent only to neighbours, so little traffic.
- **Convergence:** slow in large domains, but the 15-hop limit makes it acceptable. Count-to-infinity and loops can slow it; **split horizon** and **poison reverse** help.
- **Robustness:** a failure or corruption in one router **propagates to all**, because each depends on its neighbours.

2. Open Shortest Path First (OSPF)

OSPF is an **intradomain** protocol based on **link-state routing**. It is an **open** protocol (its specification is public).

Metric: The cost is from the router to the **destination network**, but each link can be given a **weight** based on throughput, round-trip time, reliability and so on (or hop count). **Different types of service (TOS)** can have different weights.



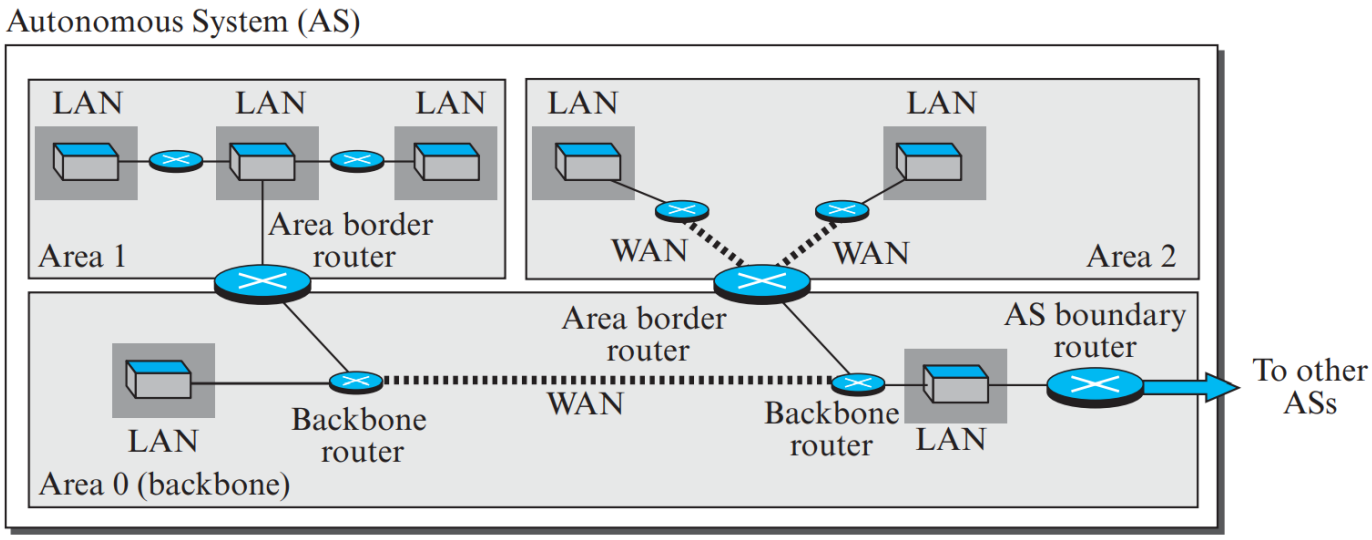
About Figure: Routers R1, R2, R3 and networks N1 to N4 with link costs 4, 5, 4 and 3. The total costs from R1 are 4 (N2), 9?? as shown in the book, and so on, summed along the path.

Forwarding table:

Each router builds a **shortest-path tree with Dijkstra's algorithm** and makes the forwarding table from it. The table is the same as for RIP except for the **cost values** (with hop-count cost they would be identical).

Areas

- In a large AS, flooding LSPs to all routers would create **huge traffic**. So the AS is divided into **areas**: each area is a small independent domain for flooding. This adds a second level of hierarchy (AS, then area).
- One area is the **backbone area** (**area identification 0**). Its routers pass information between areas, so every router receives the LSPs of other areas.

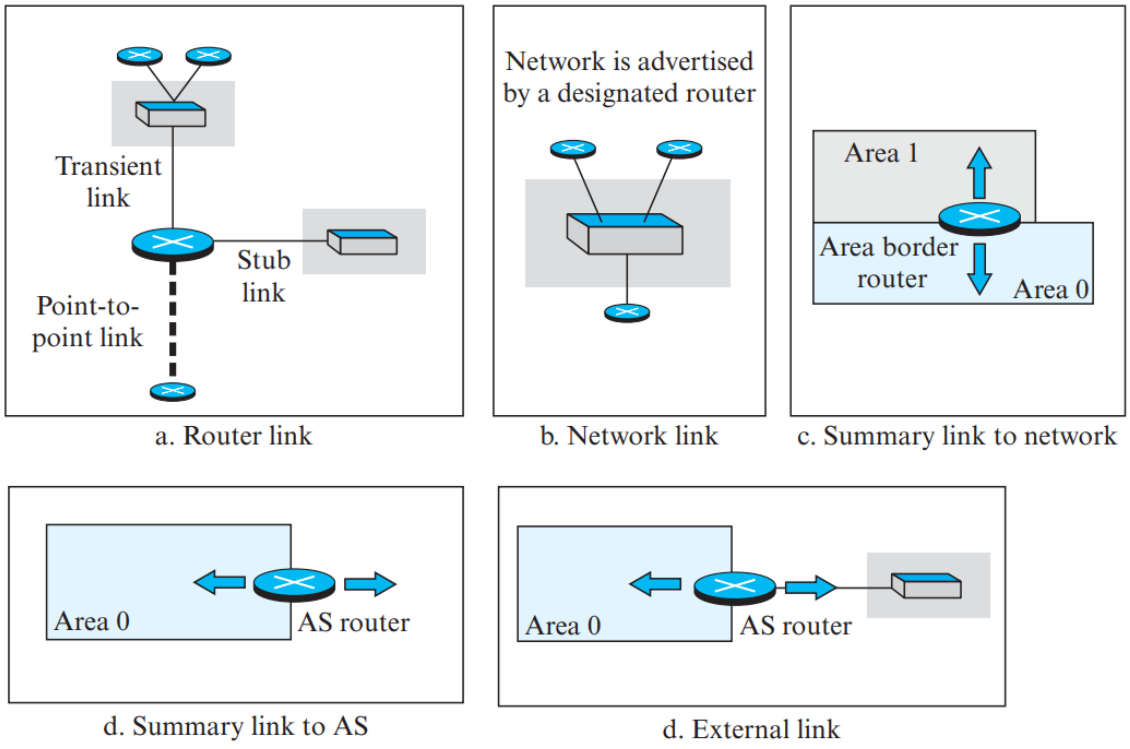


About Figure: An AS with Area 0 (backbone) joined to Area 1 and Area 2 through area border routers. Backbone routers and an AS boundary router (connected to other ASs) are marked.

Link-state advertisements

Real networks need different entities as nodes, different link types and costs.
OSPF defines **five types**:

LSA	Advertised by	Purpose
Router link	A router	Advertises a router as a node and its links: **transient** link (to a transient network, with address and cost), **stub** link (to a stub network, with address and cost), **point-to-point** link (address of the router at the other end and cost)
Network link	A **designated router** for the network	Advertises the **network as a node** (a network is passive and cannot advertise itself). Gives the addresses of all routers on it. **No cost**, since each router gives its cost in its router link
Summary link to network	**Area border router**	Advertises a summary of links collected by the backbone to an area, or collected by an area to the backbone (glues the areas)
Summary link to AS	**AS boundary router**	Summarises links from other ASs to the backbone of this AS
External link	AS boundary router	Announces a **single network outside the AS** to the backbone



a. Router link
b. Network link
c. Summary link to network
d. Summary link to AS
d. External link

About Figure: Five panels showing a router link (transient, stub and point-to-point links), a network link advertised by a designated router, a summary link to a network from an area border router, a summary link to an AS, and an external link from AS routers.

Implementation

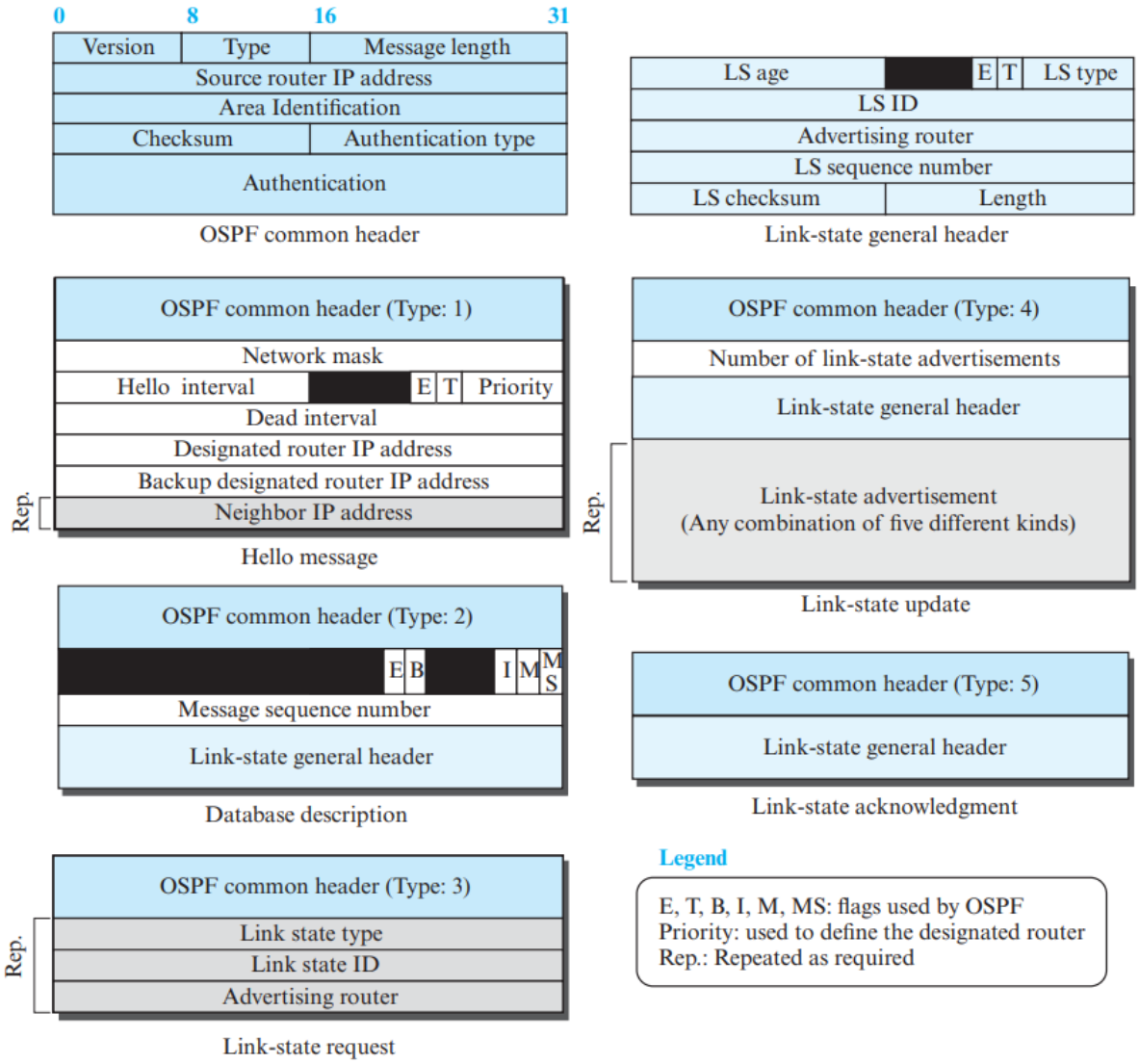
- OSPF is a **network-layer program** that uses IP. An IP datagram carrying OSPF has the **protocol field = 89**.
- Versions 1 and 2 exist; version 2 is mostly used.

OSPF messages

five types, with a common header and a link-state general header:

Type	Message	Use
1	**Hello**	A router introduces itself to neighbours and lists neighbours it knows
2	**Database description**	Normally in response to a hello; lets a new router get the full LSDB
3	**Link-state request**	Asks for information about a specific link state
4	**Link-state update**	The main message for building the LSDB; five versions (the five LSAs)
5	**Link-state acknowledgment**	Gives **reliability**: each update must be acknowledged

- **Authentication:** the common header has a field for authenticating the sender, to stop a malicious entity from joining the routing system.
- The algorithm is link-state routing plus creating the forwarding table from the shortest-path tree and handling all five messages.



About Figure: The OSPF common header (version, type, message length, source router IP, area identification, checksum, authentication type, authentication) and the layouts of the hello, database description, link-state request, update and acknowledgment messages.

Performance

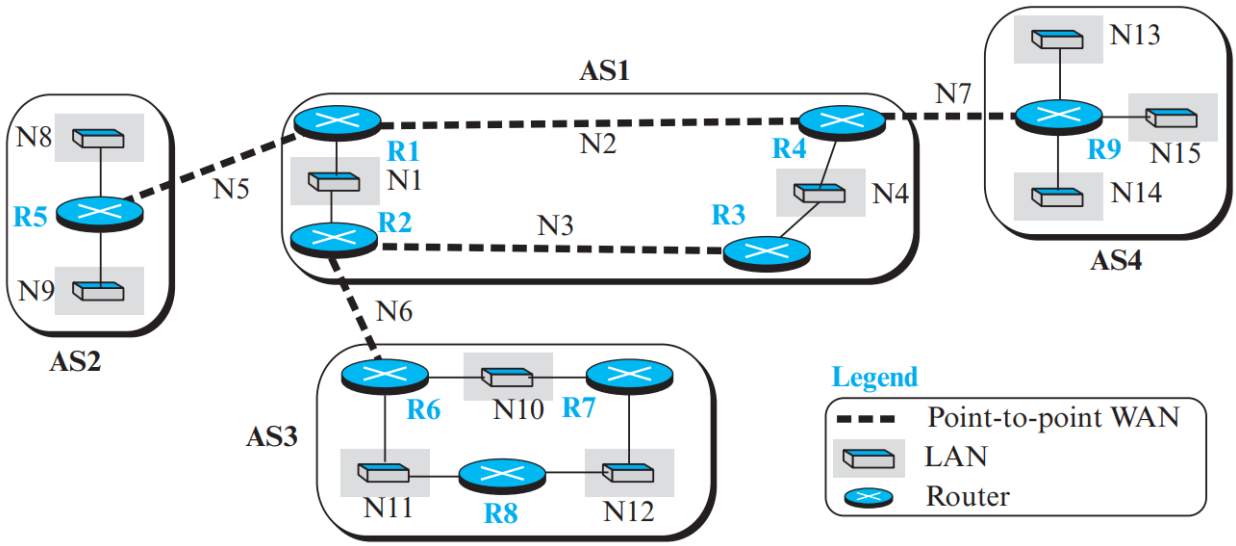
- **Update messages:** complex format and flooded to the whole area, which may use a lot of bandwidth in a large area.
- **Convergence:** fairly quick after flooding, but each router must run Dijkstra's algorithm.
- **Robustness:** better than RIP. After getting the full LSDB, each router is independent, so one failing router does not harm others seriously.

Parameter	RIP	OSPF
Algorithm	Distance-vector	Link-state (Dijkstra)
Metric	Hop count (max 15)	Weighted cost (throughput, RTT, reliability), per TOS
Runs over	**UDP**, port 520	**IP**, protocol 89
Exchanged	Whole forwarding table with neighbours	LSPs flooded to the area
Suitable AS size	Small	Small or large (uses areas)
Robustness	Failure propagates	More robust
Problems	Slow convergence, count to infinity	Flooding traffic, Dijkstra computation

3. Border Gateway Protocol Version 4 (BGP4)

BGP4 is the **only interdomain routing protocol** of the Internet. It is based on **path-vector** routing but tailored to give **reachability** of networks.

eBGP and iBGP. Take four ASs: AS1 is transient, AS2, AS3, AS4 are stub. Each AS runs RIP or OSPF inside. A router knows only networks of its own AS.

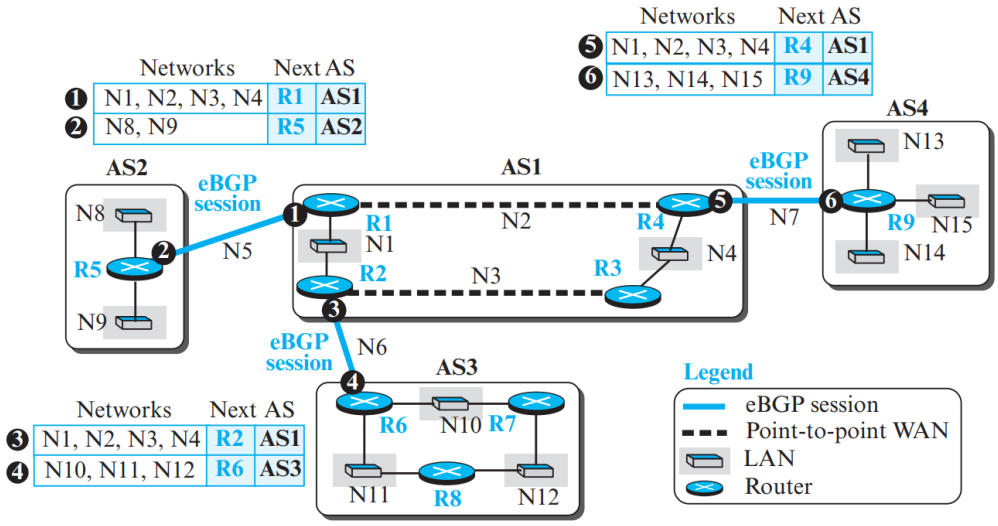


About Figure: AS1 (routers R1 to R4 with networks N1 to N4) is connected to AS2 (R5), AS3 (R6, R7, R8) and AS4 (R9) through point-to-point WANs N5, N6 and N7.

- **External BGP (eBGP)** is installed on each **border router** (the edge router connected to another AS).
- **Internal BGP (iBGP)** is installed on **all routers**.
- So border routers run three protocols (intradomain, eBGP, iBGP); other routers run two (intradomain, iBGP).

eBGP operation

- Two BGP routers (**BGP peers or speakers**) create a **TCP connection on well-known port 179** and exchange messages. Each logical connection is a **session**.
- eBGP pairs here: **R1–R5, R2–R6, R4–R9**, over the physical WANs N5, N6, N7.
- Update messages announce reachability. For example R1 tells R5 that **N1, N2, N3, N4 can be reached through R1**. R5 adds this to its table.



About Figure: Three eBGP sessions (R1–R5, R2–R6, R4–R9) with simplified update messages. Each message lists the networks, the AS and the next router.

Two problems remain after eBGP:

1. Border routers do not know how to reach **non-neighbour ASs** (R5 does not know AS3 or AS4).
2. **Non-border routers** know nothing about other ASs.

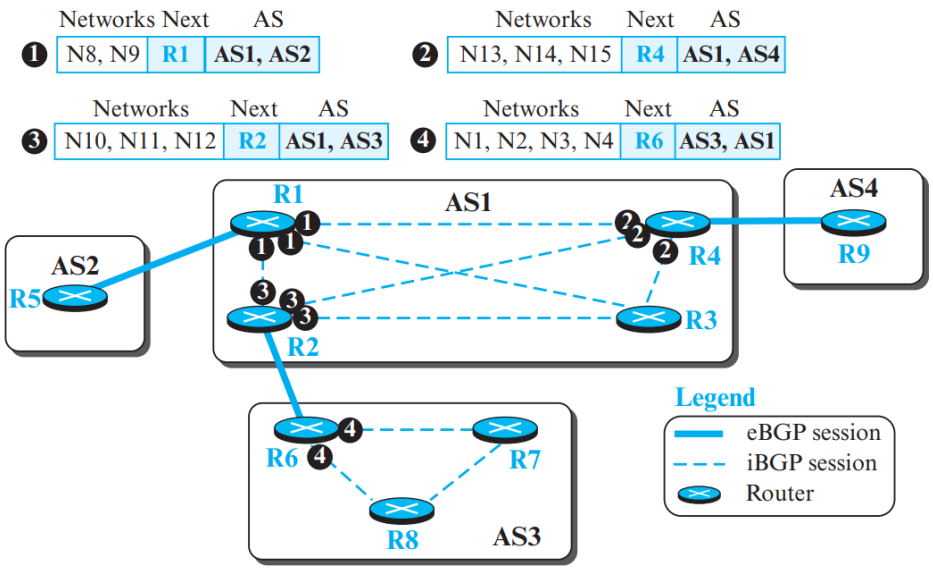
These are solved by **iBGP**.

iBGP operation

- It also uses **TCP port 179**, but creates a session between **every pair of routers inside an AS**.
- If an AS has only one router, no iBGP session exists (AS2 and AS4).
- With n routers, there are **n(n − 1)/2** sessions (a full mesh), so each router advertises only its own reachability and loops are avoided.

$$\text{Number of iBGP sessions, } S = \frac{n(n-1)}{2}$$

Where S = number of iBGP sessions in an AS and n = number of routers in that AS (for example AS1 with 4 routers: S = 6).



About Figure: The eBGP sessions between the ASs and iBGP sessions between all routers inside AS1 and AS3, with the first four update messages (for example R1 announces N8, N9 via AS1–AS2 to R2, R3, R4).

After the updates have passed through all ASs, each router combines eBGP and iBGP information into a **path table** after applying policy. For example, router R1's path table says N8, N9 go through **AS1, AS2**, next router **R5**.

Networks	Next	Path
N8, N9	R5	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

Path table for R1

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R6	AS1, AS3
N13, N14, N15	R1	AS1, AS4

Path table for R2

Networks	Next	Path
N8, N9	R2	AS1, AS2
N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4

Path table for R3

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R1	AS1, AS3
N13, N14, N15	R9	AS1, AS4

Path table for R4

Networks	Next	Path
N1, N2, N3, N4	R1	AS2, AS1
N10, N11, N12	R1	AS2, AS1, AS3
N13, N14, N15	R1	AS2, AS1, AS4

Path table for R5

Networks	Next	Path
N1, N2, N3, N4	R2	AS3, AS1
N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R2	AS3, AS1, AS4

Path table for R6

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R7

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R8

Networks	Next	Path
N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R4	AS4, AS1, AS2
N10, N11, N12	R4	AS4, AS1, AS3

Path table for R9

About Figure: The final path table of each router (R1 to R9), listing networks, the AS path and the next router.

Injection of information into intradomain routing

Path tables are **not used for forwarding themselves**. They are injected into the **RIP or OSPF forwarding tables**.

- **Stub AS:** the border router adds a **default entry** whose next router is the eBGP peer (for example R5 uses R1 as default).
- **Transient AS:** the **whole path table** is injected into the intradomain table of each router.
- **Cost:** foreign networks are given the cost of reaching the **first AS in the path** (taken from the intradomain table).

Des.	Next	Cost
N1	—	1
N4	R4	2
N8	R5	1
N9	R5	1
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R1

Des.	Next	Cost
N1	—	1
N4	R3	2
N8	R1	2
N9	R1	2
N10	R6	1
N11	R6	1
N12	R6	1
N13	R3	3
N14	R3	3
N15	R3	3

Table for R2

Des.	Next	Cost
N1	R2	2
N4	—	1
N8	R2	3
N9	R2	3
N10	R2	2
N11	R2	2
N12	R2	2
N13	R4	2
N14	R4	2
N15	R4	2

Table for R3

Des.	Next	Cost
N1	R1	2
N4	—	1
N8	R1	2
N9	R1	2
N10	R3	3
N11	R3	3
N12	R3	3
N13	R9	1
N14	R9	1
N15	R9	1

Table for R4

Des.	Next	Cost
N8	—	1
N9	—	1
0	R1	1

Table for R5

Des.	Next	Cost
N10	—	1
N11	—	1
N12	R7	2
0	R2	1

Table for R6

Des.	Next	Cost
N10	—	1
N11	R6	2
N12	—	1
0	R6	2

Table for R7

Des.	Next	Cost
N10	R6	2
N11	—	1
N12	—	1
0	R6	2

Table for R8

Des.	Next	Cost
N13	—	1
N14	—	1
N15	—	1
0	R4	1

Table for R9

About Figure: The forwarding tables of R1 to R9 after BGP information is added (shaded). Stub routers have default entries (shown as 0), and transient routers have entries for all foreign networks.

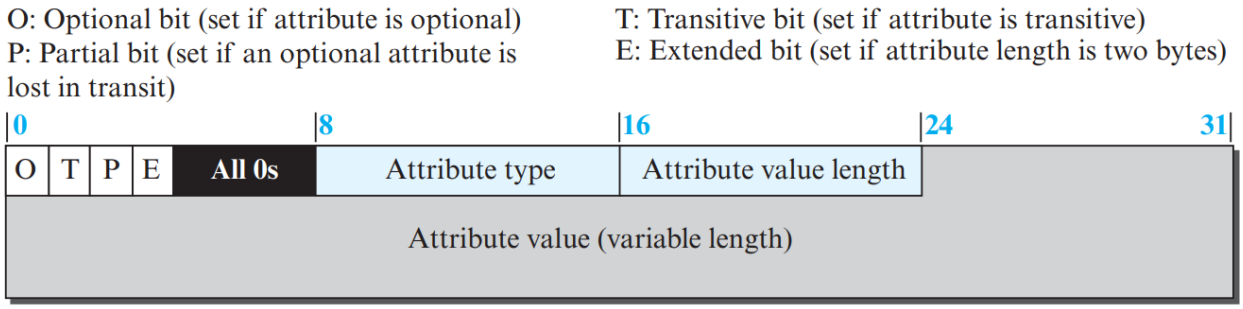
Address aggregation

BGP4 uses **prefixes** as destination identifiers and allows **aggregation**. For example 14.18.20.0/26, 14.18.20.64/26, 14.18.20.128/26 and 14.18.20.192/26 combine into **14.18.20.0/24**. The **longest-prefix** rule handles exceptions.

Path attributes

A destination in BGP has up to **seven path attributes**.

- **Well-known** attributes must be recognised by all routers. They are **mandatory** (always present in an update) or **discretionary**.
- **Optional** attributes need not be recognised. They are **transitive** (can pass to the next AS) or **intransitive**.



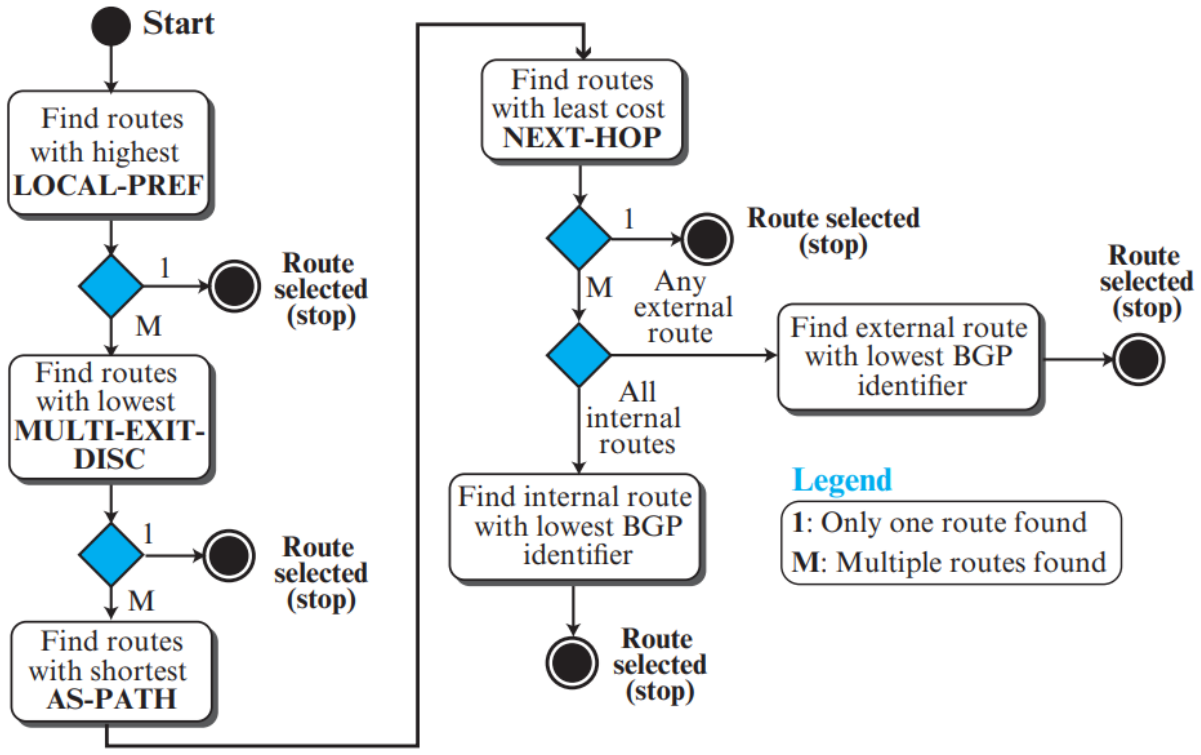
About Figure: The attribute starts with flag bits (O optional, T transitive, P partial, E extended), then the attribute type, the value length and the variable-length value.

Attribute	Type	Category	Meaning
ORIGIN	1	Well-known mandatory	Source of the routing information: 1 = intradomain protocol, 2 = BGP, 3 = unknown
AS-PATH	2	Well-known mandatory	List of ASs through which the destination is reached. **Prevents loops** (a path that includes the current AS is dropped); used in route selection
NEXT-HOP	3	Well-known mandatory	Next router for the data packet; used to inject information into RIP/OSPF
MULT-EXIT-DISC	4	Optional **nontransitive**	Discriminates among multiple exit paths to a destination. The **lowest value** wins. Not passed to another AS
LOCAL-PREF	5	Well-known discretionary	Set by the administrator by **policy**. A **higher value** is preferred (for example 400 for the most secure path)
ATOMIC-AGGREGATE	6	Well-known discretionary	The prefix is **not** an aggregate (a single network). No value field
AGGREGATOR	7	Optional transitive	The prefix **is** an aggregate. Gives the last AS number that aggregated and the router's IP address

Route selection

When several routes to a destination exist, BGP selects one in these steps. The router extracts the routes meeting each criterion; if only one is left it is selected, otherwise it goes on to the next step.

1. Highest **LOCAL-PREF** (policy first).
2. Shortest **AS-PATH**.
3. Lowest **MULT-EXIT-DISC**.
4. Least-cost **NEXT-HOP** (from the intradomain table).
5. If all are internal routes: the internal route with the lowest **BGP identifier**; otherwise prefer an **external** route with the lowest BGP identifier.

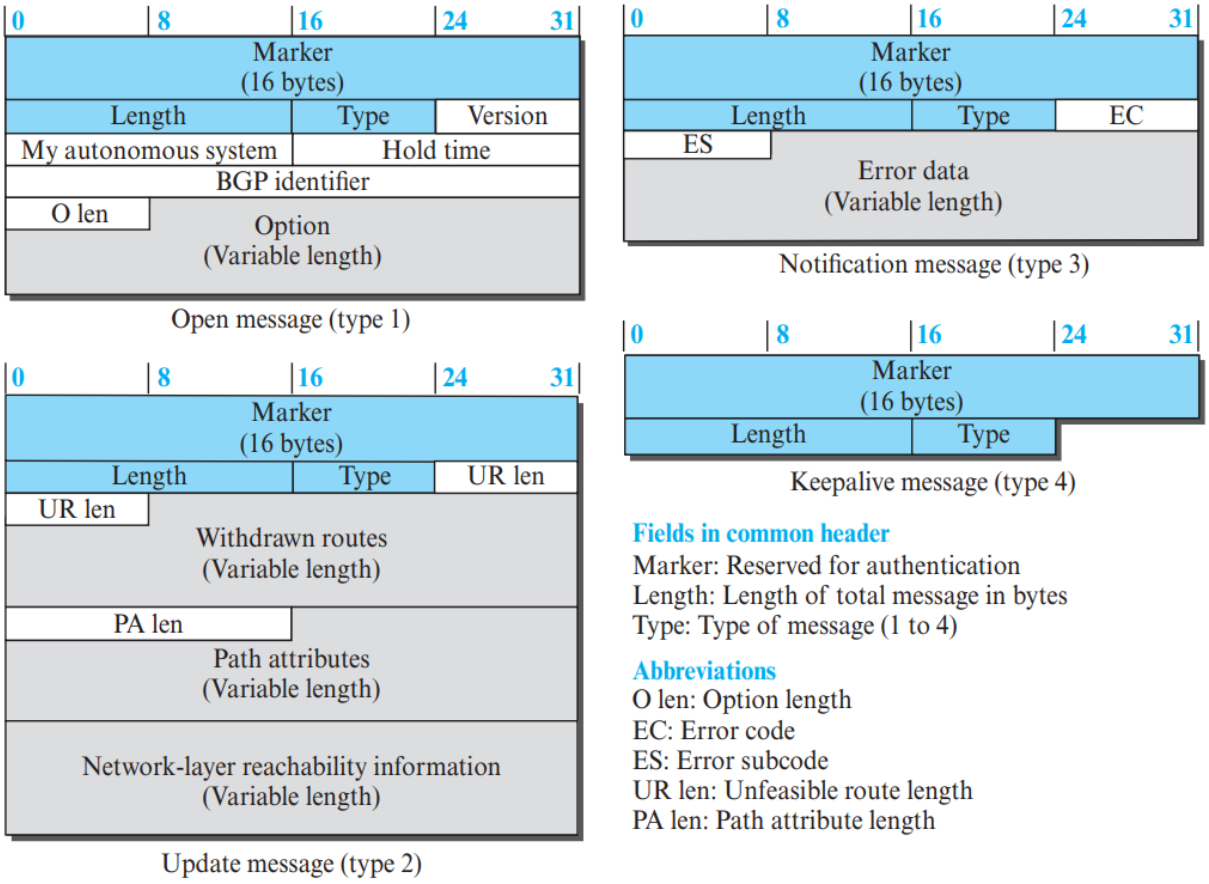


About Figure: A flow chart: find routes with highest LOCAL-PREF, then shortest AS-PATH, then lowest MULT-EXIT-DISC, then least-cost NEXT-HOP, then an external route with lowest BGP identifier, or an internal route with lowest BGP identifier. At each stage, one route found stops the process.

BGP messages

four types, all with a common header: marker, length, type:

Type	Message	Purpose
1	**Open**	Starts a neighbour relationship (a TCP connection is opened first)
2	**Update**	The heart of BGP: **withdraws** destinations advertised earlier, **announces** a route to a new destination, or both. It can withdraw several destinations but announce only one new destination (or several with the same path attributes)
3	**Notification**	Sent on an error condition or to close the session
4	**Keepalive**	Exchanged regularly (before the hold time expires) to show peers are alive



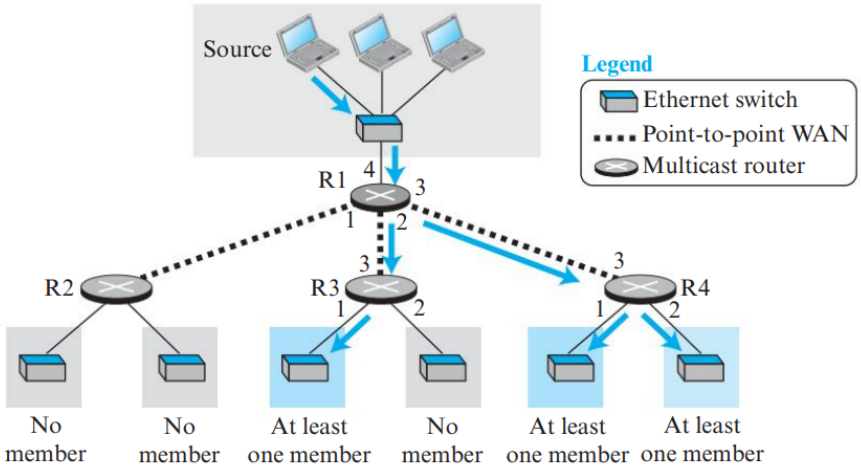
About Figure: The common header (marker, length, type) and the formats of the open, update, notification and keepalive messages.

Performance: BGP exchanges many messages to build tables but is **free from loops and count-to-infinity**. The weakness of propagating a failure or corruption (as in RIP) also exists.

Multicast Routing

Unicasting, Multicasting and Broadcasting

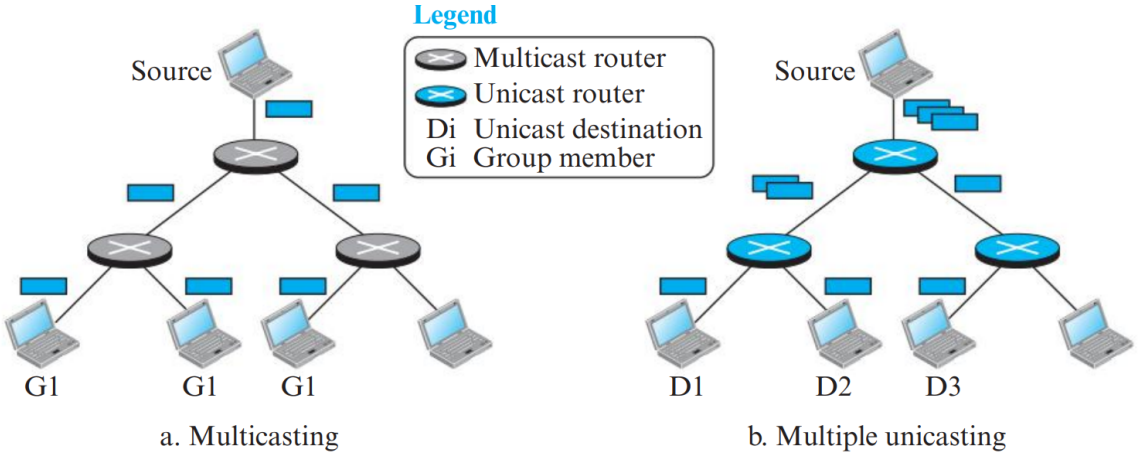
Type	Relationship	Description
Unicasting	One source, **one destination network** (one-to-one)	Each router forwards the packet through **only one** interface
Multicasting	One source, **a group of destinations** (one-to-many)	Source address is unicast; destination address is a **group address**. A router may send copies through **more than one** interface
Broadcasting	One-to-all	Not provided at Internet level (huge traffic). **Controlled broadcasting** is used inside a domain, mostly as a step in multicasting



About Figure: The same small internet with multicast routers. R1 sends the datagram through interfaces 2 and 3, R4 through both its interfaces, and R3 only through interface 1 (no members on interface 2).

Multicasting versus multiple unicasting

- **Multicasting:** one packet from the source is **duplicated by routers**. All copies have the **same destination address**. Only **one copy** travels between any two routers.
- **Multiple unicasting:** the source creates **several packets**, each with a **different unicast destination**. Several copies may travel on the same link (for example an e-mail sent to a group).



a. Multicasting b. Multiple unicasting

About Figure: (a) Multicasting: one packet from the source is copied by multicast routers towards group members G1. (b) Multiple unicasting: the source sends separate packets to each unicast destination D1, D2, D3.

Why not emulate multicasting with unicasting?

1. Multicasting is **more efficient**: it needs less bandwidth.
2. In multiple unicasting, the source creates packets with a **delay between them**; for 1,000 destinations the delay between first and last may be unacceptable.

Applications: access to **distributed databases**, **information dissemination** (for example software updates, news), **teleconferencing**, **distance learning**.

Multicasting Basics

Multicast addresses

- A multicast address identifies a **group**, not a single host. A host in n groups has **n + 1** addresses (one unicast and n multicast).
- In IPv4 the multicast block is **224.0.0.0/4** (224.0.0.0 to 239.255.255.255), with a **28-bit group identifier**.

Number of addresses in the IPv4 multicast block, $N = 2^{28}$

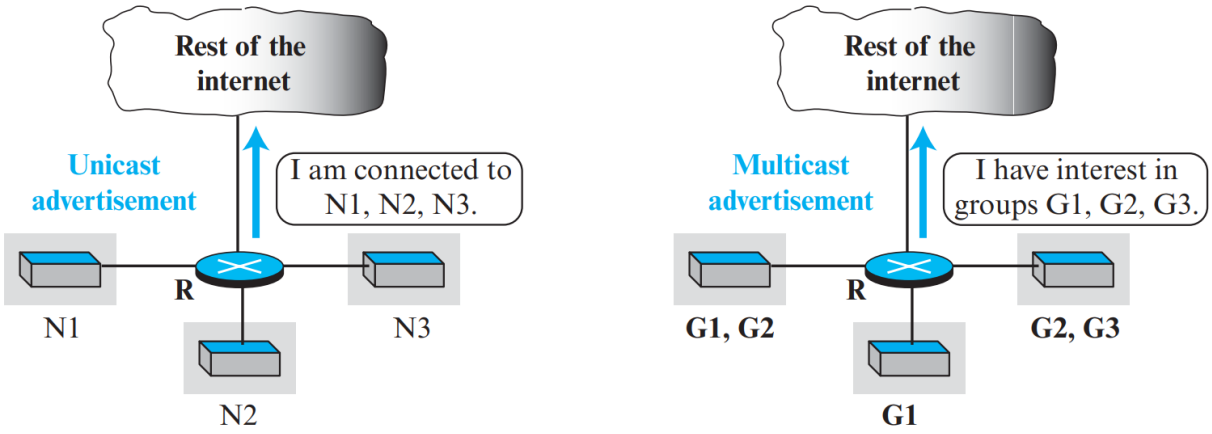
Subblock	Use
224.0.0.0/24 (local network control)	For protocols inside a network; **not forwarded** by routers. 224.0.0.1 = all hosts and routers; 224.0.0.2 = all routers
224.0.1.0/24 (internetwork control)	For protocols across the Internet; **can be forwarded**
232.0.0.0/8 (SSM)	**Source-specific multicast**: routing based on group and source
233.0.0.0/8 (GLOP)	Inside an AS: AS number x.y gives 233.x.y.0 to 233.x.y.255 (256 addresses)
239.0.0.0/8 (administratively scoped)	Restricted to an **organisation**; packets must not leave it

Selecting a multicast address

- **Limited group** (inside an AS): use the **AS number** to pick a free address in GLOP or administratively scoped block (for example AS 23452 = (91.156) in base 256 gives 233.91.156.0 to 233.91.156.255).
- **Larger group** (beyond one AS): choose from the **SSM block 232.0.0.0/8**. No permission needed, since routing uses group and source together.

Collecting information about groups

In unicast routing, a router **automatically** knows its connected networks. In multicast routing it **cannot**: (1) group membership has no relation to the network prefix, and (2) membership changes quickly (hosts join and leave). So a router needs **IGMP** to collect membership, and then a routing protocol to propagate it.



About Figure: A router R with networks N1, N2, N3 advertises "I am connected to N1, N2, N3" in unicast. For multicast it must learn the groups (G1, G2, G3) that have members on its interfaces and advertise "I have interest in groups G1, G2, G3".

Internet Group Management Protocol (IGMP)

- A **network-layer auxiliary protocol** like ICMP. Its messages are encapsulated in IP datagrams.
- IGMPv3 has two message types: **query** and **report**.

Query message **(sent periodically by a router to hosts)**

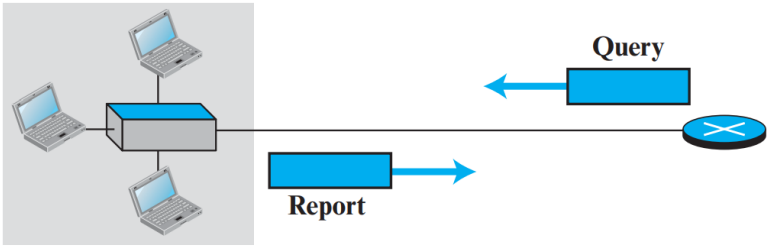
Form	Purpose	Destination address
General query	Asks about membership in **any** group	**224.0.0.1** (all hosts and routers); other routers on the network refrain from resending
Group-specific query	Asks about a **specific group**, when the router got no response and wants to be sure there are no members	That group's multicast address
Source-and-group-specific query	Asks about a group **from specific sources**	That group's multicast address

Report message **(sent by a host in response to a query)**

- Contains **records**: the group (multicast address) and the **sources** the host wants (inclusion) or does not want (exclusion) messages from.
- Sent to **224.0.0.22** (the multicast address of IGMPv3).

Join and leave

To **join**, a host waits for a query and then sends a report. To **leave**, it simply **does not respond**; if no other host responds, the group is purged from the router's database.



About Figure: A router sends a query to hosts on its network, and the hosts reply with reports.

Multicast forwarding differs from unicast forwarding in two ways:

1. **Number of interfaces:** a unicast packet goes out of **one** interface. A multicast packet may go out of **more than one**, because the group may have members in several parts of the internet.
2. **Basis of the decision:** unicast forwarding depends only on the **destination address**. Multicast forwarding depends on **both the destination (group) and the source address**. If the source is in a part of the internet that already has a group member, the router must **not send the packet back** that way (the member already has a copy).



a. Packet sent out of two interfaces b. Packet sent out of one interface

About Figure: (a) Source in a part with no group member: the router sends the packet out of two interfaces. (b) Source in a part containing a member: the router sends it out of only one interface.

Two approaches to multicast routing

Parameter	Source-based tree	Group-shared tree
Tree	A **separate tree for each source-group combination** (source is the root, members are the leaves)	One tree per group with a **core (rendezvous-point) router** as the root
Number of trees	**m** × **n** (m groups, n sources)	**m** (one per group)
Delivery	Directly from the source	In two steps: source **unicasts** (by **tunneling**: multicast packet encapsulated in a unicast packet) to the core router, which **decapsulates and multicasts**
Overhead	Many trees to store	Needs an algorithm to **select the core router** for each group
Protocols	DVMRP, MOSPF, PIM-DM	PIM-SM, MBGP, BGMP

Intradomain Multicast Routing Protocols

Distance Vector Multicast Routing Protocol (DVMRP)

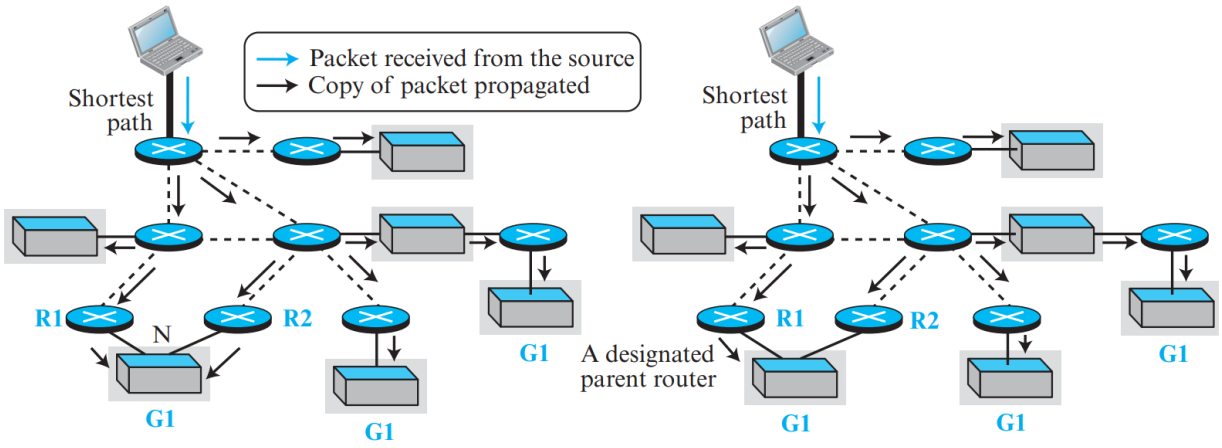
- The multicast extension of **RIP**. It uses the **source-based tree** approach.
- A router implicitly builds the source-based multicast tree in **three steps**:

1. Reverse Path Forwarding (RPF)

- The router forwards a multicast packet only from the **one interface** that lies on the shortest path **from the source** to itself.
- It finds that interface by consulting its **unicast forwarding table as if sending a packet to the source** (the shortest path is the same in both directions).
- A packet arriving on any other interface is **dropped**. This prevents **looping** and duplicate copies.

2. Reverse Path Broadcasting (RPB)

- RPF may still deliver **several copies to one network** if the network is connected to several routers. RPB allows **only one router, the designated parent router**, to pass the packet to that network.
- The parent is usually the router with the **shortest path to the source** (ties are broken by the smaller IP address).
- RPB makes a **broadcast (spanning) shortest-path tree** with the source as root and all networks as leaves.



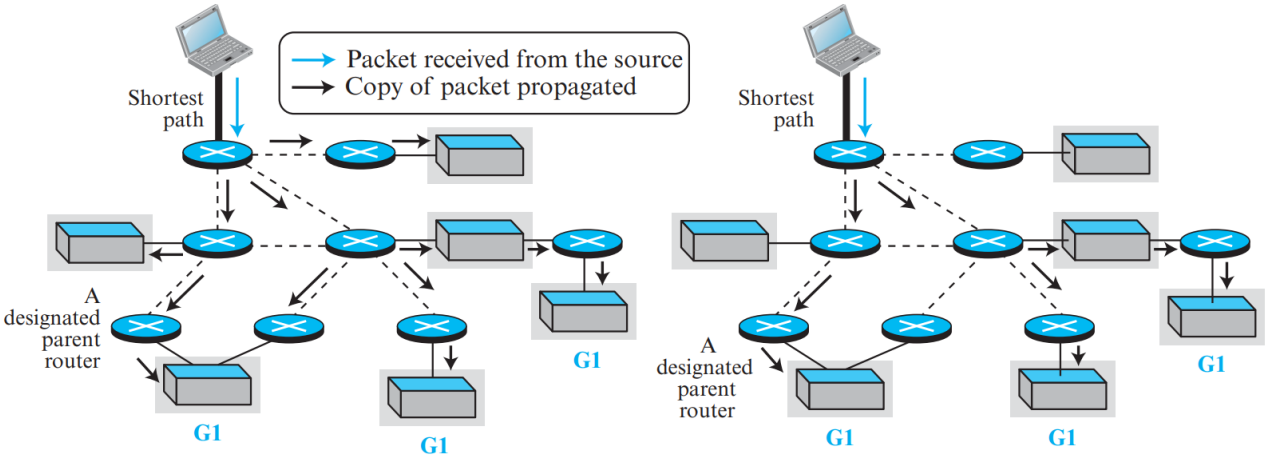
a. Using RPF, N receives two copies.

b. Using RPB, N receives only one copy.

About Figure: (a) With RPF, network N receives two copies from routers R1 and R2. (b) With RPB, only the designated parent router R1 passes a copy to N, so N receives one copy.

3. Reverse Path Multicasting (RPM)

- RPB **broadcasts**, which is inefficient. In RPM each router **prunes** (makes inactive) the interfaces that do not lead to networks with active members of that source-group combination.
- Pruning goes **bottom-up** from the leaves. Routers on a network learn membership by **IGMP**. The parent router sends this information **up the reverse shortest-path tree** (like DV messages). Membership is refreshed periodically, so joining and leaving continues.



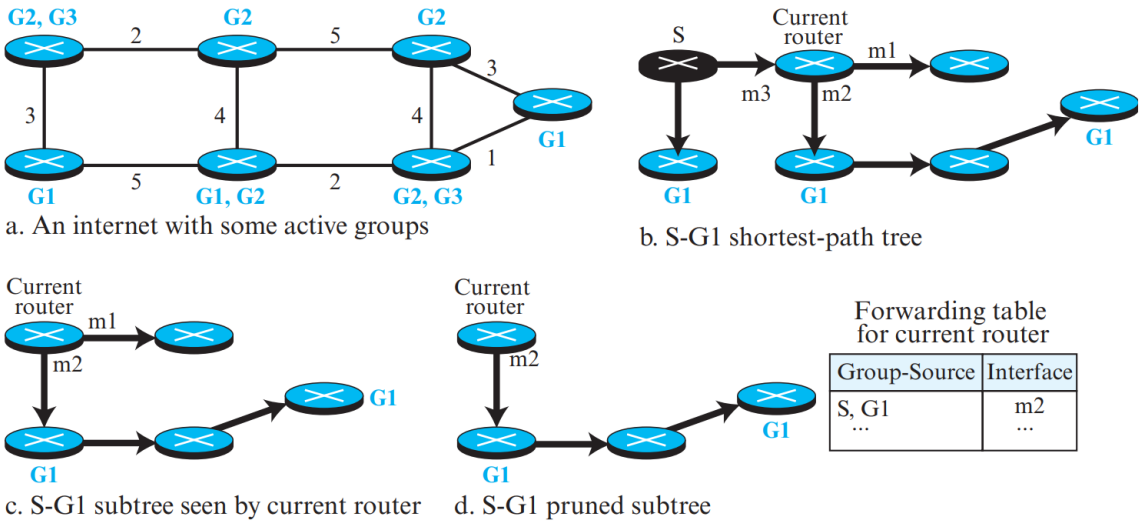
a. Using RPB, all networks receive a copy.

b. Using RPM, only members receive a copy.

About Figure: (a) RPB sends a copy to all networks. (b) RPM prunes the branches without members, so only networks with group members (or on the path to them) receive a copy.

Multicast Open Shortest Path First (MOSPF)

- The multicast extension of **OSPF**, also **source-based**. Each router has the **LSDB**, plus a database showing which interface has an active member of which group (membership is flooded using a new type of link-state update packet).
- For a packet from source S to group G, a router:
    1. Uses **Dijkstra's algorithm** to build a shortest-path tree with **S as the root** (not itself) and all destinations as leaves. It can do this because it has the whole topology. The tree depends on the source.
    2. **Finds itself** in this tree and takes the **subtree rooted at itself**.
    3. The subtree is a broadcast subtree. The router **prunes** it into a **multicast tree** using IGMP membership information.
    4. Forwards the packet **only out of the interfaces on the multicast tree branches**.



a. An internet with some active groups

b. S-G1 shortest-path tree

c. S-G1 subtree seen by current router

d. S-G1 pruned subtree

About Figure: (a) An internet with active groups. (b) The source-group shortest-path tree. (c) The subtree seen by the current router. (d) The pruned subtree used for forwarding, with the forwarding table entry (source, group) and the interfaces.

Protocol Independent Multicast (PIM)

PIM needs a unicast routing protocol's **forwarding table** to find the next router towards a destination, but it does **not matter** whether that table is built by a distance-vector or link-state protocol. It works in two modes.

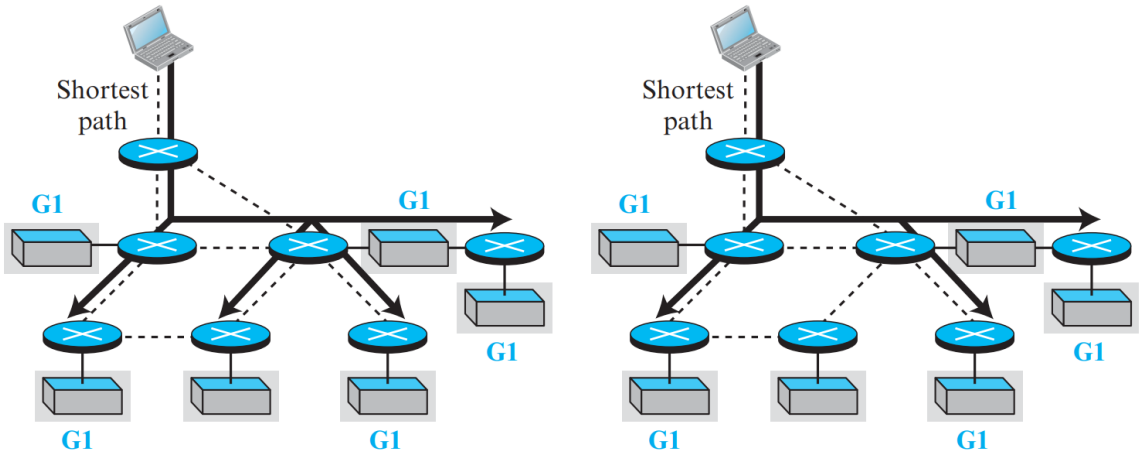
- **Dense mode (PIM-DM):** the number of routers with active members is **large** relative to all routers.
- **Sparse mode (PIM-SM):** only **a few** routers have active members.

PIM-DM

- **Source-based** tree, similar to DVMRP but simpler. It uses only **RPF and RPM**.
- Forwarding is **not suspended** until pruning finishes (it assumes most routers want the group).

1. A router receiving a packet from source S for group G uses **RPF**. If the packet did **not** arrive from the next router on the reverse path to S, it **drops** the packet and sends a **prune message** in that direction.
2. If it arrived correctly, the router forwards the packet out of **all interfaces except** the incoming one and any interface from which a **prune** for (S, G) was received.

- The **first packet is broadcast** to all networks. After prune messages return from routers with no members, the **second packet is multicast**.

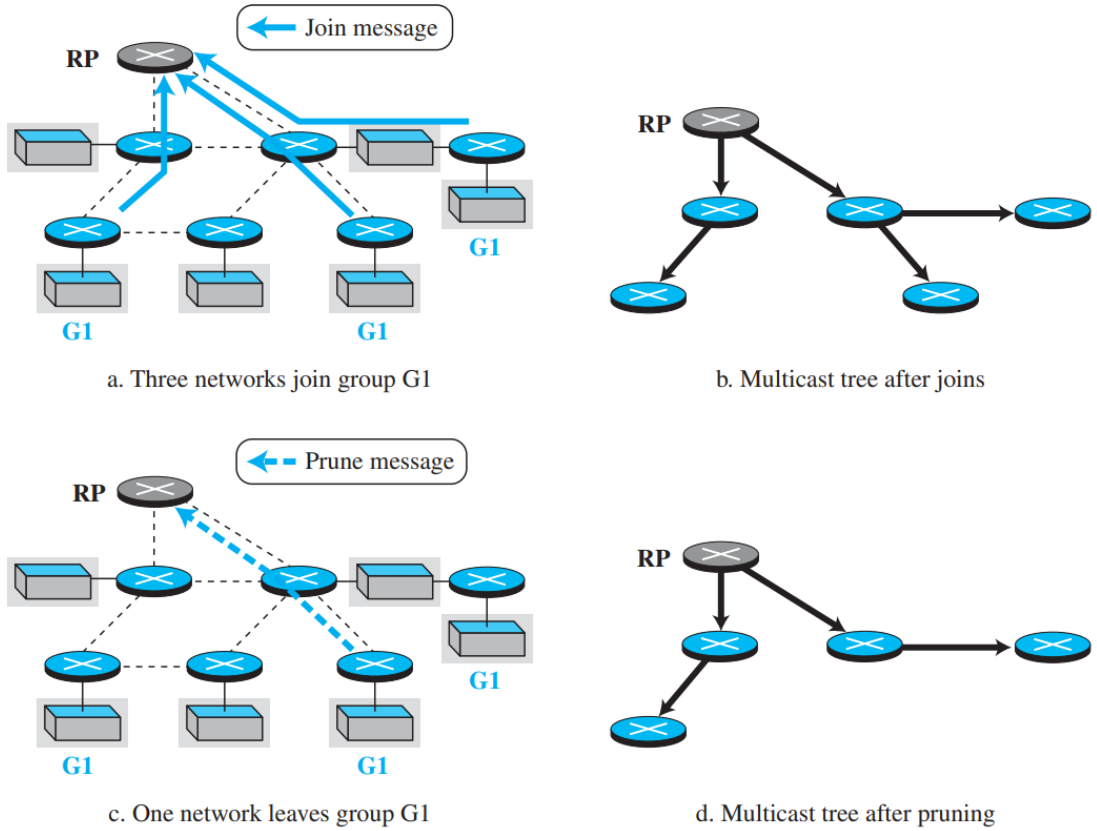


a. First packet is broadcast. b. Second packet is multicast.

About Figure: (a) The first packet is broadcast to all networks. (b) After prune messages, the second packet reaches only networks with group members.

PIM-SM

- **Group-shared tree** approach; the core router is the **rendezvous point (RP)**.
- Communication has **two steps**: a router with a multicast packet **encapsulates it in a unicast packet (tunnelling) and sends it to the RP**, and the RP **decapsulates** and sends the multicast packet to the destinations.
- With m groups there are m RPs (one router may serve many). Each router stores the group and the RP address for each group.
- A **spanning multicast tree rooted at the RP** has leaves at **designated routers** connected to networks with active members (only designated routers send PIM-SM messages, so a network does not get duplicate copies).
- The tree is built with **join** and **prune** messages.
    - **Join:** when a designated router learns by IGMP that its network has a new member, it **sends a join message as a unicast packet to the RP**. Each router on the way marks the interface from which the join came as an **output interface** for the group (and increases a **join count**), and marks the interface towards the RP as the only **input interface** for the group. The first join creates a path from the RP to the network.
    - **Prune:** when a designated router finds no active member (IGMP), it sends a **prune message** to the RP. Each router **decrements the join count** for the interface. When the count reaches **zero**, the interface is no longer part of the tree.



a. Three networks join group G1

b. Multicast tree after joins

c. One network leaves group G1

d. Multicast tree after pruning

About Figure: (a) Three networks send join messages to the RP for group G1. (b) The multicast tree after the joins. (c) One network leaves and sends a prune message. (d) The tree after pruning.

Parameter	DVMRP	MOSPF	PIM-DM	PIM-SM
Based on	RIP (distance-vector)	OSPF (link-state)	Any unicast protocol	Any unicast protocol
Approach	Source-based tree	Source-based tree	Source-based tree	**Group-shared** tree (RP)
Techniques	RPF, RPB, RPM	Dijkstra from source, pruning	RPF, RPM; prune messages	Join and prune messages, tunnelling to RP
Suited to	Dense groups	Dense groups	**Dense** groups	**Sparse** groups
First packet	Waits for pruning	—	Broadcast first	Unicast to RP

Interdomain Multicast Routing Protocols

DVMRP, MOSPF and PIM work **inside one AS**. For groups spread over several ASs, interdomain protocols are needed.

Protocol	Description
MBGP (Multicast BGP)	An extension of BGP that gives **two paths between ASs**, one for unicast and one for multicast. Border routers exchange multicast information. It is a **shared-group** protocol with one **RP** per AS. Problem: hard to tell an RP about **sources in other ASs**
MSDP (Multicast Source Discovery Protocol)	A proposed protocol that assigns a **source representative router** in each AS to tell all RPs about sources in that AS
BGMP (Border Gateway Multicast Protocol)	A possible replacement for MBGP. Builds **shared-group trees with a single root in one AS**: one shared tree per group, with leaves in different ASs. Problems: how to **choose the root AS** and how to tell sources where the root is

Next Generation IP (IPv6)

Why IPv6?

The address depletion of IPv4 and its other shortcomings led to IPv6 (IPng) in the early 1990s. (IPv5 was an OSI-based proposal that never materialised.)

Main changes in IPv6

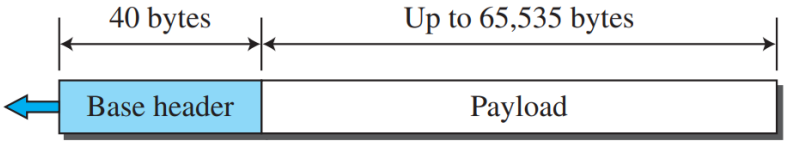
- **Larger address space:** 128-bit addresses (2^96 times IPv4).
- **Better header format:** options are separated from the base header and placed (when needed) between the base header and the data. This speeds up routing, since most options need not be checked by routers.
- **New options** for additional functions.
- **Allowance for extension** for new technologies.
- **Support for resource allocation:** the type-of-service field is removed and **traffic class** and **flow label** are added, so the source can request special handling (for example real-time audio and video).
- **More security:** encryption and authentication options give confidentiality and integrity.

Why adoption is slow:

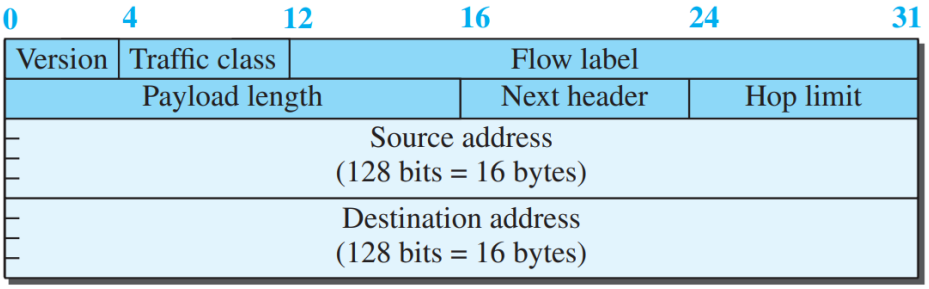
the original reason (address depletion) was eased by three short-term remedies: **classless addressing, DHCP and NAT**. Mobile IP, IP telephony and IP-capable mobile phones may need IPv6 in the future.

IPv6 Packet Format

A packet has a **40-byte base header** followed by the **payload** (up to 65,535 bytes).



a. IPv6 packet



b. Base header

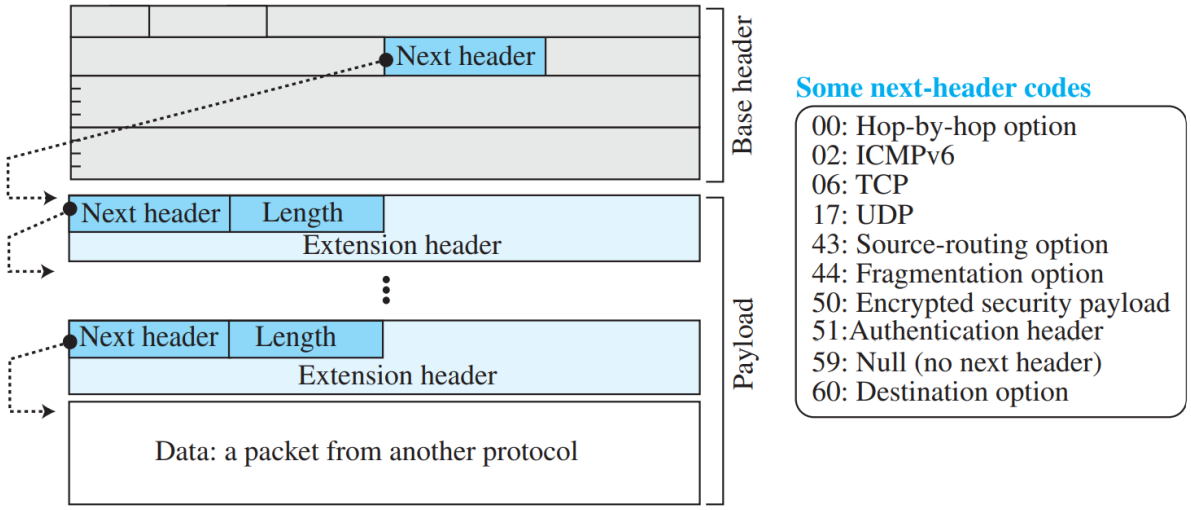
About Figure: (a) The IPv6 packet is a 40-byte base header plus the payload. (b) The base header has version, traffic class, flow label, payload length, next header, hop limit, and the 128-bit source and destination addresses.

Field	Size	Function
Version	4 bits	Value 6
Traffic class	8 bits	Distinguishes payloads with different delivery needs; replaces the IPv4 type-of-service field
Flow label	20 bits	Special handling for a flow of data
Payload length	16 bits	Length of the datagram **excluding the base header** (the base header length is fixed, so only the payload length is needed)

Field	Size	Function
Next header	8 bits	Type of the first extension header, or of the data after the base header (like the protocol field in IPv4)
Hop limit	8 bits	Same purpose as TTL in IPv4
Source address	128 bits	Original source
Destination address	128 bits	Destination

Payload

It holds **zero or more extension headers** followed by data from another protocol (UDP, TCP and so on). Each extension header has the mandatory fields **next header** and **length**, then option information. The last next-header value gives the protocol carried.



About Figure: The payload is a chain of extension headers (each with next header and length) followed by the data. Some next-header codes: 00 hop-by-hop option, 02 ICMPv6, 06 TCP, 17 UDP, 43 source routing, 44 fragmentation, 50 encrypted security payload, 51 authentication header, 59 null, 60 destination option.

Concept of flow and priority

- A **flow** is a sequence of packets sharing characteristics (same path, same resources, same security and so on).
- A router that supports flow labels keeps a **flow label table**, with an entry for each active label defining the service required. The router looks up the label in the packet to give it that service.
- **Simple use:** speed up forwarding. The router finds the next hop in the flow label table without the full routing algorithm.
- **Advanced use:** support **real-time audio and video**, with resources reserved beforehand (using protocols such as RTP and RSVP).
- The flow label is what lets IPv6 behave like a **connection-oriented** protocol. The table entries come from other means (hop-by-hop options or other protocols).

Fragmentation and reassembly

- In IPv6 only the **source** may fragment. **Routers are not allowed to fragment**; reassembly is at the destination.
- This speeds up the routers, since fragmentation needs a lot of processing.
- A router that gets a packet **larger than the next link's MTU drops it** and sends a **packet-too-big ICMPv6** message to the source.

IPv6 Addressing

An IPv6 address is **128 bits (16 bytes)**, four times the length of an IPv4 address.

Notation

- **Binary:** used inside computers.
- **Colon hexadecimal:** eight sections of **four hex digits** separated by colons, for example: FEF6:BA98:7654:3210:ADEF:BBFF:2922:FF00.

Abbreviation

- **Leading zeros** in a section may be omitted (0074 → 74, 000F → F, 0000 → 0). Zeros inside a section (3210) cannot be dropped.
- **Zero compression:** consecutive all-zero sections are replaced by a **double colon (::)**, **only once per address**. Example: FDEC:0:0:0:0:BBFF:0:FFFF → FDEC::BBFF:0:FFFF.
- **Mixed notation:** colon hex plus dotted decimal for an embedded IPv4 address (rightmost 32 bits), for example ::130.24.24.18 (the leftmost 96 bits are zeros).
- **CIDR (slash) notation** is allowed, for example FDEC::BBFF:0:FFFF/60 (a 60-bit prefix).

Address space

$$\text{IPv6 address space, } N = 2^{128}$$

$$\text{Ratio of IPv6 to IPv4 address space: } \frac{2^{128}}{2^{32}} = 2^{96}$$

The space is about 3.4×10^{38} addresses (340,282,366,920,938,463,374,607,431,768,211,456). If only 1/64 of the addresses go to the 2^{34} people expected on Earth, each person still gets 2^{88} **addresses**. Depletion is impossible.

Three address types

Type	Meaning
Unicast	Defines a **single interface** (computer or router). The packet goes to that interface
Anycast	A **group** of computers share one address. The packet is delivered to **only one member**, the **most reachable**. There is no special block; addresses come from the unicast block. Example: several servers able to answer an inquiry
Multicast	A group of computers share an address. **Each member** receives a copy. It has a designated block

IPv6 does **not define broadcasting**; it is treated as a special case of multicasting.

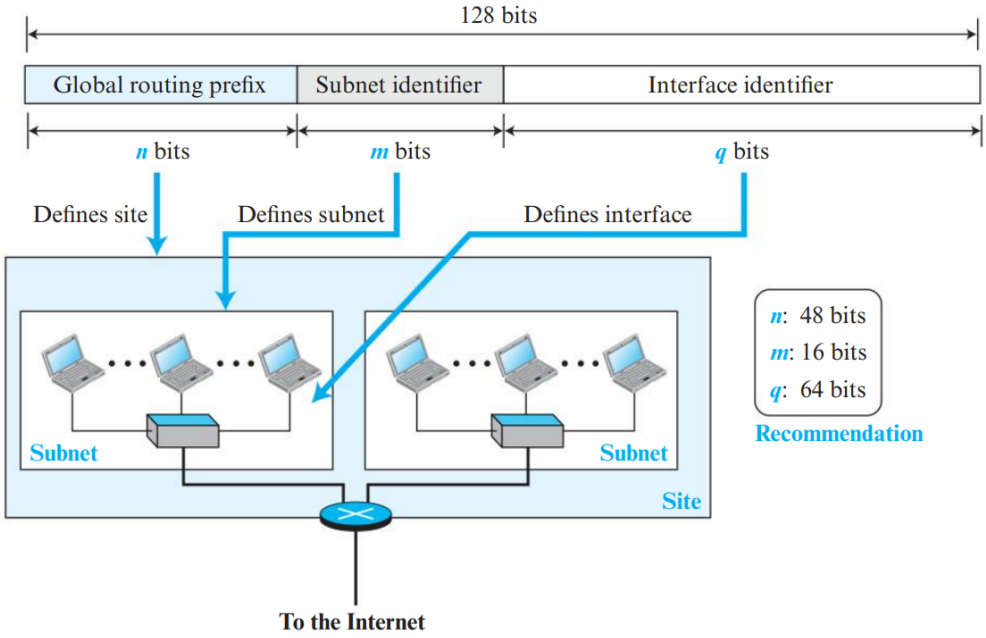
Address space allocation

Block prefix	CIDR	Assignment	Fraction
0000 0000	0000::/8	Special addresses	1/256
001	2000::/3	**Global unicast**	1/8
1111 110	FC00::/7	Unique local unicast	1/128
1111 1110 10	FE80::/10	Link local	1/1024
1111 1111	FF00::/8	Multicast	1/256

Global unicast addresses (2000::/3)

The three leftmost bits are 001. An address has three parts:

Part	Recomm. length	Meaning
Global routing prefix (n)	48 bits	Routes the packet through the Internet to the **site** (organisation or ISP). The fixed 001 leaves 45 bits, so up to 2^{45} sites
Subnet identifier (m)	16 bits	Defines a **subnet** in the organisation (up to 2^{16} = 65,536 subnets)
Interface identifier (q)	64 bits	Defines the **interface** (like hostid in IPv4)



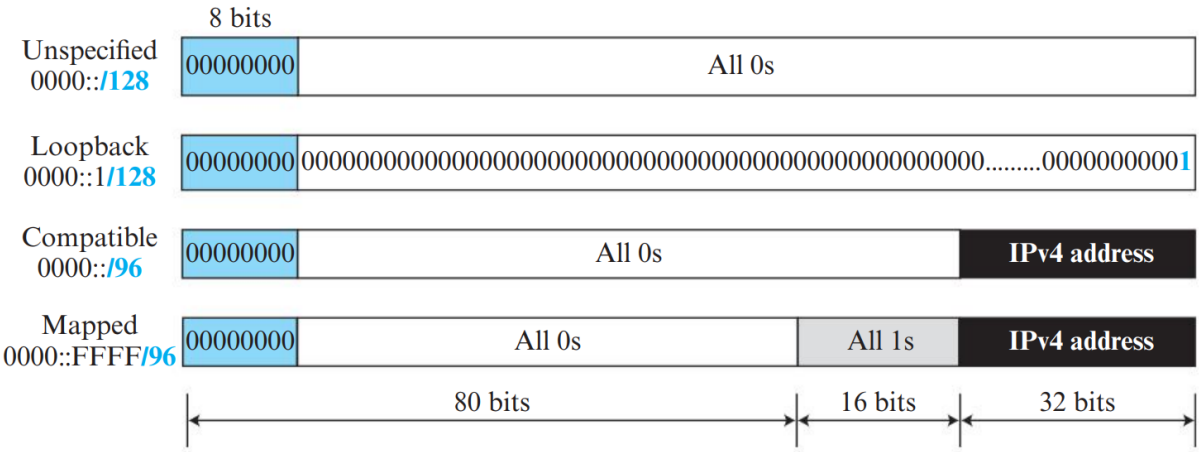
About Figure: A 128-bit address split into a global routing prefix (n bits, defines the site), a subnet identifier (m bits, defines the subnet) and an interface identifier (q bits, defines the interface).

- A link-layer address shorter than 64 bits (for example the 64-bit **EUI-64** or the 48-bit Ethernet address) can be **embedded in the interface identifier**, removing the need for address mapping (unlike IPv4).

Special addresses (block 0000::/8)

Address	Form	Use
Unspecified	0000::/128 (all zeros)	Used at **bootstrap** when a host does not know its own address (see DHCP)
Loopback	0000::1/128	A **single** address for testing (IPv4 has a block)
Compatible	0000::/96 + IPv4 address	96 zero bits then the IPv4 address; used when an **IPv6 computer sends to another IPv6 computer** during transition
Mapped	0000::FFFF/96 + IPv4 address	Used when a computer **already on IPv6 sends to a computer still on IPv4**

Compatible and mapped addresses are designed so that the **checksum is unchanged** whether the embedded or the whole address is used (extra 0s or 1s in multiples of 16 bits do not change the sum). This matters for UDP and TCP, whose pseudoheader uses the address.

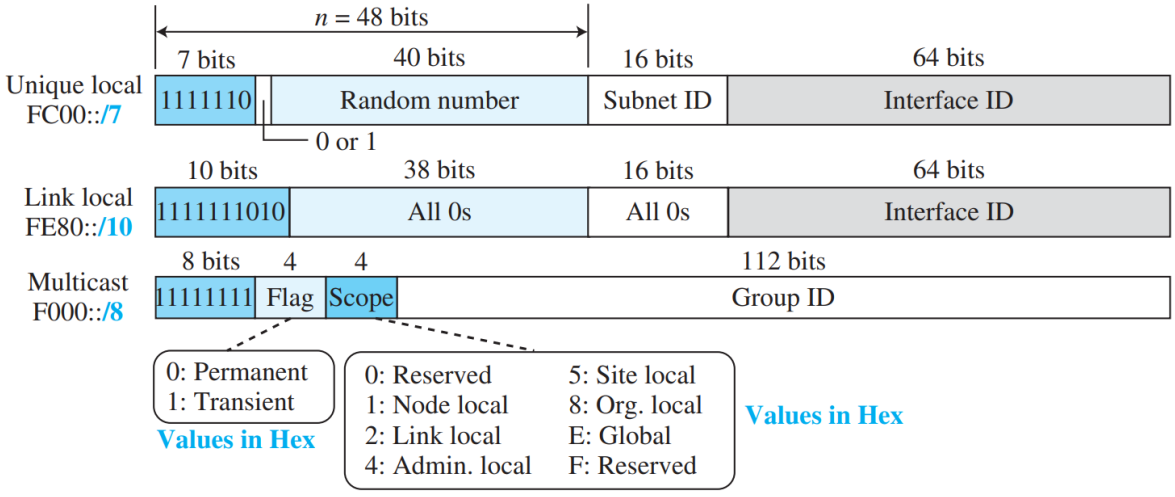


About Figure: The unspecified address (all 0s), the loopback address (0s ending in 1), the compatible address (80 zero bits, 16 zero bits, IPv4 address) and the mapped address (80 zero bits, 16 one bits, IPv4 address).

Other assigned blocks

- **Unique local unicast (FC00::/7): private** addresses for a site, not expected to be routed. Format: **1111110**, one bit (0 or 1), a **40-bit random number**, a 16-bit subnet ID and a 64-bit interface ID. The random number makes duplication extremely unlikely.
- **Link local (FE80::/10):** private addresses **within one network (link)**. Format: **1111111010**, then **54 zero bits**, then the 64-bit interface ID.
- **Multicast (FF00::/8):** prefix **11111111**, a 4-bit **flag** (0 = permanent group defined by the Internet authorities, 1 = transient group used temporarily), a 4-bit **scope** and a **112-bit group ID**.

Scope value	Meaning
1	Node local
2	Link local
4	Admin local
5	Site local
8	Organisation local
E	Global
0, F	Reserved

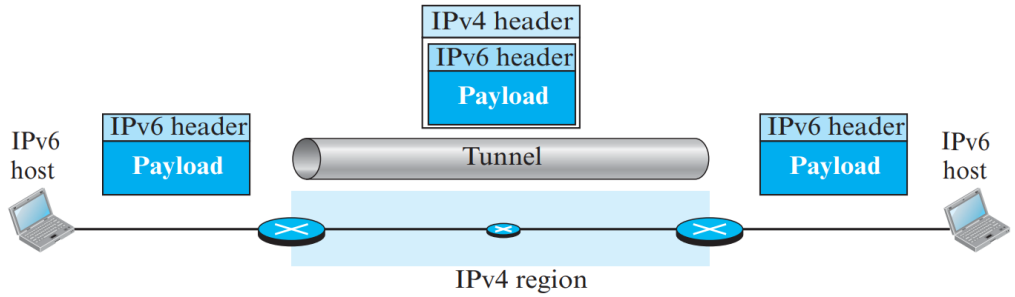


About Figure: The unique local block (7-bit prefix, 0 or 1, 40-bit random number, 16-bit subnet ID, 64-bit interface ID), the link local block (10-bit prefix, 54 zero bits, 64-bit interface ID) and the multicast block (8-bit prefix, flag, scope, 112-bit group ID).

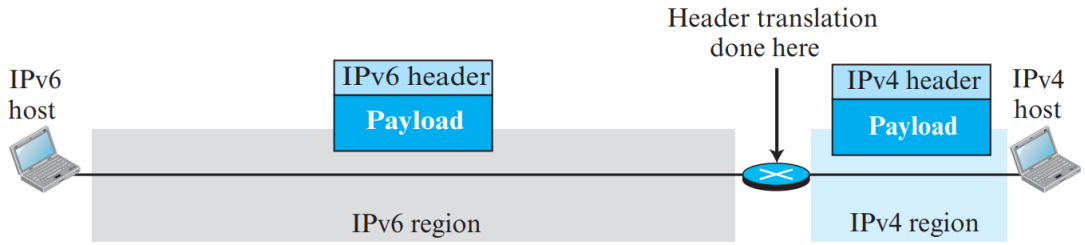
Transition from IPv4 to IPv6

A sudden switch on one day is impossible because of the huge number of systems. The transition must be smooth. IETF recommends **three strategies**.

Strategy	Working
Dual stack	A host runs **IPv4 and IPv6 together** until the whole Internet uses IPv6. To choose the version, the source **queries DNS**: an IPv4 address means send an IPv4 packet, an IPv6 address means send an IPv6 packet
Tunneling	Two **IPv6 hosts** communicate through a region that uses **IPv4**. The IPv6 packet is **encapsulated in an IPv4 packet** when it enters the region and leaves its capsule on exit. The IPv4 **protocol field is set to 41** to show it carries an IPv6 packet
Header translation	Used when most of the Internet is IPv6 but the **receiver uses only IPv4**. Tunneling does not work, since the packet must be in IPv4 format. The **IPv6 header is converted to an IPv4 header**



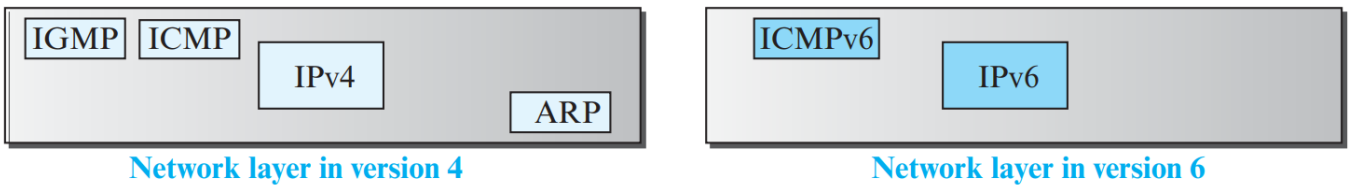
About Figure: An IPv6 host sends a packet across an IPv4 region. At the tunnel entrance the IPv6 packet (header and payload) is wrapped in an IPv4 header, and at the exit the IPv4 header is removed.



About Figure: An IPv6 host sends a packet in the IPv6 region. Header translation at the boundary converts the IPv6 header into an IPv4 header before it enters the IPv4 region towards the IPv4 host.

ICMPv6

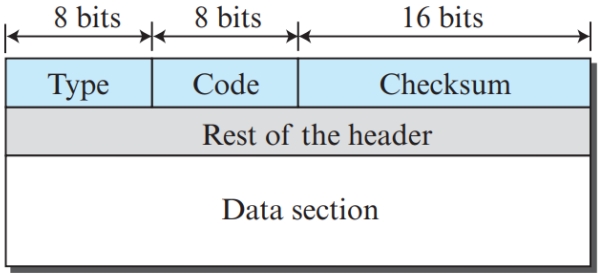
ICMPv6 has the same purposes as ICMPv4 but is **more complicated**. It **combines ICMP, ARP and IGMP** of version 4 into one protocol.



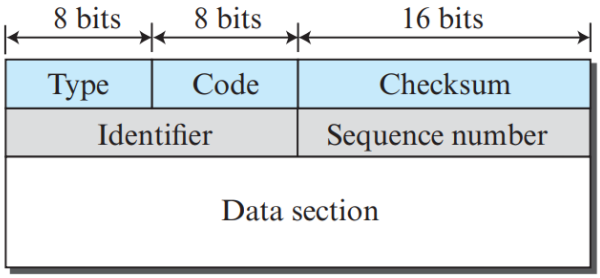
About Figure: In version 4 the network layer has IPv4 with IGMP, ICMP and ARP. In version 6 it has IPv6 with a single protocol, ICMPv6.

Messages fall into **four groups**:

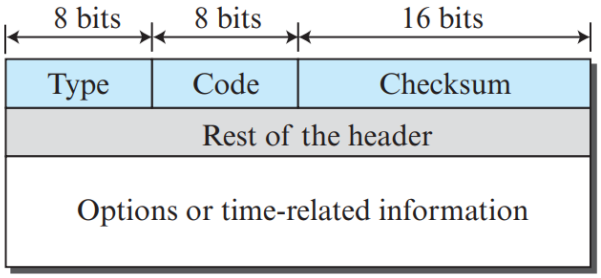
Group	Messages
Error reporting	Destination unreachable (type 1), **packet too big** (2), time exceeded (3), parameter problem (4). Source quench was **dropped** (rarely used); redirection moved to neighbour discovery. **Packet-too-big is new**, because routers cannot fragment
Informational	**Echo request and reply** (128, 129). Timestamp messages were dropped (rarely used)
Neighbor discovery (new group)	**Router solicitation/advertisement** (133, 134): find a default router (done by DHCP in v4). **Neighbor solicitation/advertisement** (135, 136): find the **link-layer address** from an IPv6 address (done by ARP in v4). **Redirection** (137). **Inverse neighbor solicitation/advertisement** (141, 142): find the IPv6 address from a link-layer address
Group membership	**Membership query** (130) and **membership report** (131), used in multicasting (the IGMP job)



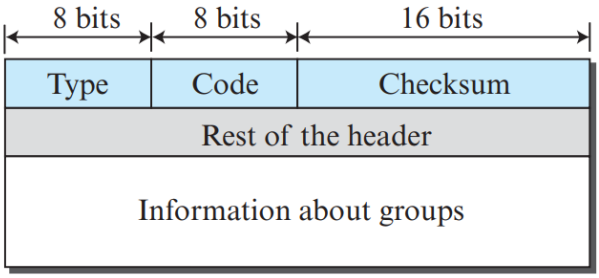
Error-reporting messages



Informational messages



Neighbor-discovery messages



Group-membership messages

Type and code values

Error-reporting messages
01: Destination unreachable (codes 0 to 6)
02: Packet too big (code 0)
03: Time exceeded (codes 0 and 1)
04: Parameter problem (codes 0 to 2)

Informational messages
128 and 129: Echo request and reply (code 0)

Group-membership messages
130: Membership query (code 0)
131: Membership report

Neighbor-discovery messages
133 and 134: Router solicitation and advertisement (code 0)
135 and 136: Neighbor solicitation and advertisement (code 0)
137: Redirection (codes 0 to 3)
141 and 142: Inverse neighbor solicitation and advertisement (code 0)

About Figure: The message formats and type values for error-reporting, informational, neighbour-discovery and group-membership messages, each with type, code and checksum fields.

Parameter	IPv4	IPv6
Address size	32 bits	**128 bits**
Header	20 to 60 bytes (variable, options included)	**Fixed 40 bytes**; options are extension headers
Header checksum	Present (recalculated at each router)	Not in the base header
Fragmentation	Source or any router	**Source only**; routers drop and send packet-too-big
Type of service	Service type field	**Traffic class** and **flow label**
Length field	Header length and total length	**Payload length** only
TTL / protocol	Time-to-live, protocol	**Hop limit**, **next header**
Broadcast	Supported (limited)	**None** (special case of multicast)
Auxiliary protocols	ICMP, IGMP, ARP separate	All combined in **ICMPv6**
Security	Added by IPSec	Built-in authentication and encryption options

Quality of Service (QoS)

The Internet was designed for **best-effort service** with no guarantee. This suits traffic that is not delay-sensitive, such as file transfer and e-mail. Such traffic is **elastic** (can stretch under delay) and **available bit rate** (applications speed up or slow down to the available rate).

Real-time multimedia traffic is delay-sensitive and needs guaranteed, predictable performance.

Quality of service (QoS) is a set of techniques and mechanisms that **guarantee the performance of the network** to deliver predictable service to an application.

Data-Flow Characteristics

Four characteristics are attributed to a flow:

Characteristic	Meaning
Reliability	Packets reach the destination safe and sound. Loss means retransmission. Most important for e-mail, file transfer and Internet access; less for telephony
Delay	Source-to-destination delay. Telephony, audio and video conferencing, and remote login need **minimum delay**; file transfer and e-mail do not
Jitter	**Variation in delay** for packets of the same flow. Example: packets leaving at 0, 1, 2, 3 and arriving at 20, 21, 22, 23 all have 20 units of delay (acceptable). Arrivals at 21, 23, 24, 28 give different delays (not acceptable for audio and video)
Bandwidth	Different applications need different rates (video conferencing needs millions of bits per second; e-mail much less)

Sensitivity of applications

Application	Reliability	Delay	Jitter	Bandwidth
FTP	High	Low	Low	Medium
HTTP	High	Medium	Low	Medium
Audio-on-demand	Low	Low	High	Medium
Video-on-demand	Low	Low	High	High
Voice over IP	Low	High	High	Low
Video over IP	Low	High	High	High

Meaning: high reliability sensitivity needs error checking and discarding of corrupted packets. High delay sensitivity needs **priority**. High jitter sensitivity needs packets of the flow to pass the network with the **same delay**. High bandwidth needs **enough bandwidth allocated**.

Flow Classes

The Internet has not formally defined flow classes. For example FTP needs high reliability and medium bandwidth, with delay and jitter unimportant. The **ATM** protocol defines five service classes:

Class	Use
CBR (constant bit rate)	Emulates circuit switching; sensitive to cell-delay variation (telephone, video conferencing, television)
VBR-NRT (variable bit rate, non real time)	Rate varies with the available user information (multimedia e-mail)
VBR-RT (variable bit rate, real time)	Like VBR-NRT but for applications sensitive to cell-delay variation (interactive compressed video)
ABR (available bit rate)	Rate-based flow control; for data traffic such as file transfer and e-mail
UBR (unspecified bit rate)	All other traffic; widely used for TCP/IP

Techniques to Improve QoS

The IP header has a ToS field that informally defines the service required. Provisions are made through scheduling, traffic shaping, admission control and resource reservation.

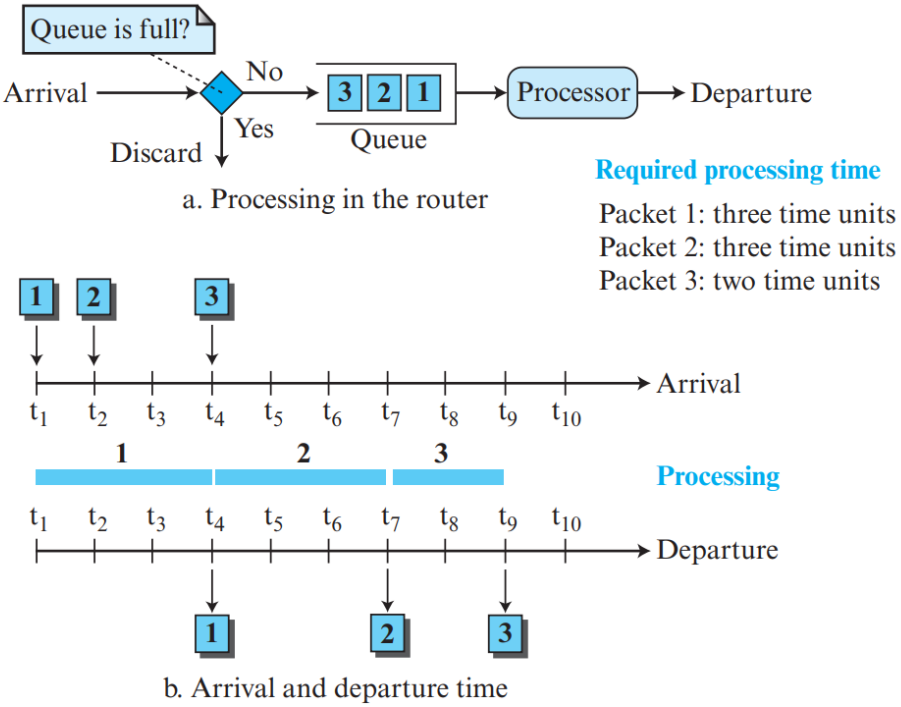
Scheduling

Packets are treated by their required service at **routers**.

1. FIFO queuing

- Packets wait in a buffer until the router is ready. If the arrival rate is higher than the processing rate, the queue fills and **new packets are discarded**.
- **Default** scheduling in the Internet. Packets leave in the **order they arrive**.
- **No distinction between classes**: FTP, VoIP and e-mail are equally subject to loss, delay and jitter. Larger packets need longer processing, so packets of one application get different delays (jitter).

About Figure (NEXT PAGE): (a) Packets arrive at a queue (discarded if full) and are served by the processor. (b) A time line of arrival, processing (packets 1 and 2 need 3 time units, packet 3 needs 2) and departure.

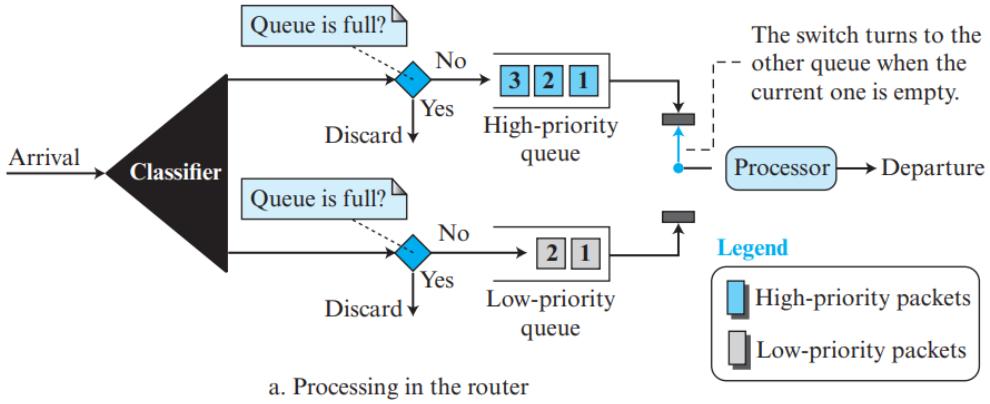


a. Processing in the router

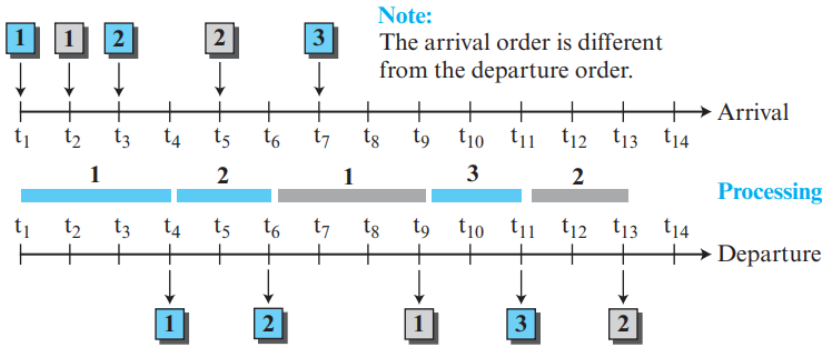
b. Arrival and departure time

2. Priority queuing

- Packets are assigned to **priority classes**, each with its **own queue**. The **highest-priority queue is served first**. The system does not move to a lower queue until the higher one is **empty**.
- Priority is taken from the ToS field of IPv4, the priority field of IPv6, a destination address, or an application (port number).
- Better QoS than FIFO for multimedia. **Drawback: starvation**. A continuous high-priority flow means lower-priority packets are never served (and may be dropped).



a. Processing in the router

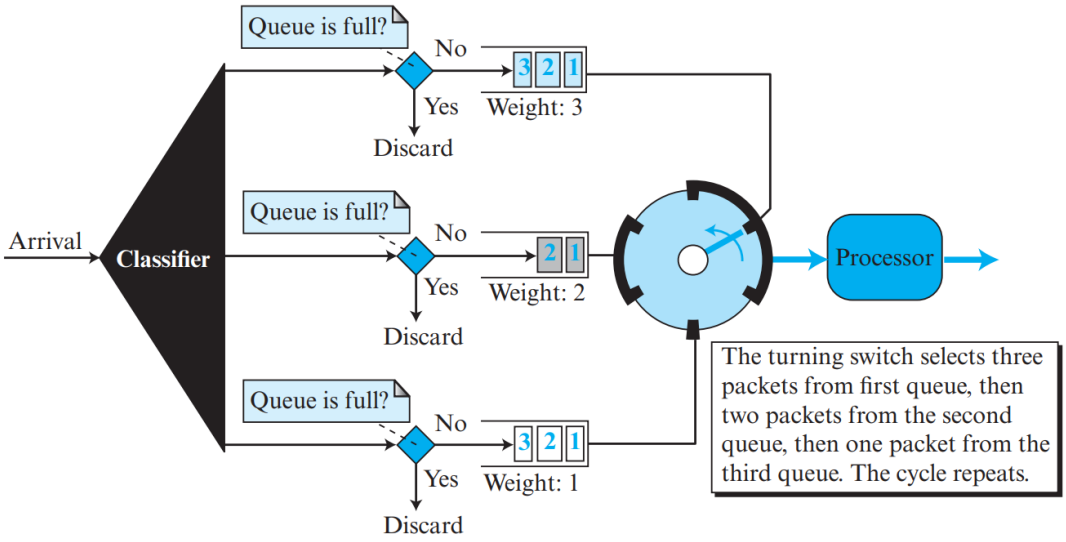


b. Arrival and departure time

About Figure: A classifier sends packets to a high-priority queue and a low-priority queue. The processor serves the high-priority queue until it is empty, so packets depart in a different order from arrival.

3. Weighted fair queuing

- Packets are classified into queues that carry **weights** (higher priority means higher weight). The processor serves the queues **round robin**, taking from each queue a number of packets equal to its weight. Weights 3, 2, 1 mean 3 packets from the first queue, 2 from the second and 1 from the third.
- Each class gets a **fraction of the time** depending on its priority. For throughput R with weights 3, 2, 1, the classes get R/2, R/3 and R/6 (assuming equal packet sizes; unequal sizes cause imbalance).



About Figure: A classifier feeds three queues with weights 3, 2 and 1. A turning switch takes three packets from the first queue, two from the second and one from the third, then repeats.

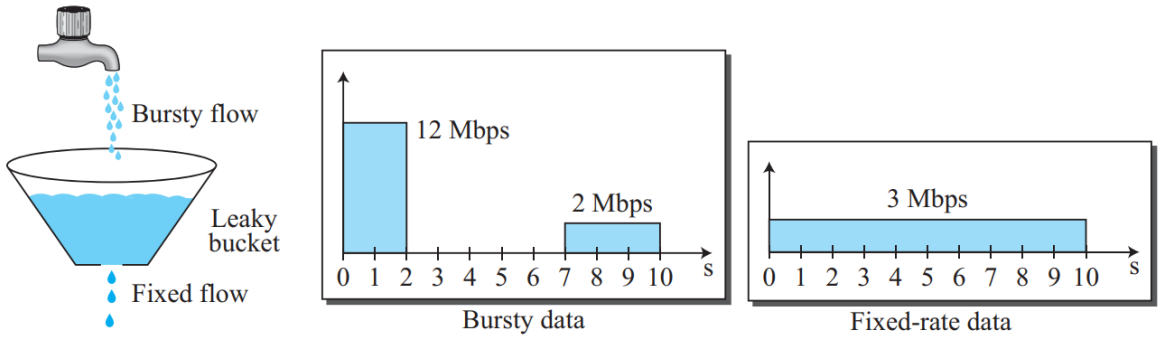
Parameter	FIFO	Priority queuing	Weighted fair queuing
Queues	One	One per priority class	One per class, with weights
Service rule	Arrival order	Highest non-empty queue first	Round robin by weight
Distinguishes classes	No	Yes	Yes
Problem	Same treatment for all	**Starvation** of low priority	Packets of different sizes give imbalance

Traffic Shaping or Policing

Controlling the **amount and rate** of traffic. **Shaping** is used when traffic **leaves** a network; **policing** when it **enters**. Two techniques: **leaky bucket** and **token bucket**.

Leaky bucket

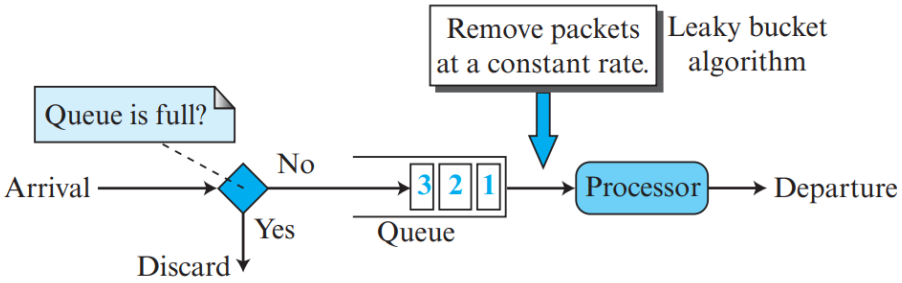
- Like a bucket with a small hole: water leaks at a **constant rate** whatever the input rate, and overflows if the bucket is full. In networking, **bursty traffic is stored and sent at a fixed average rate**.
- *Example:* the network commits 3 Mbps to a host. The host sends 12 Mbps for 2 s (24 Mb), is silent for 5 s, then sends 2 Mbps for 3 s (6 Mb). In all, 30 Mb in 10 s. The bucket sends at **3 Mbps for the 10 s**.



About Figure: A bursty flow (12 Mbps for 2 s, silence, then 2 Mbps for 3 s) enters the leaky bucket, and a fixed flow of 3 Mbps over 10 s comes out.

Implementation: a FIFO queue holds packets. For fixed-size packets, a fixed number are removed per clock tick. For variable-length packets the **output is based on bytes or bits**:

1. Initialise a counter to n at the clock tick.
2. If n is greater than the packet size, send the packet and decrement n by the packet size. Repeat until n is smaller than the next packet.
3. Reset the counter to n and go to step 1.



About Figure: Arriving packets enter a queue (discarded if the queue is full), and the processor removes them at a constant rate.

Token bucket

- The leaky bucket is restrictive: it gives **no credit to an idle host**. The token bucket lets an idle host **accumulate tokens** for later bursts.
- Capacity is c tokens; tokens enter at r per second; **one token is removed for each cell sent**; tokens are discarded if the bucket is full.

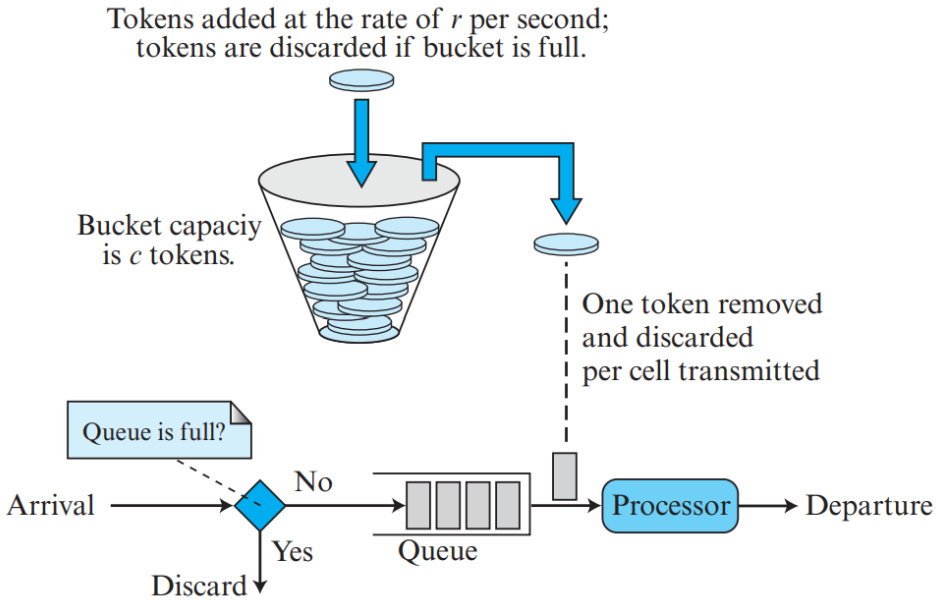
Maximum number of cells in an interval t, Maximum number of cells $= r\,t + c$

Maximum average rate, Maximum average rate $= \frac{r\,t + c}{t}$ cells per second

Where r = token rate (tokens per second), c = bucket capacity (tokens) and t = time interval (seconds).

Example*:* c = 10,000 tokens, r = 1000 tokens per second. After 10 s of idleness the bucket is full (10,000 tokens). The maximum average rate is: $\frac{1000\,t + 10000}{t}$

(For t = 10 s this is 2000 cells per second; as t grows it approaches 1000, the token rate.) It can be implemented with a **counter**: incremented when a token is added, decremented when a unit is sent, and the host cannot send when it is zero.



About Figure: Tokens are added at r per second into a bucket of capacity c. Each cell that is sent removes one token, and arrivals wait in a queue (discarded if full).

Combining them: the **leaky bucket is applied after the token bucket**, crediting an idle host and also regulating the traffic. The leaky bucket rate must be higher than the token rate.

Parameter	Leaky bucket	Token bucket
Output rate	**Constant**	Can burst up to the credit saved
Idle host	**No credit**	Accumulates tokens
Controls	Average rate	Maximum average rate (rt + c)
Full bucket	Packets discarded	Tokens discarded

Resource Reservation and Admission Control

- A flow needs resources (**buffer, bandwidth, CPU time**). QoS is better if they are **reserved beforehand**.
- **Admission control:** a router or switch **accepts or rejects a flow** by checking its **flow specification** against its capacity (bandwidth, buffer, CPU speed) and previous commitments. In ATM this is **Connection Admission Control (CAC)**.

Integrated Services (IntServ)

IntServ is a **flow-based QoS model for IP**. Resources such as bandwidth are **explicitly reserved** for a flow, whatever the application. It rests on three schemes:

1. Packets are **classified** by the service they require.
2. **Scheduling** forwards packets by flow characteristics.
3. **Admission control** checks whether a router has the resources before committing.

It needs a **connection-oriented** network layer, but IP is connectionless, so another protocol, **RSVP**, runs over IP to make it connection-oriented.

Flow specification has two parts:

- **Rspec (resource specification):** the resources the flow needs to reserve (buffer, bandwidth).
- **Tspec (traffic specification):** the traffic characteristics of the flow.

Admission: the router admits or denies a flow based on its previous commitments and current resources.

Service classes

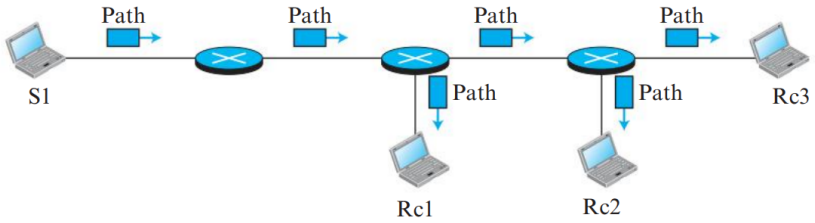
Class	Purpose
Guaranteed service	For **real-time traffic** needing a **guaranteed minimum end-to-end delay**. Packets arrive within a set time and are not discarded if the flow stays within Tspec. **Quantitative**: delay and data rate are defined by the application. Only the router delays can be guaranteed
Controlled-load service	For applications that **accept some delay** but are sensitive to an overloaded network and packet loss (file transfer, e-mail, Internet access). **Qualitative**: asks for low-loss or no-loss delivery

Resource Reservation Protocol (RSVP)

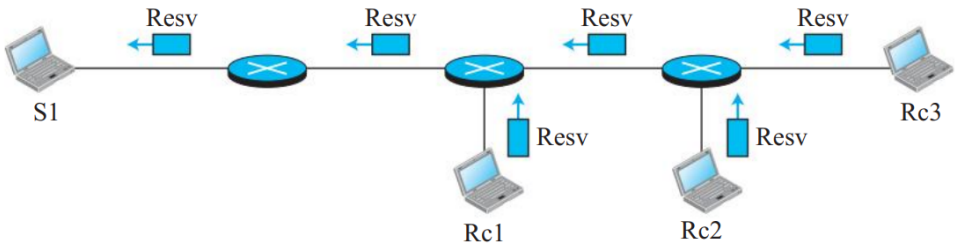
- An independent signalling protocol that helps IP make a flow and reserve resources.
- **Multicast trees:** RSVP is based on multicasting (unicast is a special case with one member), so it can reserve for multimedia.
- **Receiver-based reservation:** the **receivers, not the sender**, make the reservation (as in multicast routing, where receivers join or leave groups).

RSVP messages

- **Path message:** travels **from the sender to all receivers** along the multicast path, storing the information the receivers need (they do not know the path before reserving). A new message is made where the path diverges.
- **Resv message:** after receiving a Path message, a receiver sends a Resv message **upstream towards the sender**. It makes the reservation on routers that support RSVP. A router without RSVP uses best-effort delivery.

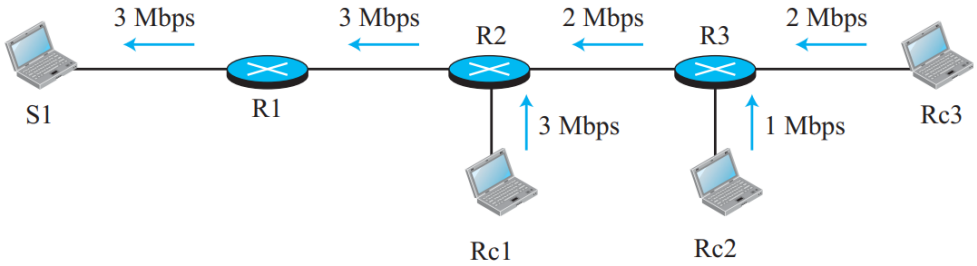


About Figure: A Path message from sender S1 is copied at branching routers and reaches receivers Rc1, Rc2 and Rc3.



About Figure: Receivers Rc1, Rc2 and Rc3 send Resv messages back along the same paths towards S1.

Reservation merging: Resources are not reserved separately for each receiver of a flow. A router **merges** requests and reserves the **larger**. For example Rc3 requests 2 Mbps and Rc2 requests 1 Mbps, so router R3 reserves **2 Mbps**, which serves both. Receivers may ask for different amounts because they can handle different quality.



About Figure: Receivers Rc1 (3 Mbps), Rc2 (1 Mbps) and Rc3 (2 Mbps) send requests through routers R1, R2 and R3, which merge them into reservations of the largest request on each link.

Reservation styles

Style	Reservation	Used when
Wildcard filter (WF)	A **single** reservation for **all senders**, based on the **largest** request	Flows from different senders do **not occur at the same time**
Fixed filter (FF)	A **distinct** reservation for **each flow** (n flows, n reservations)	Flows from different senders are **likely at the same time**
Shared explicit (SE)	A **single** reservation **shared by a set of flows**	Selected senders share it

Soft state: The reservation information at each node must be **refreshed periodically** (default **30 seconds**), unlike the **hard state** of virtual-circuit protocols such as ATM, where information stays until erased.

Problems with IntServ

1. **Scalability:** each router must keep information **for each flow**, which is a serious problem for core routers that mainly switch packets at a high rate.
2. **Service-type limitation:** only two service types, guaranteed and controlled-load. Applications may need more.

Differentiated Services (DiffServ)

DiffServ is a **class-based QoS model for IP**. Packets are **marked by applications into classes** by priority, and routers use queuing strategies. IETF introduced it to overcome IntServ's shortcomings through **two changes**:

1. The **main processing moves from the core to the edge** of the network, which solves scalability. Routers store **no flow information**; the host defines the service each time it sends a packet.
2. **Per-flow service becomes per-class service**: the router forwards by the **class** in the packet. Many classes can be defined, which solves the service-type limitation.

DS field: Each packet has a DS field, set at the **network boundary** by the host or the first (boundary) router. It replaces the **ToS** field of IPv4 or the **priority class** field of IPv6. It has two subfields:

- **DSCP (Differentiated Services Code Point): 6 bits**, defines the **per-hop behaviour (PHB)**.
- **CU (Currently Unused):** 2 bits.

A DiffServ node uses the 6 DSCP bits as an **index into a table** that defines how to handle the packet.



About Figure: An 8-bit DS field divided into the 6-bit DSCP and the 2-bit CU subfield.

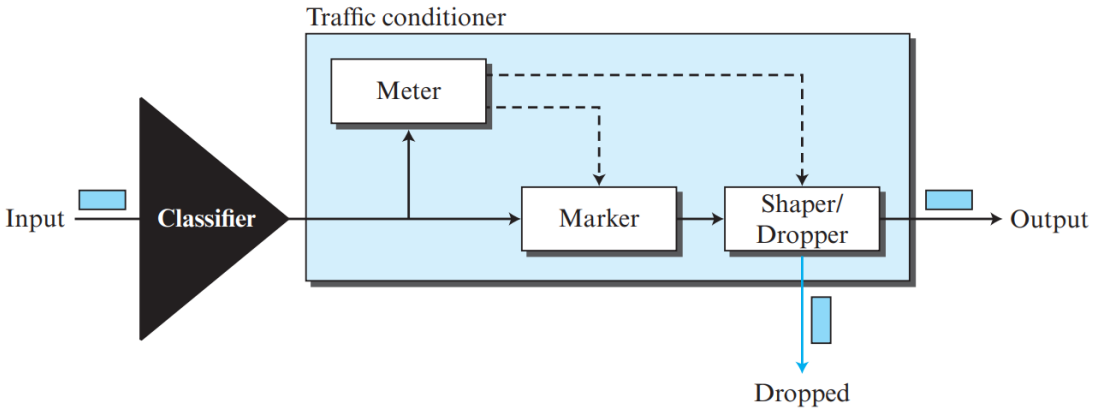
Per-hop behaviours (PHB):

PHB	Meaning
DE PHB (default)	Same as **best-effort** delivery; compatible with ToS
EF PHB (expedited forwarding)	**Low loss, low latency, ensured bandwidth**; like a virtual connection between source and destination
AF PHB (assured forwarding)	Delivers with **high assurance** as long as the class traffic stays within the node's **traffic profile**. Users must know some packets may be discarded

Traffic conditioner:

A DS node uses these components to implement DiffServ:

Component	Function
Meter	Checks whether the incoming flow matches the **negotiated traffic profile** and reports the result to other components (can use a token bucket)
Marker	**Remarks** a best-effort packet (DSCP 000000) or **down-marks** a packet (lowers the class) if the flow violates the profile. It **never up-marks**
Shaper	**Reshapes** non-compliant traffic using the meter's information
Dropper	A shaper with **no buffer**: **discards** packets if the flow **severely violates** the profile

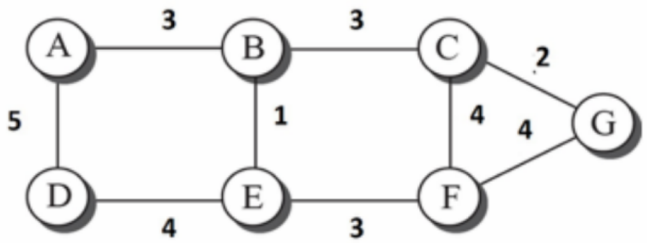


About Figure: Input packets pass through a classifier, a meter, then a marker and a shaper/dropper, producing the output. Dropped packets leave the flow.

Parameter	Integrated Services	Differentiated Services
Basis	**Flow-based**	**Class-based**
Resource handling	Explicit **reservation** (RSVP)	Marking by class; no per-flow state
Where processing is done	At every router, including the **core**	At the **edge**; core routers just apply the PHB
Marking	Routers mark packets by flow characteristics	**Applications** mark packets by priority
Service types	Two: guaranteed, controlled-load	Many classes (DE, EF, AF and more)
Signalling	Needs RSVP and a connection-oriented layer	None
Main problem	**Scalability** and service-type limitation	Less strict guarantees (for example AF packets may be discarded)

MODEL QUESTIONS

1. Explain the working of the Stop-and-Wait protocol. Describe the sender's handling of a lost packet and of a lost acknowledgement. (3 Marks)
2. An organization is given the address 192.168.10.75/26 for one of its hosts. Find (a) the subnet mask (b) the network address and the broadcast address of the subnet (c) the range of usable host addresses and the number of usable hosts. (3 Marks)
3. Explain the establishment of a TCP connection using the three-way handshake. Justify the statement that a SYN segment consumes one sequence number. (6 Marks)
4. A company runs three services over the Internet: (i) an online fund-transfer application, (ii) a live video-conferencing application, and (iii) a DNS query from a host to its local DNS server. For each service, choose the suitable transport protocol (TCP or UDP) and justify your choice. (3 Marks)
5. All the routers in a network run a link state routing protocol and have flooded their link state packets, so every router holds the same link state database. The database corresponds to the network shown in the figure, with the numbers on the links denoting the link costs. Using Dijkstra's algorithm, find the least-cost path from router F to every other router, and write the routing table (destination, cost, next hop) of F. (6 Marks)



6. An IP datagram of total length 4000 bytes (including a 20-byte header) must pass through a link with an MTU of 1500 bytes. Find the number of fragments. For each fragment, give the total length, the fragment offset and the MF (more fragments) flag. (3 Marks)

© NOTES BY CLEARITT LEARNING
SUBSCRIBE ON YOUTUBE | DONATE TO US