

Datadcommunicatie

Bachelor in de Industriële Wetenschappen: elektronica-ICT
3^e jaar, 6^e semester

3Ba E / SP E

Ba3.EL.05 / SPBa3.EL.05

EmSD
Embedded System Design

ir. J. Meel
april 2008

Bachelor in de Industriële Wetenschappen: elektronica-ICT
3^e jaar, 5^e / 6^e semester

Datacommunicatie

Ba3.EL.05 (SPBa3.EL.05)

ir. J. Meel

1 Doelstelling

De student kent de basisbegrippen en werkingsprincipes van belangrijke hedendaagse datacommunicatiesystemen (LAN, Internet, cellulaire netwerken, industriële netwerken) en kan een netwerk dimensioneren.

2 Competenties

Bachelorcompetenties	Ba Comp-Matrix
kent het gelaagde communicatiemodel en kan hedendaagse protocols hierin situeren	KI3
kent de basisbegrippen van datacommunicatie	KI3
kent de algemene werkingsprincipes van datacommunicatiesystemen en kan deze herkennen in hedendaagse protocols	KI3
kent de werkingsprincipes en uitvoeringsvormen van LAN netwerken	KI3
kent de voornaamste internetprotocols	KI3
kent de basiswerkingsprincipes van cellulaire netwerken	KI3
kan op basis van de basisbegrippen van datacommunicatie en de algemene werkingsprincipes van datacommunicatiesystemen de werking van ongekende protocols onderzoeken	AV2, IV1, OV1, SV1
kan relevante verschillen en gelijkenissen tussen protocols identificeren	AV2, OV1

3 Inhoud

Datacommunicatie bestudeert het proces van overdracht van data tussen twee of meer entiteiten (computers, dienstenterminals). Data kan een brede waaier aan informatie voorstellen: gedigitaliseerde spraak of beelden, meetresultaten, een computer bestand, Waar historisch een telecommunicatienetwerk werd ontwikkeld voor een specifieke dienst (telefonie, TV-distributie, computerdata, ...), is er nu een tendens om binnen een netwerk meerdere diensten te integreren.

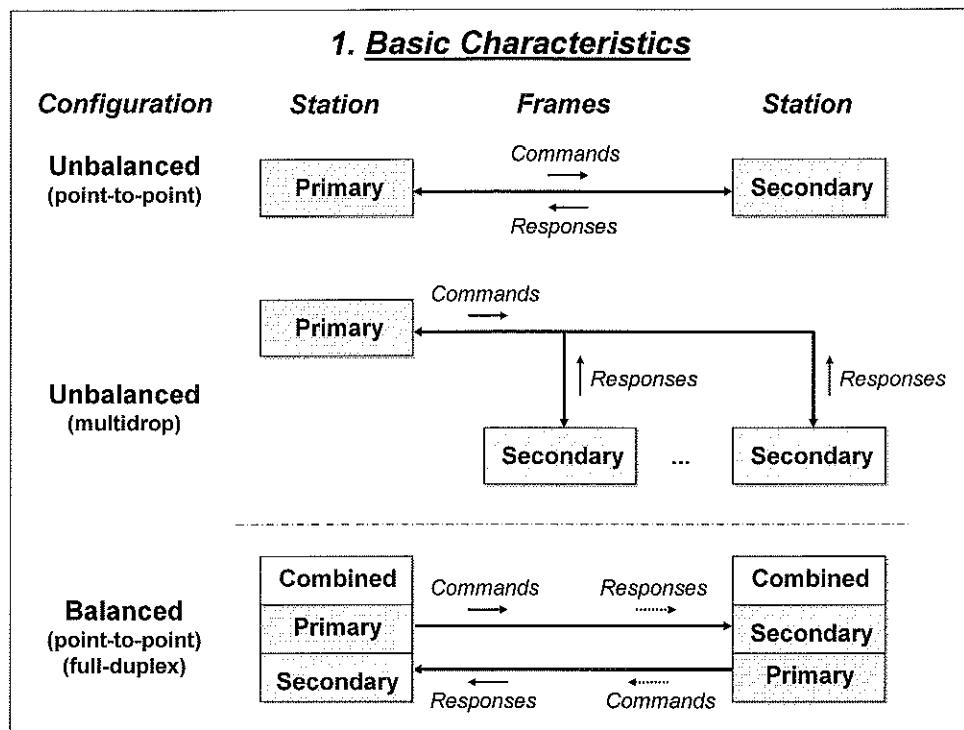
Communicatiesystemen worden omwille van hun complexiteit opgesplitst in lagen. Basisbegrippen van datacommunicatie worden aangebracht, belangrijke standaarden worden bestudeerd en praktisch geïllustreerd. In Datacommunicatie wordt het volledige OSI-model geïntroduceerd, maar blijft de gedetailleerde uitwerking beperkt tot de lagen 1 t.e.m. 4, met andere woorden van de fysische laag tot en met het TCP/IP-protocol.

- 1 Inleiding tot datacommunicatie
 - 1.1 evolutie van netwerken (telefonie, TV, data) naar ISDN, ATM
 - 1.2 basisbegrippen
- 2 Gelaagd communicatiemodel (OSI en TCP/IP)
- 3 Data transmissie
 - 3.1 asynchrone vs. asynchrone transmissie
 - 3.2 technieken voor duplex (2 richtingen) en meervoudige toegang op eenzelfde medium
 - 3.3 bronnen van data en hun frequentieinhoud (spectrum)
 - 3.4 degradatie van het signaal door het transmissiemedium
 - 3.5 basisband en banddoorlaat transmissie
- 4 Transmissie media (UTP, STP, coax, fiber)
- 5 Standaard interfaces
 - 5.1 serieel (RS-232)
 - 5.2 parallel (IEEE 1284)
- 6 Data link protocol
 - 6.1 karaktergeoriënteerd (BISYNC)
 - 6.2 bitgeoriënteerd (HDLC)
- 7 LAN netwerken
 - 7.1 topologie (ster, ring, bus)
 - 7.2 medium access control (MAC)
 - 7.3 IEEE 802 standaarden
- 8 Internet protocols (TCP/IP protocol stack)
 - 8.1 Internet Protocol (IP) en routing
 - 8.2 Transmission Control Protocol (TCP) en User Datagram Protocol (UDP)
- 9 Cellulaire netwerken
 - 9.1 2G systemen: GSM en CDMA (IS-95)
 - 9.2 2+G systemen: EDGE, GPRS
 - 9.3 3G systemen: UMTS



High-Level Data Link Control HDLC (ISO 33009, ISO 4335)

- synchronous
- bit-oriented
- connection-oriented
- full-duplex
- transparent-mode
- point-to-point, multipoint, multidrop



Stations

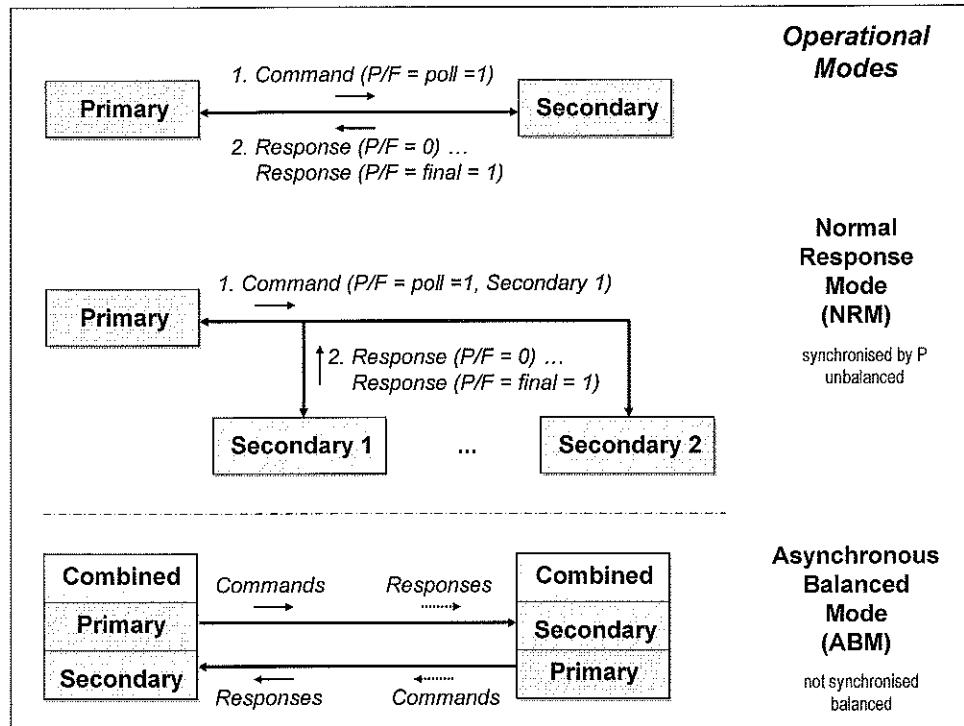
- **Primary:** controls the operation of the link, only one active at a time
- **Secondary:** controlled by the primary station, multiple active at a time

Frames

- **Commands:** frames from Primary to Secondary (P/F bit = Poll)
- **Response:** frames from Secondary to Primary (P/F bit = Final)

Link Configurations

- **Unbalanced:** One primary and one (point-to-point link) or more (multipoint link) secondary station
- **Balanced:** Two combined stations (point-to-point link)



Operational Modes (Transfer Modes)

- **Normal Response Modes (NRM):**

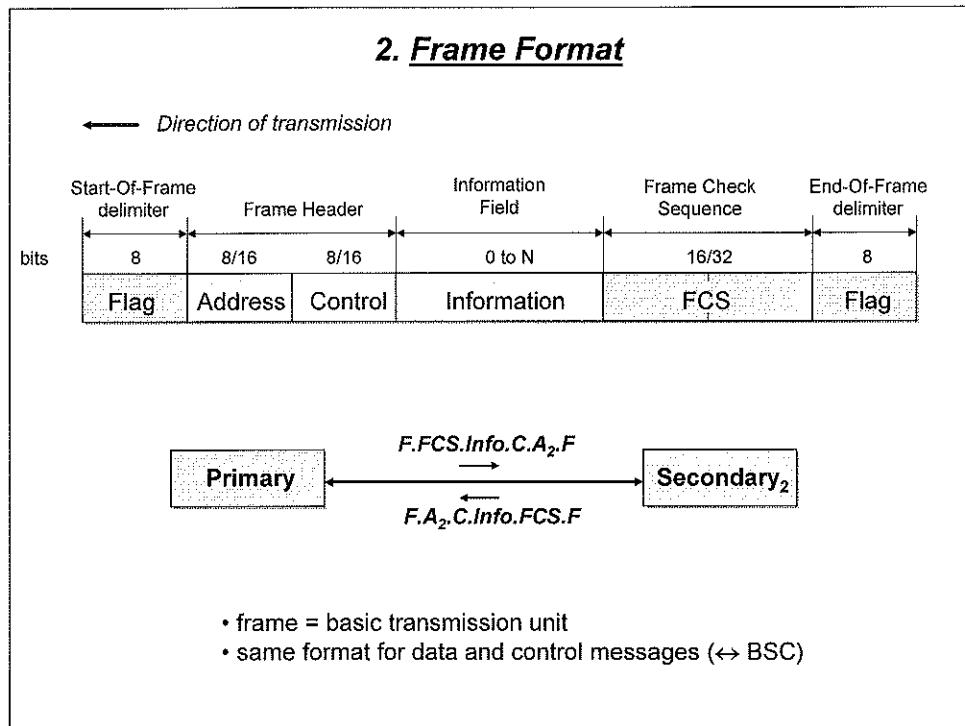
- Used with an unbalanced configuration.
- The Primary may initiate data transfer to a Secondary.
- A Secondary can only transmit after receiving a *poll* command ($P/F=1$) addressed to it by the primary. It may then send a series of responses, but after it indicates a *final* response ($P/F=1$), it cannot transmit any more until it receives another *poll*.

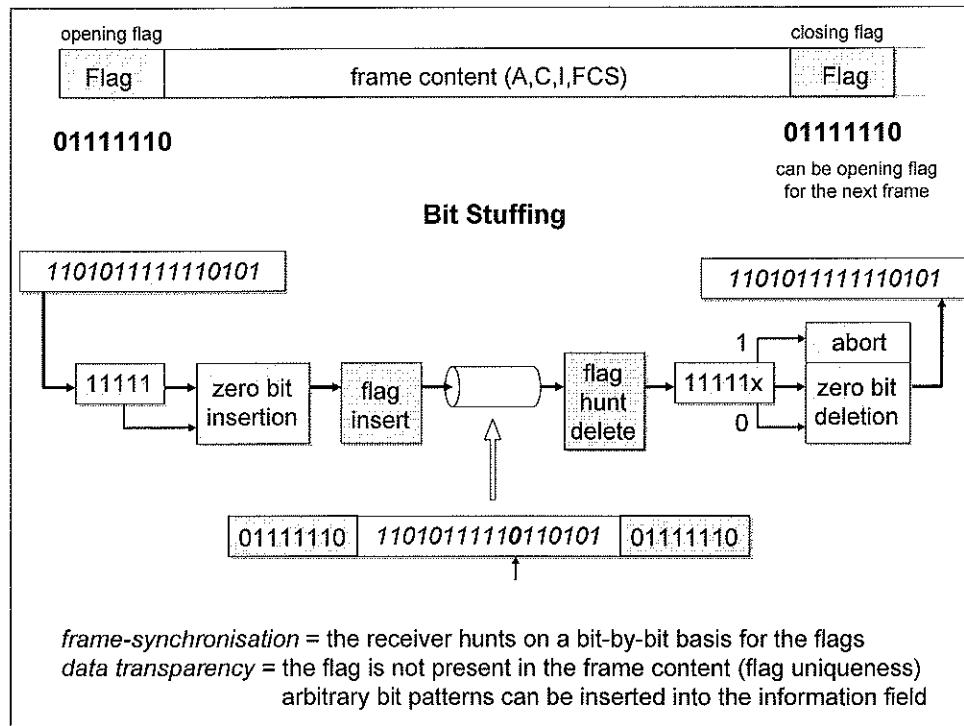
- **Asynchronous Balanced Mode (ABM):**

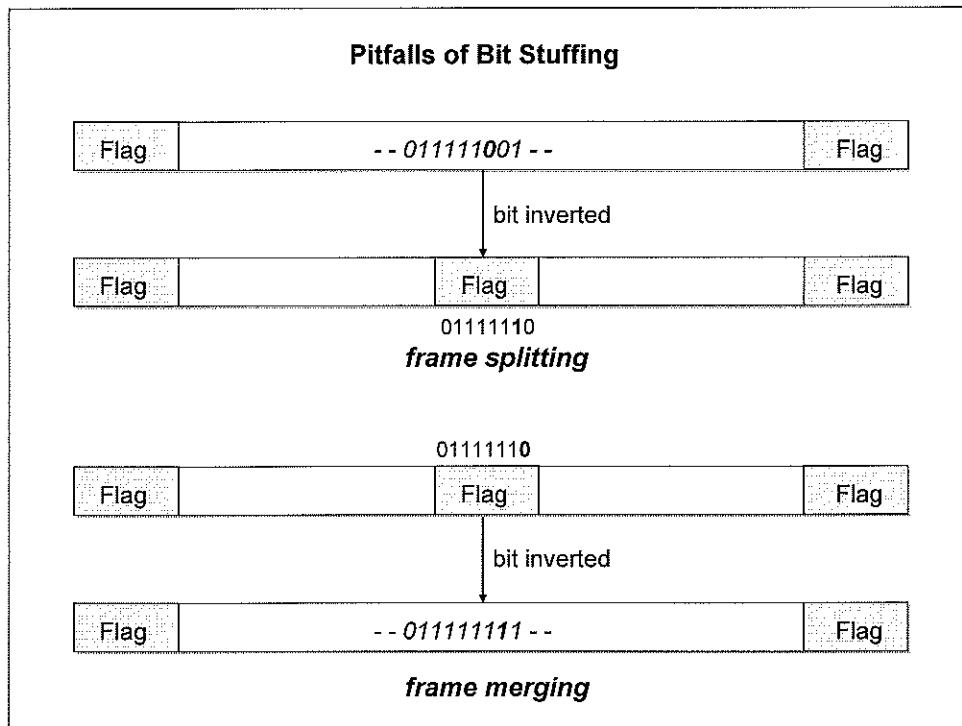
- Used with a balanced configuration (point-to-point).
- Either end can initiate transmission without waiting for a *poll* command from the other end (asynchronous). P and F bits must still be paired. That is, a P bit set by a Primary station must be paired with an F bit received from a Secondary station.

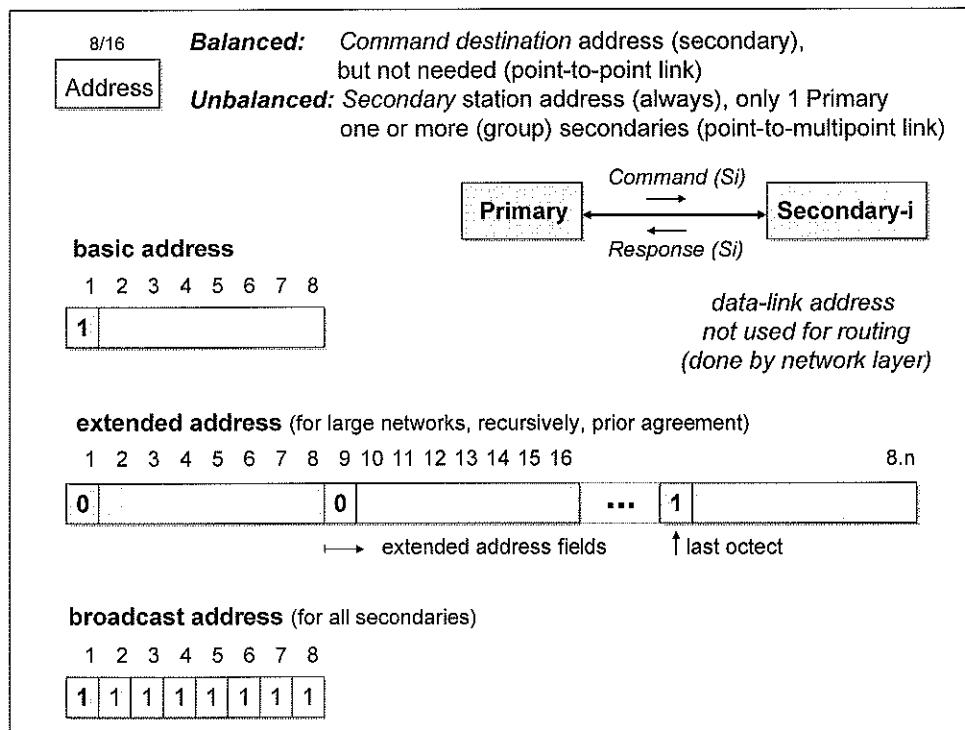
- **Asynchronous Response Mode (ARM):**

- Used with an unbalanced configuration.
- Rarely used.
- With one Primary and one active Secondary (selected by the Primary) either active station may initiate transmission at any time (asynchronous) without waiting for a *poll* (P) or *final* (F) bit.







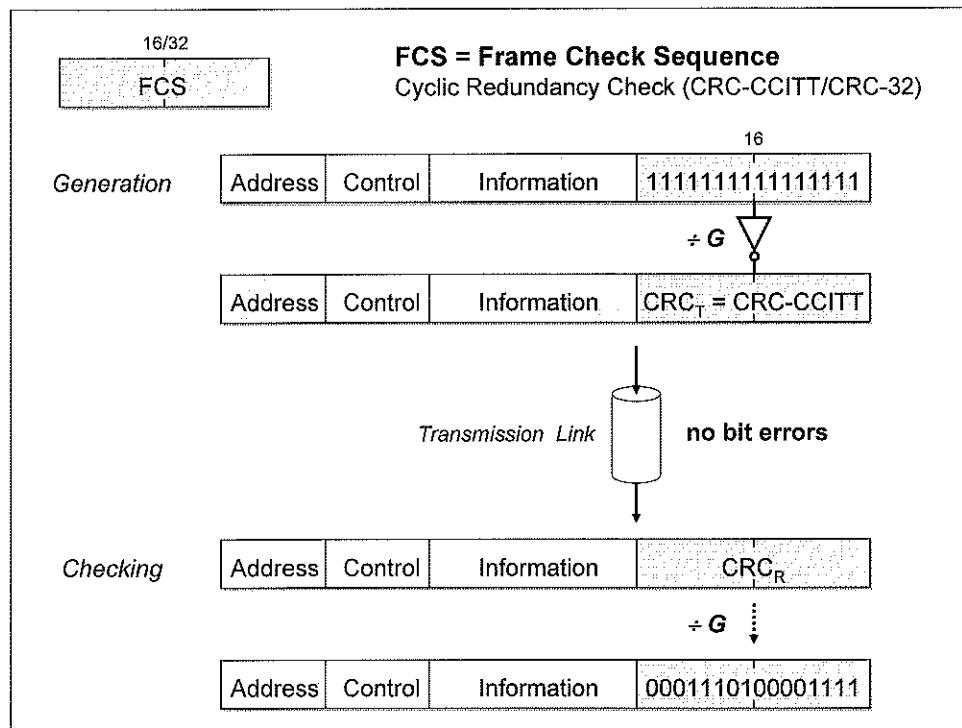


Unbalanced

The address field always contains the address of the secondary.

Multipoint link

- On a multipoint link there may be several secondary stations, but only one primary is allowed and cannot change. Every secondary station is assigned a unique address. Whenever the primary station communicates (send/receive) with a secondary, the address field contains the address of the secondary. This identifies the only station that can vary. Command frames are always sent with the receiving station's address. Response frames are always sent with the sending station's address.
- Group addresses can be used to address several stations.
- A broadcast address can be used to transmit a frame to all secondary stations on the link.
- An *extended address format* is available, in which the actual address is a multiple of 7 bits. The first bit of the final address field octet is one, while the first bit of each preceding address field is zero, so the extended address field is recursively extendible.



The Frame Check Sequence is a 16-bit (32-bit) CRC for the complete contents enclosed between the two flag delimiters. The generator polynomial used with HDLC is normally CRC-CCITT (16 bit) or CRC-32 (32 bit).

The FCS generation procedure is enhanced to make the check more robust. In the 16-bit case, 16 ones (instead of zeros) are added to the tail of the dividend prior to division, and inverting the remainder. This has the effect that the remainder computed by the receiver is not all zeros but a special bit pattern - 0001 1101 0000 1111.

8/16															
Control	The control field identifies the type and the function of the frame.														
<i>I-frame:</i> Information-frame	→ sequence numbers (CO) = multiple frame mode → user information (layer 3) and piggyback ACK														
<i>S-frame:</i> Supervisory-frame	→ sequence numbers (CO) → error and flow control (ARQ)														
<i>U-frame:</i> Unnumbered-frame	→ no sequence numbers (CL) → link management and link information (L2 + UI=L3)														
Control															
<i>I-frame</i>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Flag</td><td>Address</td><td>0</td><td>N(S) N(R)</td><td>User-Info (L3)</td><td>FCS</td><td>Flag</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	Flag	Address	0	N(S) N(R)	User-Info (L3)	FCS	Flag							
Flag	Address	0	N(S) N(R)	User-Info (L3)	FCS	Flag									
<i>S-frame</i>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Flag</td><td>Address</td><td>1</td><td>0</td><td>N(R)</td><td>FCS</td><td>Flag</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	Flag	Address	1	0	N(R)	FCS	Flag							
Flag	Address	1	0	N(R)	FCS	Flag									
<i>U-frame</i>	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td>Flag</td><td>Address</td><td>1</td><td>1</td><td>Control-Info (L3)</td><td>FCS</td><td>Flag</td></tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	Flag	Address	1	1	Control-Info (L3)	FCS	Flag							
Flag	Address	1	1	Control-Info (L3)	FCS	Flag									

The control field identifies the function and the purpose of the frame.

I-frame ($C_1 = 0$)

Information frames are used for data transfer of *layer 3 information* in a connection oriented (CO) way: multiple frame mode (frames are numbered and acknowledged). For this purpose send N(S) and receive N(R) sequence numbers are included. The data may be of any length and may consist of any code or grouping of bits.

S-frame ($C_1 C_2 = 10$)

Supervisory frames are used for error and flow control and hence contain receive sequence numbers N(R) for acknowledgment.

U-frame ($C_1 C_2 = 11$)

Unnumbered frames are used to transfer control/data information in a connectionless (CL) way. They do not contain sequence numbers.

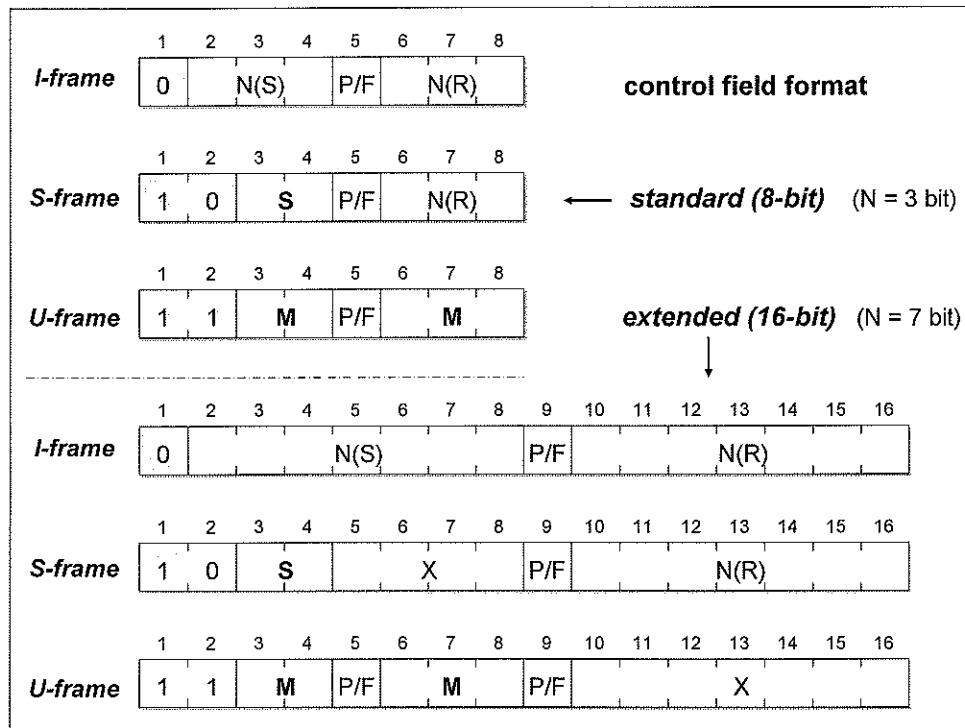
N(S) and N(R): sequence numbers that support error and flow control.

- N(S) = *send sequence number*

Only for I-frames. The number of the frame being transmitted, to uniquely identify the frame.

- N(R) = *receive sequence number*

For both I and S-frames. The number of the next frame the receiver expects, used for acknowledgment.



N(S) and N(R) bit-length

- **standard format**

N = 3 bit. Sequence numbers can range from 0 to 7 (modulo 8).

Maximum send window is 7.

- **extended format**

N = 7 bit. Sequence numbers can range from 0 to 127 (modulo 128).

Maximum send window is 127. In applications with very long links (satellite links) or very high bit rates, require a larger send window if a high link utilization is to be achieved.

P/F bit (poll/final)

In a *command* (sent from primary to secondary) the P/F bit is a *poll* bit used to request an acknowledge from the secondary. The P=1 is used by a data link layer entity to solicit (poll) a response frame from the peer data link entity.

In a *response* (sent from secondary to primary) the P/F bit is a *final* bit indicating the acknowledge from the secondary and the end of a sequence of frames. F=1 is used by a data link entity to indicate the response frame transmitted as a result of a soliciting (poll) command

S-bits (supervisor)

Indicate supervisory functions: RR(00), REJ(01), RNR(10), SREJ(11).

M-bits (modifier)

M-bits specify the unnumbered frame type (not all combinations used).

X-bits

reserved bits, set to 0

Type	Name		C/R	Description
I	Information		C/R	exchange CO 'user info' (L3)
S	RR RNR REJ SREJ	Receive Ready Receive Not Ready Reject Selective Reject	C/R C/R C/R C/R	positive ack (start Tx) positive ack (stop Tx) negative ack, go-back-N negative ack, selective reject
U	SNRM(E) SARM(E) SABM(E) DISC	Set Normal Mode (Extended) Set Asyn Response Mode (E) Set Asyn Balanced Mode (E) Disconnect	C	Link Management (Modes) set NRM (N=3-bit; E:N=7-bit) set ARM (N=3-bit; E:N=7-bit) set ABM (N=3-bit; E:N=7-bit) terminate logic link connection
	UA DM	Unnumbered Acknowledge Disconnect Mode	R	mode command accepted (CL) slave stays disconnected
	RSET	Reset	C	Recovery reset V(R) and V(S)
	FRMR	Frame Reject	R	reports receipt of wrong frame
	UI	Unnumbered Information	C/R	Data Transfer exchange CL 'control info' (L3)

Receive Ready (RR): positive acknowledge of received I-frames, N(R) not included; the receive station is ready to receive

Receive Not Ready (RNR): positive acknowledge of received I-frames, N(R) not included; the receive station is not ready to receive (busy)

Reject (REJ): request retransmission (=negative acknowledge) of all I-frames starting from N(R); positive acknowledge of all previously received I-frames, N(R) not included

Selective Reject (SREJ): request retransmission of the single I-frame with sequence number N(R)

Mode Set commands (SNRM, SARM, SABM): initialize [V(S)=V(R)=0] and establish a logical connection with modulo-8 sequence numbering mode

Extended Mode Set commands (SNRME, SARME, SABME): initialize [V(S)=V(R)=0] and establish a logical connection with modulo-128 sequence numbering mode

Disconnect command (DISC): logically terminate an established connection mode

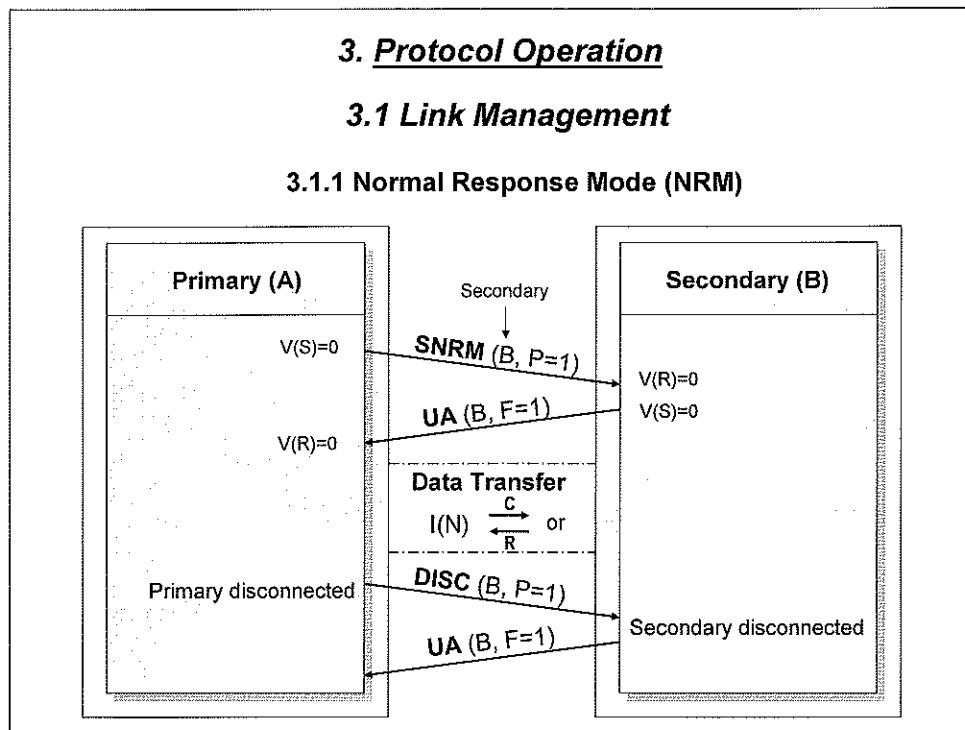
Unnumbered Acknowledge response (UA): acknowledge receipt and execution of a Mode Set, Disconnect, Reset command

Disconnect Mode response (DM): refuse Mode Set (station is logically disconnected)

Reset command (RSET): reset the send sequence variable V(S) at the transmitting station and the receive sequence variable V(R) at the receiving station to zero

Frame Reject response (FRMR): indicate and report to the transmitting station that invalid, non recoverable conditions are detected: (1) invalid control field, (2) I-frame with an I-field that exceeds the maximum length, (3) sequence number out of order

Unnumbered Information command/response (UI): send control information to one or more stations, without an established connection (connectionless, no N)



Normal Response Mode (NRM)

- Logic Connection Setup service (confirmed service)

In a multidrop link, an **SNRM**-frame is first sent by the primary station with the poll bit set to 1 ($P=1$) and the address of the appropriate secondary (B) in the address field. The secondary responds with a **UA**-frame with the final bit set ($F=1$) and its own address (B) in the address field. Since this acknowledge does not relate to an I-frame, it does not contain a sequence number (unnumbered acknowledge).

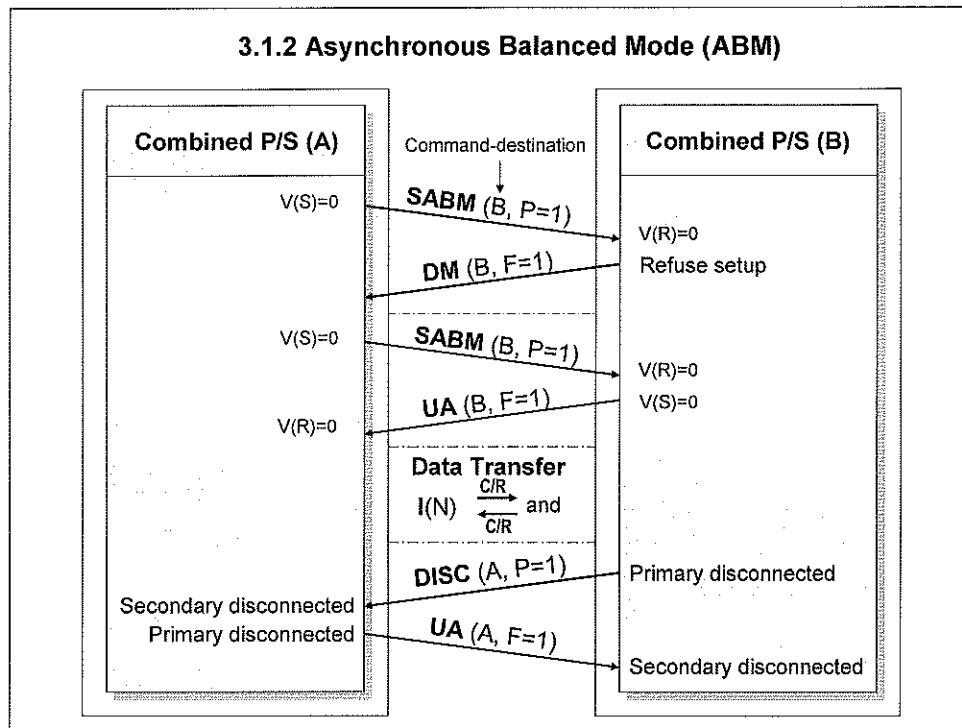
The setup procedure has the effect of initializing the sequence variables $V(S)$ and $V(R)$ held by each station. These variables are used in the error and flow control procedures.

- Date Transfer

When the logic connection (= link) is setup (active), a connection-oriented transfer of numbered I-frames takes place.

- Logic Connection Clear service (confirmed service)

Finally, after all data has been transferred, the logic connection (= link) is cleared when the primary sends a **DISC**-frame and the secondary responds with a **UA**-frame.



Asynchronous Balanced Mode (ABM)

The procedure followed to set up a point-to-point link is the same as that used for a multidrop link.

- Logic Connection Setup service (confirmed service)

A **SABM**-frame is sent first. In this mode, both sides of the link may initiate the transfer of I-frames independently, so each station is often referred to as a combined station, since it must act as both a primary and a secondary. Either station may initiate the setting-up of the link in the ABM mode. Station A initiates link setup in the given example. The station B acknowledges the connection setup with an **UA**-frame. A single exchange of frames sets up the link in *both* directions. The address field is used to indicate the direction of the command frame (SABM/DISC) and its associated response.

- Logic Connection Clear service (confirmed service)

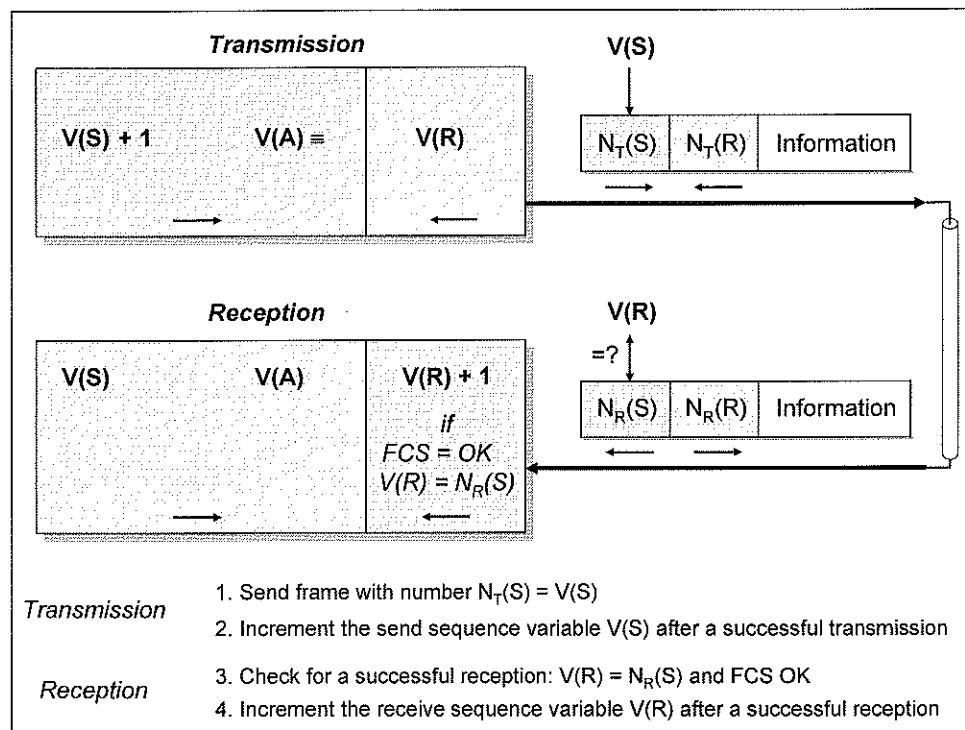
Either station may initiate the clearing of the link in this mode. In the given example station B initiates the clearing with the **DISC**-frame. Station A acknowledges the disconnection with an **UA**-frame.

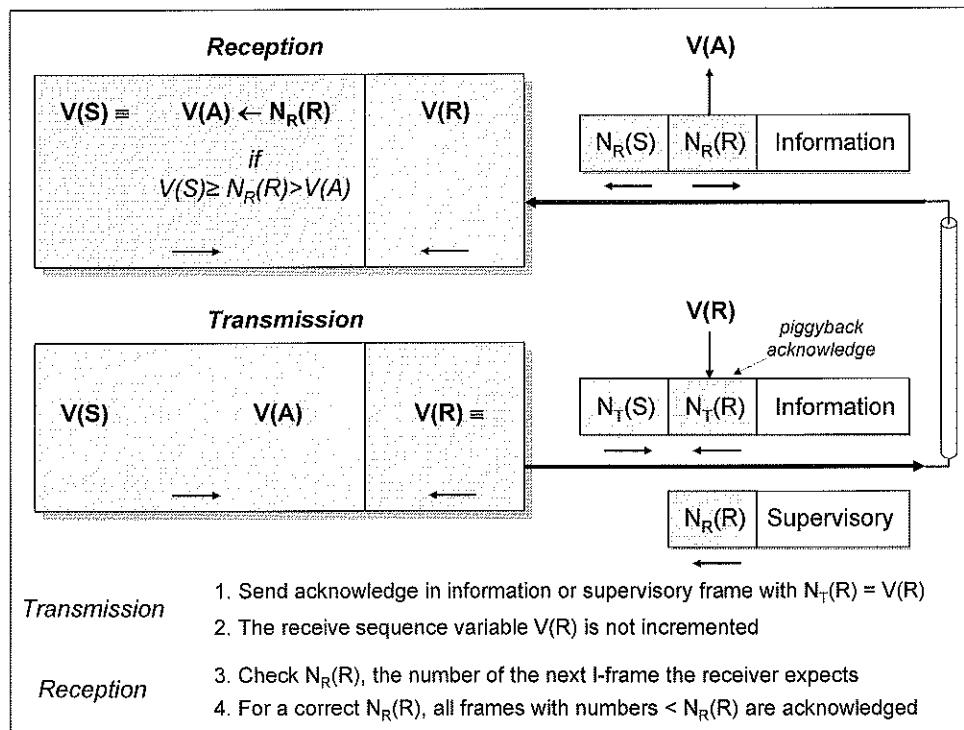
If the receiver wishes to refuse a setup command in either mode, a disconnected mode **DM**-frame is returned in response to the initial mode setting frame (SNRM or SABM). The DM-frame indicates the responding station is logically disconnected.

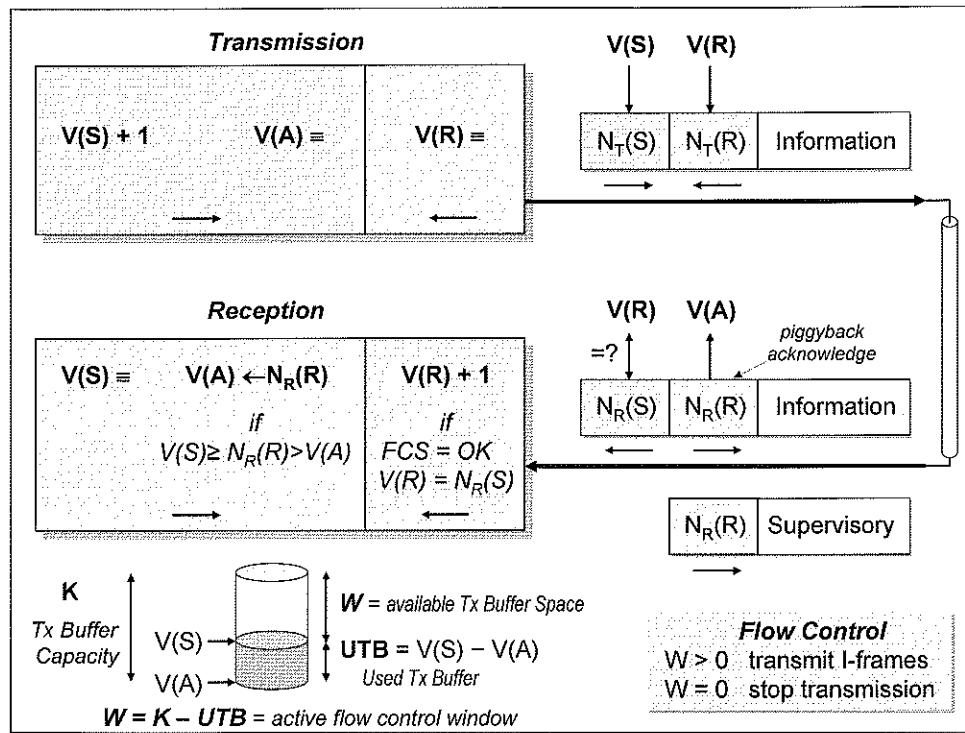
3.2 Data Transfer

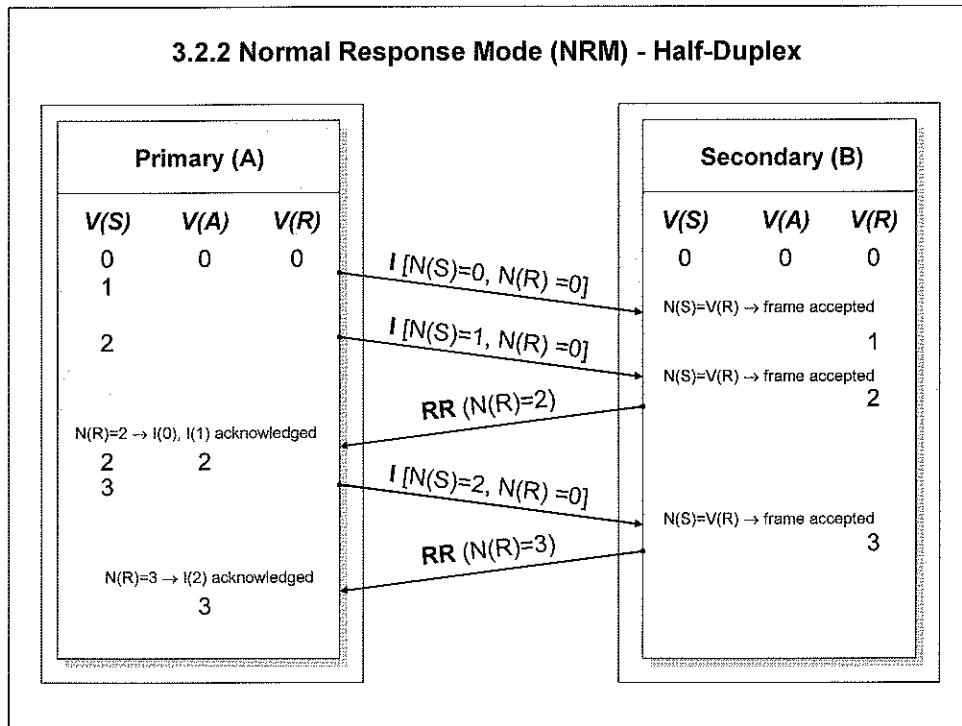
3.2.1 Definition and Function of Sequence Numbers/Variables

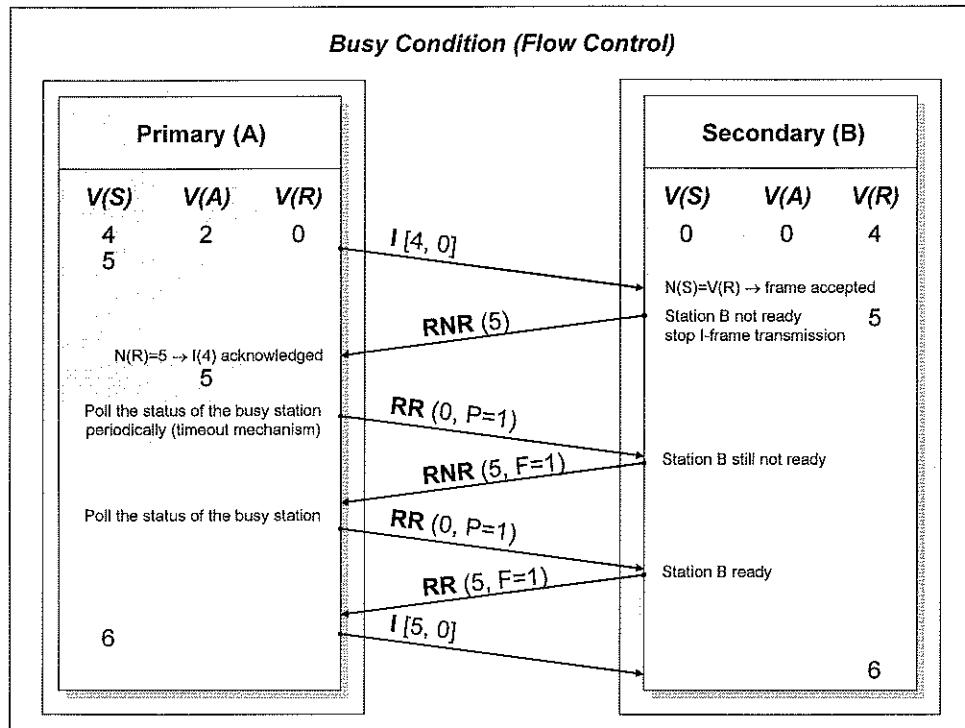
- **$N(S)$** = **Send sequence number (I-frames)**
= sequence number of the 'current' transmitted I-frame
- **$N(R)$** = **Receive sequence number (I- and S-frames)**
= acknowledge that I-frame sequence numbers $< N(R)$ are received correctly
- **$V(S)$** = **Send sequence variable**
= send sequence number $N(S)$ of the 'next' I-frame to be *transmitted*
→ incremented by 1 with the transmission of each successive I-frame
- **$V(A)$** = **Acknowledge sequence variable**
= equals $N(S) + 1$ of the last *transmitted* I-frame acknowledged by its peer
('next' I-frame expected to be *acknowledged*)
→ updated by the valid $N(R)$ received from its peer [in I- or S- frame]
- **$V(R)$** = **Receive sequence variable**
= send sequence number $N(S)$ of the 'next' I-frame expected to be *received*
→ incremented by 1 with the receipt of an error-free, in sequence I-frame
whose $N(S)$ equals $V(R)$

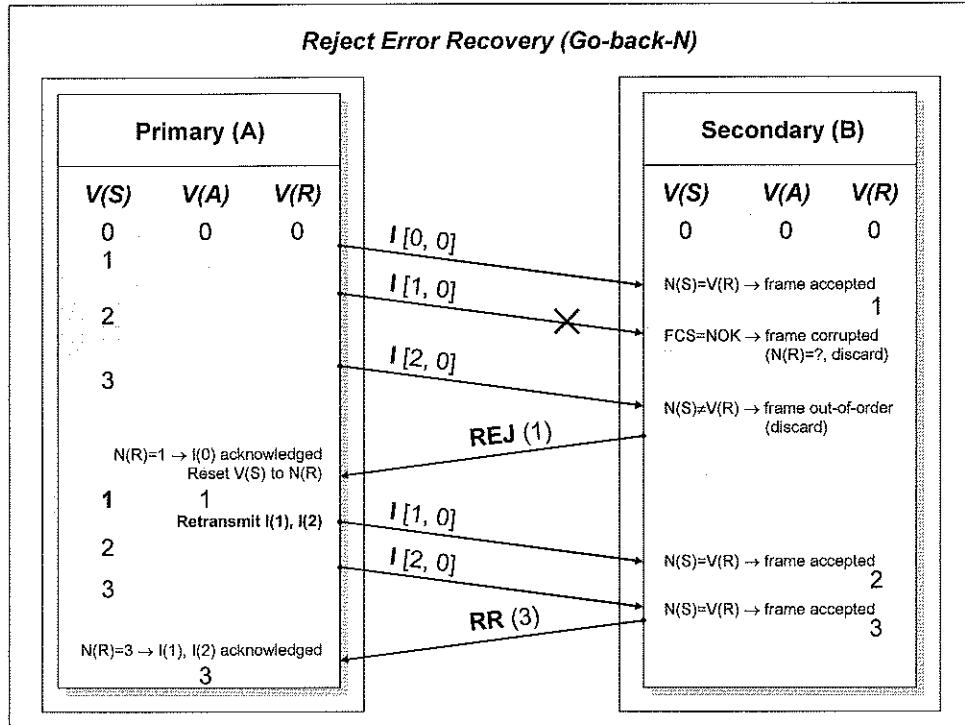


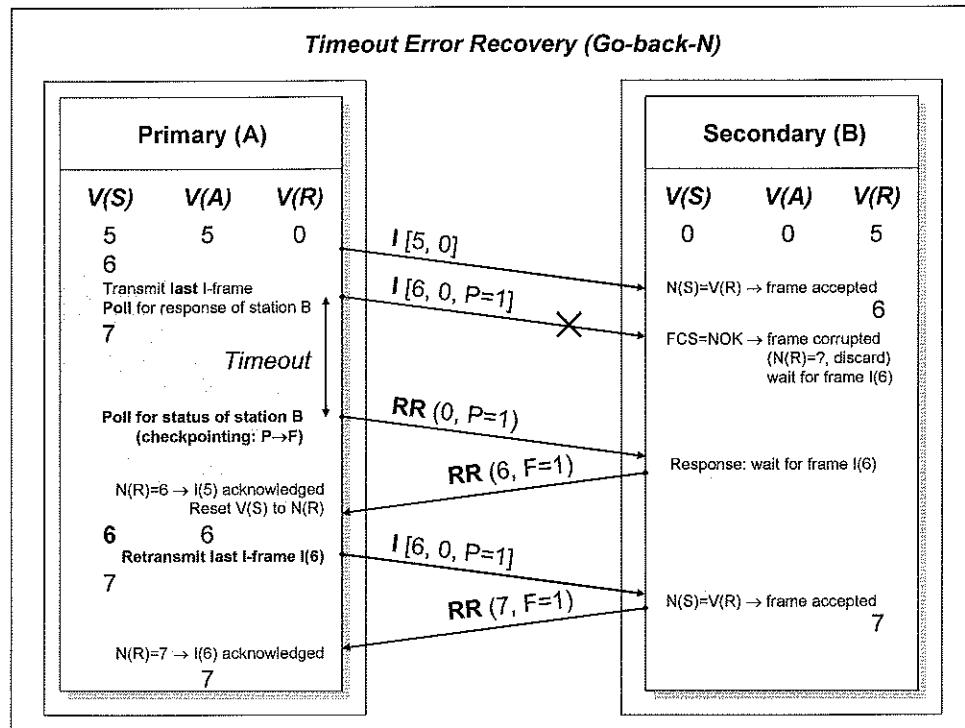


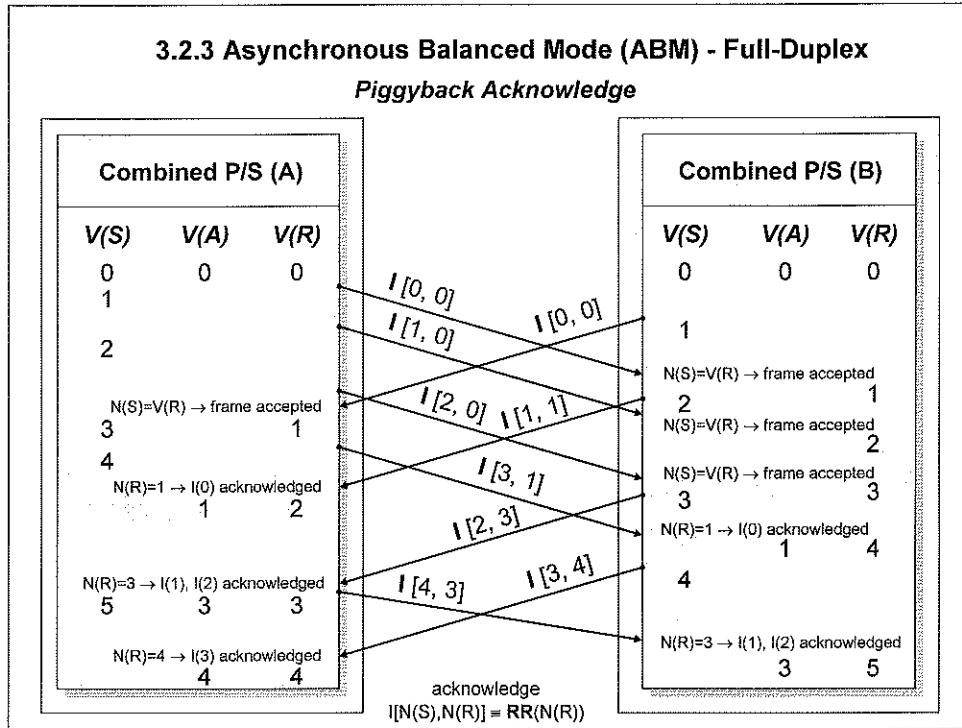






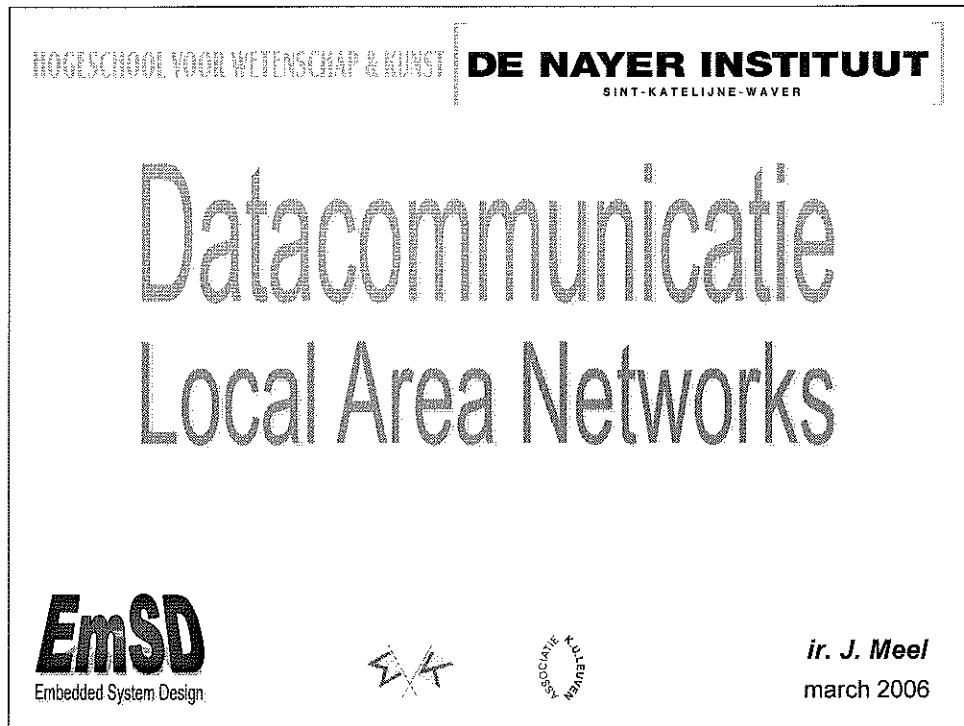


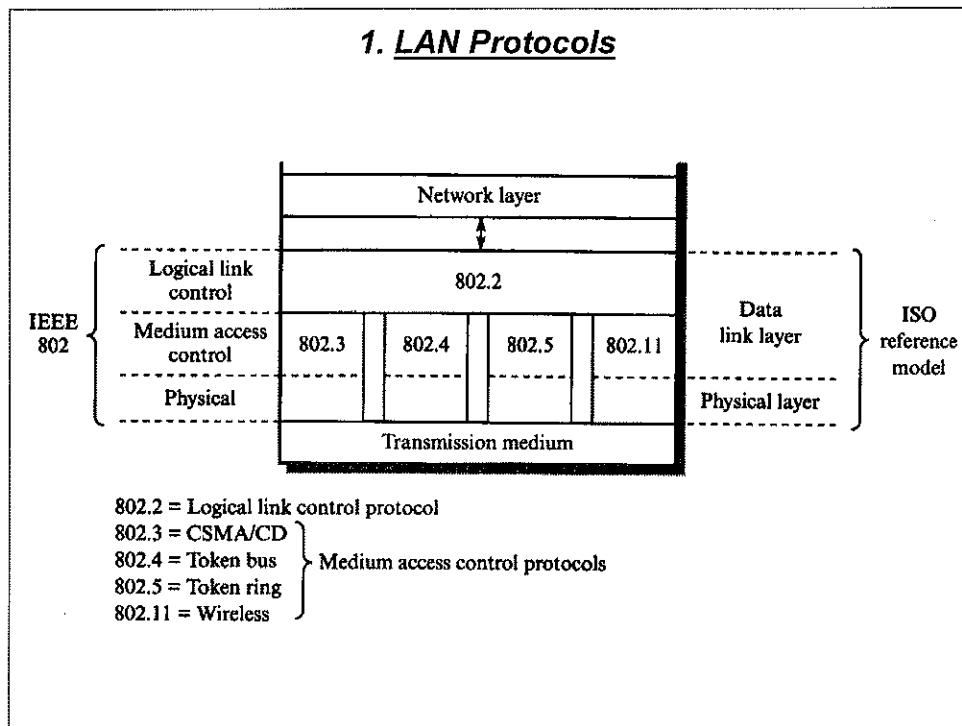




Piggyback acknowledge

A technique to return acknowledgment information across a full-duplex (two-way simultaneous) data link without the use of special (acknowledgment) messages. The *acknowledgment information* relating to the flow of messages in one direction is embedded (piggybacked) into a normal I-frame (data-carrying message) flowing in the reverse direction: $N(R) = \text{receive sequence number}$.





The various protocol standards for LANs, which deal with the physical and link layers in the context of the ISO reference model, are those defined in IEEE 802 (IEEE = Institute of Electrical and Electronics Engineers). This standard defines a family of protocols, each relating to a particular type of MAC (Medium Access Control) method. The basic MAC standards together with their associated physical media specifications are contained in the following IEEE standards documents:

- IEEE 802.3: CSMA/CD bus
- IEEE 802.4: Token bus
- IEEE 802.5: Token ring
- IEEE 802.11: Wireless

The relevant ISO standards are the same except an additional 8 is used: ISO 8802.3.

Although each MAC standard is different in its internal operation, they all present a standard set of services to the logical link control (LLC) layer, which is intended to be used in conjunction with any of the underlying MAC standards. In general, the various MAC and physical layers are normally implemented in firmware in special-purpose integrated circuits.

With a LAN the network, LLC, and MAC layers are peer (end-to-end) protocols, since there are no intermediate switching nodes within the network.

The MAC and LLC layers collectively perform the functions of the ISO data link.

The LLC protocol is based on HDLC. In almost all LAN installations, only send-data-with-no-acknowledge connectionless protocol is used. All data is transferred using the unnumbered information (UI) frame.

2. Ethernet - IEEE 802.3

2.1 Physical

2.1.1 Medium

Characteristic	Speed	Maximum Segment Length	Nodes per segment	Medium	Topology
10 BASE 5	10 Mbps	500 m	100	50 Ω coax ($\phi=10$ mm=Thick)	Bus
10 BASE 2	10 Mbps	185 m	30	50 Ω coax ($\phi=5$ mm=Thin)	Bus
10 BASE T	10 Mbps	100 m	-	UTP	Star

History

The Ethernet LAN design was created in the mid-1970s as a result of experimental work performed by Xerox Corporation. Based on its popularity, Ethernet became a defacto standard. Digital Equipment Corporation (DEC), Intel and Xerox [DIX] proposed the adoption of Ethernet as an IEEE 802 standard.

The IEEE 802.3 committee has been active in defining alternative physical configurations. This gives the user flexibility, but makes things complex.

To distinguish the various implementations that are available, the committee has developed a concise notation:

<data rate in Mbps> <signaling method> <maximum segment length in n .100 meters>

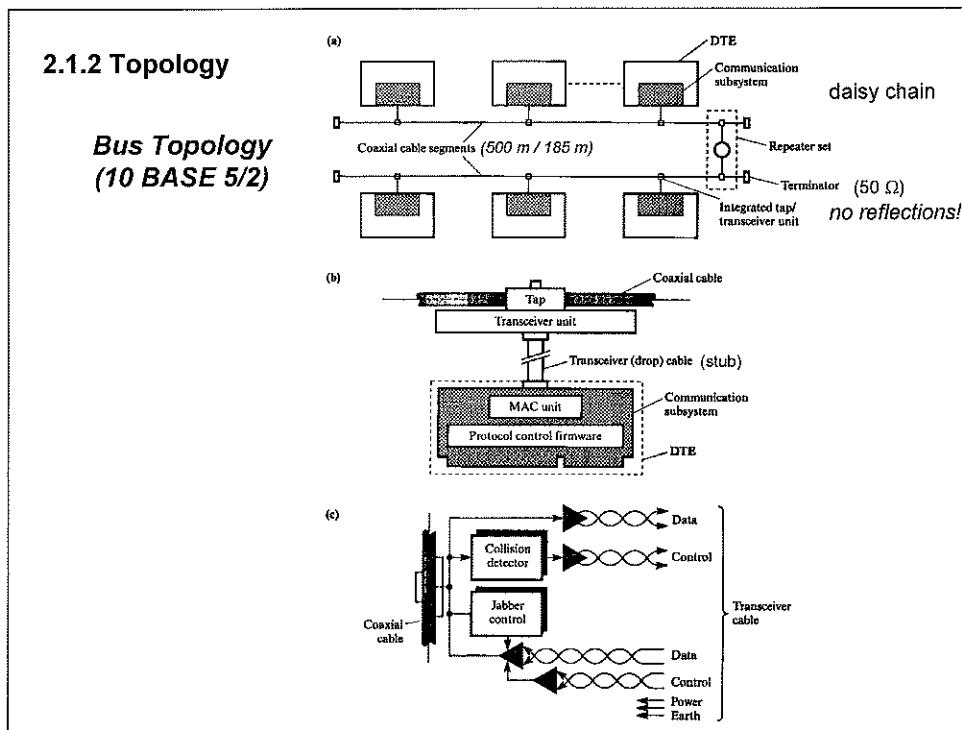
10 BASE 5

10 = 10 Mbps transmission rate

BASE = baseband operation

5 = 500 meters per segment

This is the original 802.3 medium specification and is based directly on the Ethernet. 10 BASE 5 specifies the use of 50-ohm coaxial cable with N-connectors and uses Manchester digital signaling. The maximum length of a cable segment is set at 500 meters. The length of the network can be extended by the use of repeaters. A repeater is transparent to the MAC level; as it does no buffering, it does not isolate one segment from another. So, for example, if two hosts on different segments attempt to transmit at the same time, their transmissions will collide. To avoid looping, only one path of segments and repeaters is allowed between any two hosts. The standard allows a maximum of four repeaters in the path between any two hosts, extending the effective length of the medium to 2.5 kilometers.



10 BASE 2

To provide a lower-cost system than 10 BASE 5 for personal computer LANs, the specification 10 BASE 2 was added. A thinner, thus more flexible, 50-ohm coax (RG 58) is used with a cheaper BNC connector (thin Ethernet, cheapernet). This cable supports fewer taps over a shorter distance than the 10 BASE 5 cable.

The coax must be terminated with a 50 ohm terminator at each end of the cable.

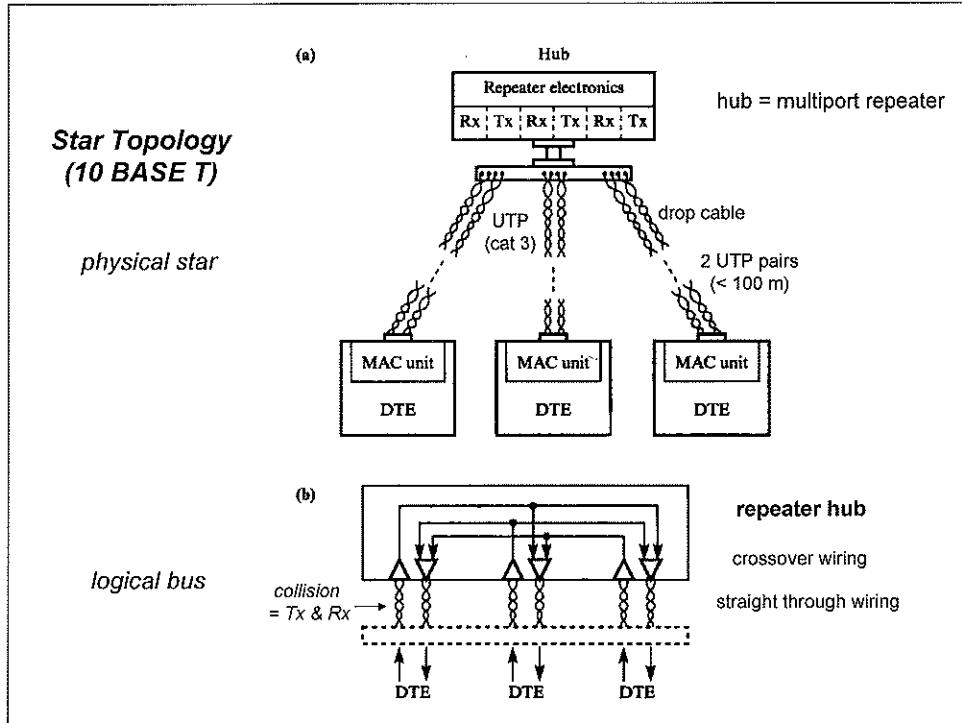
Also Manchester signaling is used.

In the 'daisy chain' topology a piece of thin coax is connected to the BNC T of a computer. This piece of coax is then attached to the BNC T on the next computer in line. The BNC T at the very end of the segment is the only one that should have a terminator.

In the daisy chain topology, if anyone incorrectly removes a thin Ethernet coax from the BNC T connector on the back of their computer, the entire segment will stop working for all other computers!

Because they have the same data rate, it is possible to combine 10 BASE 5 segments and 10 BASE 2 segments in the same network, by using a repeater that conforms to 10 BASE 5 on one side and 10 BASE 2 on the other side. The only restriction is that a 10 BASE 2 segment should not be used to bridge two 10 BASE 5 segments because a "backbone" segment should be as resistant to noise as the segments it connects.

Transceivers on coaxial cables should be able to read and write on a single pair of electrical conductors and detect the superposition of signals on the bus (collision), while at the same time disturbing the impedance of the medium to which they are attached as little as possible.



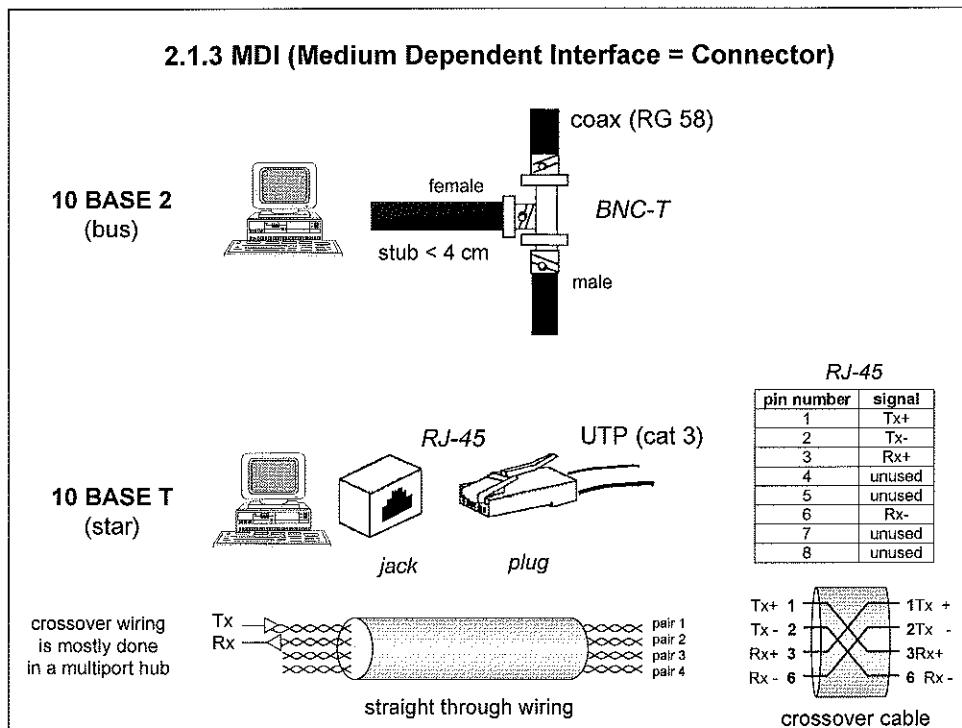
10 BASE T

By sacrificing some distance, it is possible to develop a 10-Mbps LAN using the unshielded twisted-pair (UTP) medium. Such wire is often found prewired in office buildings as excess telephone cable and can be used for LANs. The 10 BASE T specification defines a star-shaped topology. A simple system consists of a number of hosts connected via a point-to-point link to a central point, referred to as a *multiport repeater (hub)* or *concentrator*. The central point accepts input on any one line and repeats it on *all* of the other lines. Each link consists of two unshielded twisted pairs. Because of the high data rate and the poor transmission qualities of unshielded twisted pair, the length of a link is limited to 100 meters. As an alternative, an optical fiber link may be used. In this case, the maximum length is 500 m (10 BASE FP).

10 BASE T was designed to allow segments of approximately 100 meters in length when using 'voice grade' twisted-pair telephone wiring that meets the EIA/TIA category 3 wire specifications. The EIA/TIA cabling standard recommends a segment length of 90 meters between the termination equipment in the wiring closet, and the computer. This provides 10 meters of cable allowance to accommodate patch cables at each end of the link, signal losses in intermediate wire terminations on the link, etc.

While the 10 BASE T system is designed to use voice grade telephone cable (category 3) that may already be installed, many sites choose to install higher quality category 5 cables. These higher quality components work well for 10 BASE T and also provide a good signal carrying system for the 100 Mbps Ethernet media systems.

For transceivers operating on point-to-point links a collision corresponds to a situation of simultaneous transmission and reception (easier to detect than on coax). An imperfect adaptation does not necessarily harm the communication.



10 BASE 2

To make an attachment to a thin Ethernet segment, the 'female' BNC connector of the host is attached to the male end of a BNC T connector. The other two female ends of the BNC T make a physical and electrical connection to the thin Ethernet segment.

The standard notes that the length of the "stub" connection from the BNC (MDI) on the NIC (Network Interface Card) to the coaxial cable, should not be longer than 4 cm, to prevent the occurrence of signal reflections which can cause frame errors. While longer stub cables may seem to work, they actually create signal reflections (standing waves) which result in frame errors. This increases the number of retransmissions and decreases the throughput.

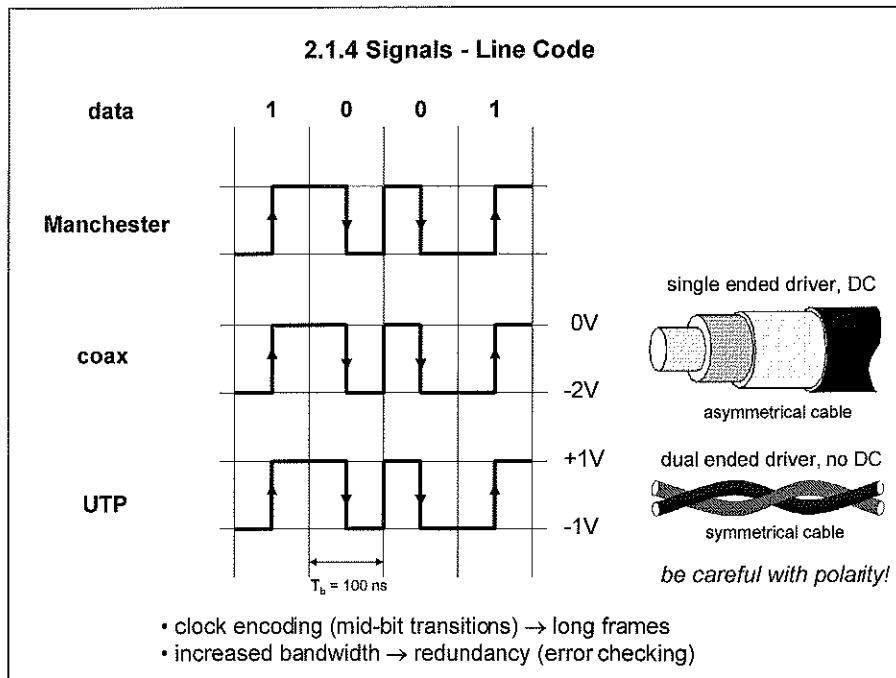
10 BASE T

This system uses two pairs of twisted wires, which are terminated in an 8-pin (RJ-45) connector. Thus 4 pins of the RJ-45 connector are used.

When connecting two UTP NICs together over a segment, the transmit data pins of one RJ-45 connector must be wired to the receive data pins of the other, and vice versa. The crossover wiring may be accomplished in two ways: with a special cable or inside the hub.

For a *single segment* connecting only two computers the signal crossover can be provided by a special crossover cable, with the transmit pins on the RJ-45 plug at one end of the cable wired to the receive data pins on the RJ-45 plug at the other end of the crossover cable and vice versa.

For *multiple segments* in a building it's much easier to wire the cable connectors "straight through" and to do all the crossover wiring at one point in the system: inside the multiport hub (recommend by the standard, the port should be marked with an X).



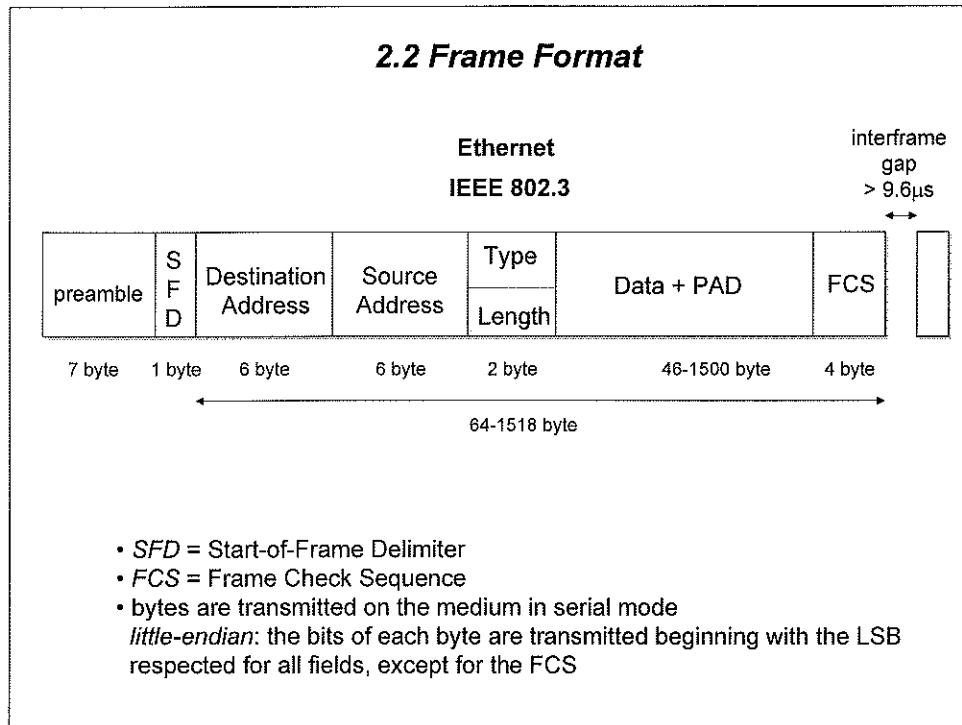
Manchester linecode

This is a code in which 1 is represented by a rising edge and 0 by a descending edge in the middle of a bit period T_b (100 ns). Thus, it cannot be read correctly by inverting the direction. The Manchester code guarantees one transition (rising or descending edge) per clock pulse, which is equivalent to transporting a synchronization signal. The DC component is always constant and may therefore be chosen to be zero. The signal spectrum is bunched between $1/T_b$ (10 MHz) and $2/T_b$ (20 MHz) with zero power at frequency zero, but it has a total width twice that of the NRZ (Non Return to Zero). The code has an efficiency of 50% since two levels are used to encode one bit of data. This code may be viewed as a code in phase ($-\pi/2$, $\pi/2$) of the signals to be transmitted, which also explains why it is called a biphase code.

The code used by baseband Ethernet is always of the Manchester type, whether on coaxial cable, twisted pair or fiber optic cable. However, the electrical levels are different:

- For coax the DC component is non-zero and the signal varies between 0 and -2 V.
- For UTP the signal is symmetric and varies between ± 1 V.

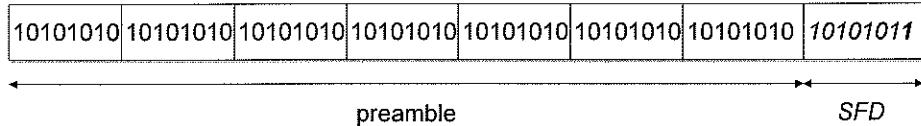
The Manchester code carries a polarity (the 1 and the 0 are distinguished by the direction of the edge), which implies that care must be taken not to invert the two conductors. This is easy when working on a coaxial cable (where the pair is naturally asymmetric), but is less straightforward for symmetric twisted pairs.



The frame is the elementary structure used to circulate data on the network. It is defined at the MAC level and follows a number of rules. For example, the fields constituting it identify the transmitting host, the destination host, and the type of data transported (indicated by the level 3 protocol). It also has an elaborate CRC control which is located in the last four bytes of the frame. Finally, the minimum and maximum lengths are regulated. Firstly by the need to always be able to detect collisions the length must be 64 bytes or longer. Secondly, by the desire not to transfer power to a single host for too long a period, the length must be limited to 1518 bytes.

The transmission is in serial mode on the medium and the useful data, which is usually represented in the form of 8-bit bytes, has to be arranged in a bit sequence. The bytes are transmitted in order, that is, respecting the ordering used by the layer above. The bits of each byte are transmitted beginning with the least-significant bit (0 or 1) and ending with the most-significant bit (factor of $128 = 2^7$).

2.2.1 Preamble + Start-of-Frame Delimiter



preamble

- 7 bytes = 56 bits
- sequence of successive 1s and 0s (no specific information)
- 'clock' synchronization: synchronization of the Rx clock on the Tx clock

Start-of-Frame Delimiter (SFD)

- 1 byte = 8 bits
- continues the preceding sequence, except the last bit set to 1
- 'frame' synchronization: indication of the start of the first significant frame-byte

If a *collision* is detected during transmission,
the transmitting host should continue to transmit the complete preamble + SFD.

Clock Synchronization

The preamble precedes the frame and allows the receiver's clock to synchronize itself with that of the transmitter (do not forget that the transmission mode is asynchronous). It is sent to stabilize the decoding circuits and, therefore, it is envisaged that part of the preamble may be lost. Thus, it is considered normal for the first bits (up to 18 bits) not to reach the layer above (MAC layer).

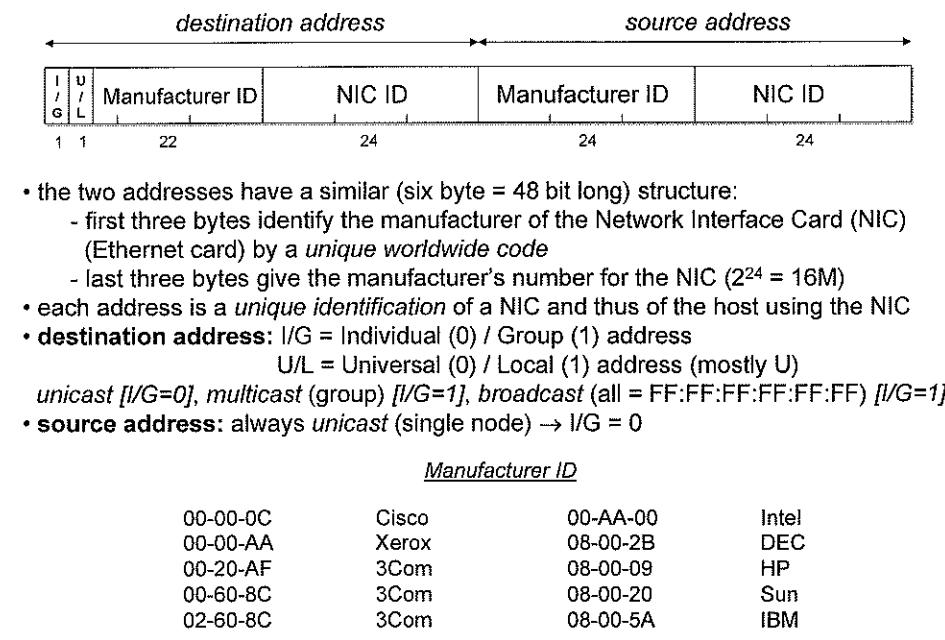
The preamble may be considered to be at the physical level, while the other fields of the frame emanate from the MAC layer. The preamble consists of a sequence of successive 1s and 0s (56 bits in length), and thus it does not contain any specific information. There are only transitions in the middle of a bit period.

Frame Synchronization

The Starting Frame Delimiter (SFD) is eight bits long and continues the preceding sequence, except that the very last bit is set to 1. This double 1 tells the receiver that the actual frame is about to start and that the subsequent bits therefore comprise significant fields of the frame. The double 1 introduces a transition at the start of a bit period.

If a collision is detected during the transmission of the preamble or SFD, the transmitting host should, nevertheless, continue to transmit the complete preamble.

2.2.2 MAC Address



The frame includes two addresses. The destination address is transmitted first, followed by the source address. The Ethernet addresses are represented conventionally in hexadecimal, with a hyphen (:) separating each of the six bytes. The two addresses have similar structures. They are six bytes long with, for IEEE 802.3, a possibility of using a two byte reduced format (which is very rare). The first three bytes identify the manufacturer of the Network Interface Card (NIC) and thus the host in a one-to-one fashion. The entries for these bytes are allocated to the manufacturers by a unique worldwide organization (in this case, the IEEE), which ensures the consistency of the system. The following three bytes give this manufacturer's number for the connector card ($256^3 = 16.78$ million possibilities). Thus, the whole forms a unique number for each NIC and, by the same token, for each host with an Ethernet card.

Ethernet also allows one to reach several targets simultaneously. There is a choice of transmitting to everyone (*broadcast*) or to a group of hosts (*multicast*). The broadcast destination address is FF-FF-FF-FF-FF-FF in hexadecimal (which corresponds to a sequence of 1s in binary). The group address has the property that the first bit transmitted is set to 1 (or the first byte is odd), while the other bits of the first three bytes are able to retain the number of the manufacturer whose hosts are targeted. Thus, a group may consist of hosts from the same manufacturer.

A source address cannot have a group bit set to 1. Thus, its first byte is even.

In the IEEE 802 structure, the first bit of an address is termed the Individual/Group (I/G) bit. However the second bit (6 byte address) transmitted also has a special meaning and indicates whether the address is of a universal type (as described above) or whether it is administered locally. In the latter case, the following 46 bits are chosen by the user and are not necessarily the manufacturer and connector Card IDs. This second bit is called the Universally/Locally (U/L) administered bit .

2.2.3 Type/Length Field

Ethernet: Type

Type specifies the upper-layer protocol carried by the frame.

IEEE 802.3: Length

Length indicates the number of data bytes (PAD and FCS not included).
The data field has a length between 46 to 1500 bytes = $(2E)_{16}$ to $(5DC)_{16}$

Protocol Types

0000-05DC	<i>length</i>	} <i>Ethernet</i>	} <i>IEEE 802.3</i>
0800	DoD IP		
0805	X.25 level 3		
0806	Ethernet ARP		
8035	RARP		
809B	AppleTalk		
80D5	IBM SNA		
814C	SNMP		

This two-byte field was defined in the Ethernet standard to indicate the type of level 3 protocol used to transport the message. This information could thus be used to switch the frame towards the software (driver) adapted for its decoding.

However, the meaning of this field changed with standardization (IEEE 802.3, then ISO 8802-3), and it now carries length information about the data field (the following field), according to the standard. This length is not indispensable to the operation of the receiver, since the start and end of the frame can be deduced from the end of the preamble (two consecutive bits set to 1) and the fall in the carrier in the last bit; moreover, the lengths of other fixed fields (addresses, type, FCS) are known. This information is also of interest both when the data field is not entirely full (a case in which it is desired to transmit only a few bytes using a short frame) and to verify the consistency between the total length of the frame received and the number of bytes as given by this field.

How can Ethernet frames be distinguished from IEEE 802.3 frames on the network? The two may coexist perfectly well, which is often the case in reality. The data field has a length of between 46 bytes (with insertion of a PAD, if necessary) and 1500 bytes, which makes 2E (or less, if there is a PAD) or 5DC in hexadecimal. Thus, if the content of the field following the two addresses is greater than 5DC, it is determined to be a type field and hence an Ethernet frame. Otherwise, it must be a length field and an IEEE 802.3 frame.

The most common protocol types are listed.

2.2.4 Data and PAD

Data

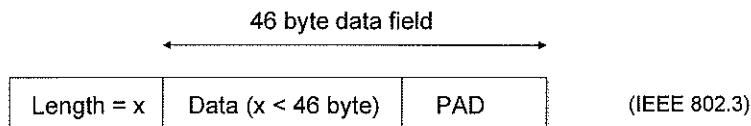
Contains a Logic Link Control (LLC = IEEE 802.3) packet

- *maximum length of 1500 bytes* (frame of 1518 bytes) to prevent the consumption of the network bandwidth by a single host and to reduce the Frame Error Rate
- *minimum length of 46 bytes* (frame of 64 bytes = 512 bits = 51.2 μ s @ 10 Mbps) to be able to detect collisions over the entire network

PAD

The PAD or stuffing sequence is only used to fill the data field to obtain a minimum length of 46 bytes (frame of 64 bytes).

It consists of a sequence of meaningless bits placed after the data itself. In IEEE802.3, the length field indicates the data length without the PAD.



Data

The data field contains the LLC-level or the level-3 packet, thus it has no intrinsic meaning as far as Ethernet is concerned (at the MAC level). The field is viewed as a sequence of 46 to 1500 bytes, incorporated in the frame, and no attempt will be made to interpret it. The only processing applied to the data will be the calculation of the CRC.

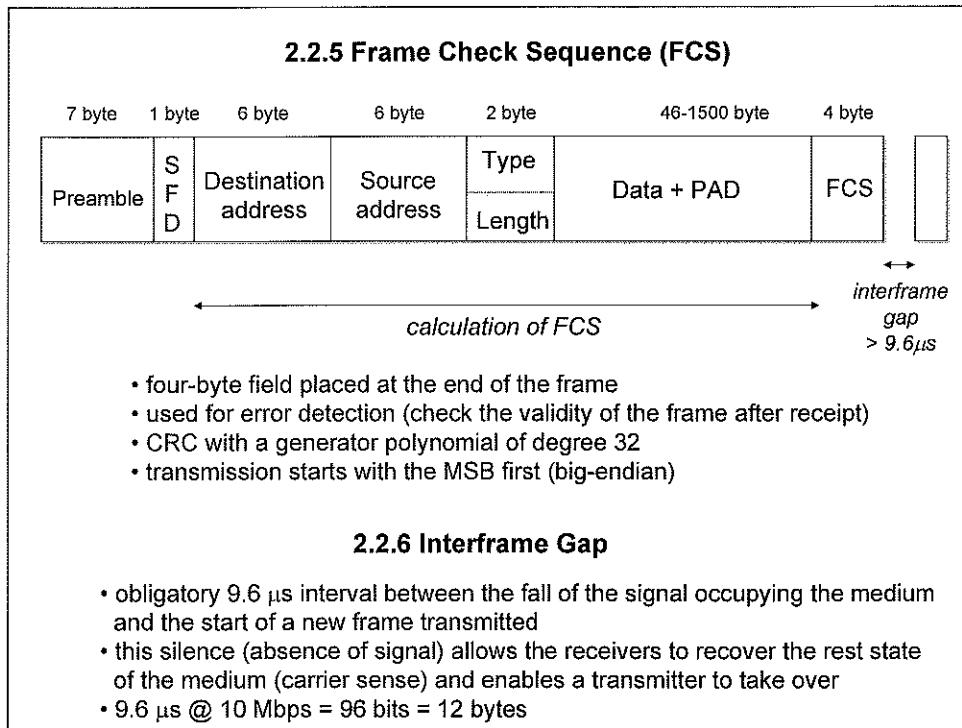
Within an ISO architecture, the IEEE 802.3 frame will encapsulate an IEEE 802.2 LLC packet.

If less than 46 bytes are provided by the layer above, the data field is filled out by the PAD.

PAD

The PAD or stuffing sequence is only used to fill the data field to obtain at least 46 bytes. It is therefore indispensable in completing the generation of a short frame from a message consisting of only a few bytes. It consists of a sequence of meaningless bits placed after the data itself.

In the case of *Ethernet*, the MAC level of the frame did not transport any information about the number of data bytes. The differentiation between the useful bytes and the stuffing therefore had to be provided by the upper layers. In *IEEE 802.3*, the length field indicates whether the data field contains a PAD and gives its length (obtained by subtraction).



FCS

Frame Check Sequence (FCS) is a four-byte field placed at the end of frames which is used to check the validity of the frame after receipt, up to a one bit. It uses a Cyclic Redundancy Check (CRC) calculated using a generator polynomial of degree 32. It covers the two address fields, the type/length field and the data (including PAD), and is thus used by the receiving host to decide whether the frame is perfectly correct and can be transmitted to the layer above (LLC or level 3).

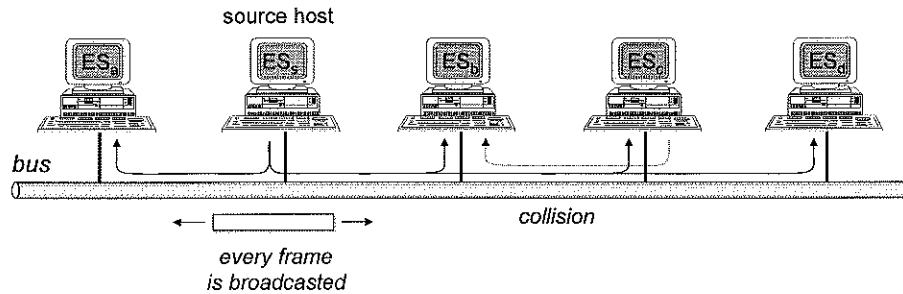
FCS is the only field of the frame to be transmitted beginning with the MSB.

Interframe Gap

A host cannot transmit all the frames it has to transmit one after the other. There is an obligatory 9.6 µs interval between the fall of the signal occupying the medium and the start of the frame transmitted; this is known as the interframe gap, or spacing. This silence allows the electronic circuits to recover the rest state of the medium (absence of signal), and it may enable other hosts wishing to transmit to take over at that time. It keeps any one transmitter from sending back-to-back frames so close together that is able to maintain continuous control of the network. Due to the Medium Access Control (MAC) method, all hosts can take their turn on a single network in which regular exchanges between other equipment take place, with a relatively small delay.

2.3 Medium Access Control (MAC): CSMA/CD

CSMA/CD = Carrier Sense Multiple Access with Collision Detection



Broadcast Bus: the transmitted frame is received by every host

Half Duplex: only one transmitter active at a time

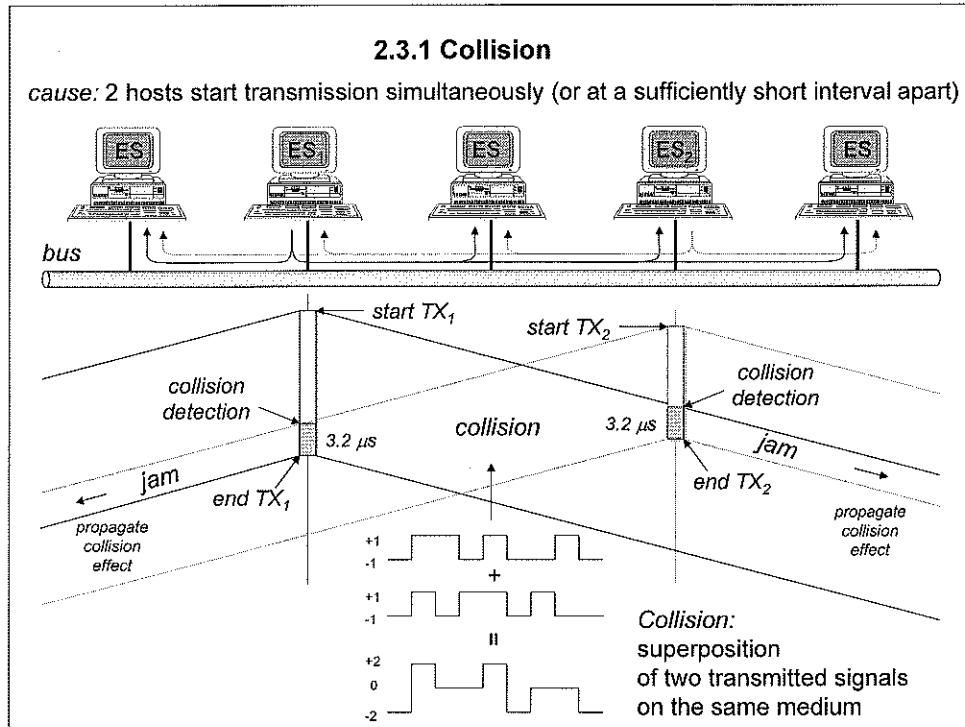
Multiple Access: a set of hosts send and receive frames over a shared medium (bus)

Time Division: one host transmits at a time

Random Access: no scheduled time for any host to transmit, first-come first-served

Collision: two hosts start transmission simultaneously

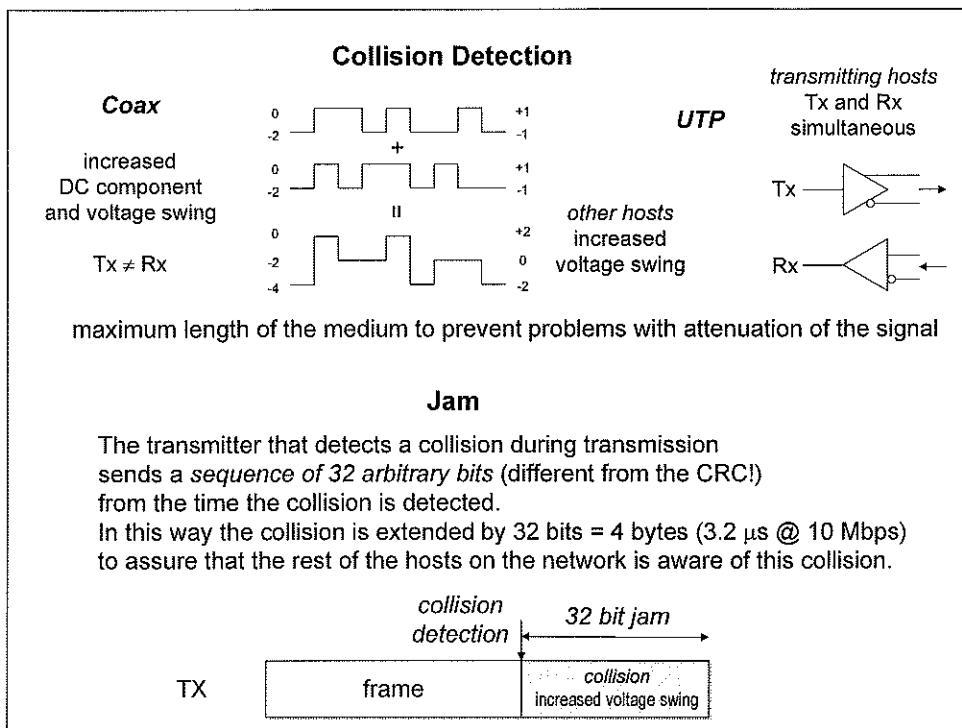
All frame transmissions between all the hosts that are attached to the LAN take place over a shared bus and the CSMA/CD method is then used to share the use of the bus in an equitable way.



The collision is the phenomenon resulting from the superposition of two signals on the medium. The collision occurs if the two transmitters started up simultaneously (or at a sufficiently short interval apart). This event is normally linked to and resolved by the access method. However, its occurrence should not exceed certain alarm thresholds (for example, one collision per 1000 frames) otherwise the traffic flow will be harmed. Suppose that two hosts wish to transmit a frame at the same time. They will sense the medium and, assuming that it is free at that time, will both decide that they can act and move simultaneously to transmit mode. What happens? Their signals are superimposed and the two frames become invisible, even for the two addressees.

In fact, it is not necessary for the hosts to start the transmission at exactly the same time. It is sufficient for the time separating the two transmissions to be less than the transmission time between their two points of attachment to the network. If this is the case, it was quite possible for the two hosts to find the medium empty at the time they sensed the carrier before transmission. However, a conflict will arise after a delay of less than the transit time separating them.

This event is acceptable if the transmitters are informed of it. Thus, they should be able to *detect* this event which prevents correct reading of the signal at any point on the medium. For this, it is decided to *extend the collision sufficiently*. In fact, the aim is to ensure that the collision does not go undetected. Otherwise the network would be incapable of detecting the problem which has arisen and would fail to recover the frame or pass the information on to the layer above.



Collision Detection

The coaxial transceiver (10 BASE 5/2) detects the collision either because it cannot read the bits it is transmitting correctly or because the continuous component of the signal on the cable is greater than that for normal signal transmission (- 1V).

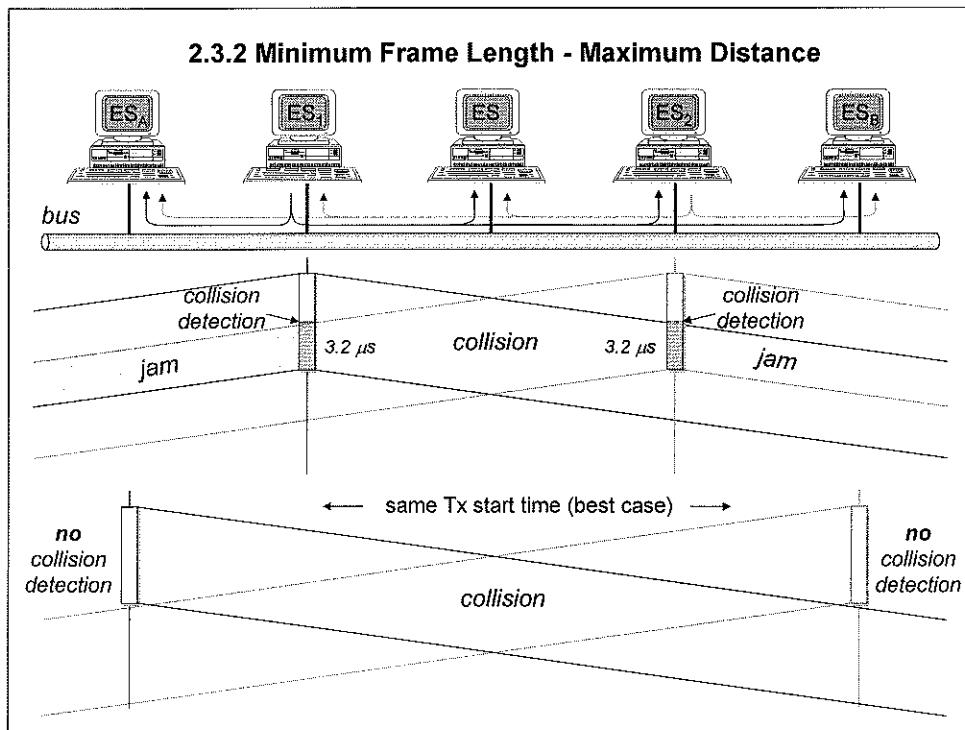
The UTP transceiver operates on a point-to-point link. There can be no superposition at the Tx host and the collision simply involves the detection of a reception during transmission. At the receiving hosts an increased voltage swing is detected.

Jam

The jam is a signal with no intrinsic meaning. It assures elements that have detected a collision on their transmissions that the rest of the network is aware of this collision. This is simply implemented by making the collision last at least 32 bit times, by transmitting a sequence of arbitrary bits from the time the transmitter detects a collision.

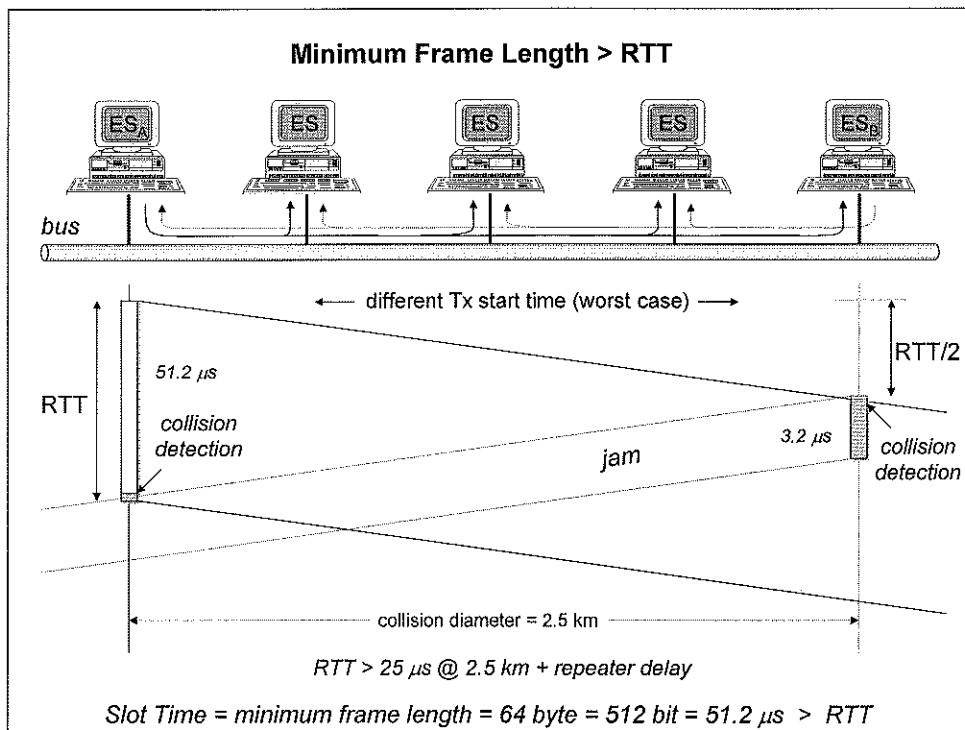
To avoid confusion, the value of these 32 bits should be different from the CRC, which would correspond to the start of the frame already transmitted. In reality, the sequence transmitted is always the same.

This signal is also called 'collision enforcement', which is evocative of its purpose.



A host can experience a collision not just at the start of a frame but after it has transmitted a number of bits. The worst-case time delay - and hence maximum number of bits that have been transmitted - before detecting that a collision has taken place is known as the *collision window* and occurs when the two colliding hosts are attached to opposite extremities of the bus, as is shown in the figure for hosts A and B. The occurrence of an *undetected collision* is illustrated. The collision is not visible for the hosts A and B. They see the two packets pass one after the other.

In the figure, host A has determined that no transmission is in progress and hence starts to transmit a frame. Irrespective of the bit rate being used, the first bit of the frame will take a small but finite time to propagate over the transmission medium determined by the length of the cable, d , and the signal propagation velocity, v . The maximum length of the cable is set at 2.5 km (*collision diameter*). Assuming a velocity v of 2×10^8 m/s = 1 m / 5 ns, the worst-case signal propagation delay time, T_p , going from one end of the cable to the other, is given by: $T_p = d/v = 12.5 \mu s$. Just prior to the first bit of the frame arriving at its interface, host B determines the transmission medium is free and also starts to transmit a frame. After B has transmitted just a few bits, the two signals collide and the collision signal then continues to propagate back to host A. Hence the worst-case time before host A detects that a collision has occurred, $2T_p$, is $25 \mu s$. In order to transmit the signal over this length of cable; the cable is made up of five 500 m segments, all interconnected together by means of four devices called *repeaters*. Each repeater introduces a delay of a few microseconds in order to synchronize to each new frame. Hence the total worst-case time is set at $50 \mu s$ or, assuming a bit rate of 10 Mbps, after A has transmitted $10 \text{ Mbps} \times 50 \mu s = 500$ bits. A safety margin of 12 bits is then added to this and the minimum frame size is set at 512 bits or 64 bytes/octets. This is called the *slot time* (in bits) and ensures that host A will have detected a collision before it has transmitted its smallest frame.

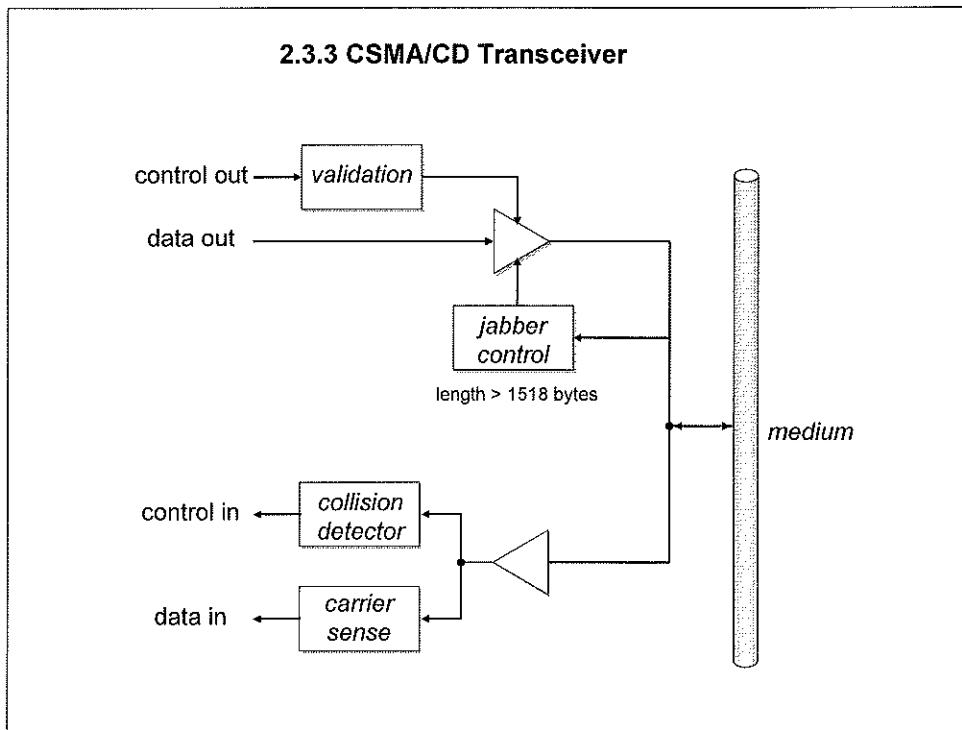


There is a way of avoiding this type of event, which is difficult to manage. The time taken to transmit a frame must always be greater than twice the signal propagation delay time T_p between the two points of transmission. This is called the Round Trip Time RTT ($2.T_p$). Since the size of an Ethernet network is bounded (the number of segments, transceivers and repeaters is limited, as is the maximum length of each segment), the RTT for an Ethernet network should not exceed a fixed value (499 bit times), and the corresponding minimum frame should consist of at least 512 bits (or 64 bytes, excluding preamble and SFD).

To ensure that the collision persists for sufficient time for it to be detected by host A, on detecting the collision, B continues to send the jam sequence, a random bit pattern of 32 bits.

With this constraint, all collisions last long enough to ensure that they are propagated to the transmitters concerned, wherever these are located, which, when informed that the last frame was lost, may then attempt to retransmit from their buffer, according to a retransmission algorithm (describe later).

Remark: An elementary way of provoking collisions involves removing the impedance adaptation plug or terminator at the coax end. Then any frame transmitted in the circuit is reflected on the open circuit and collides with itself. This exemplifies the attention which must be paid to the cabling and to its protection, since an untimely disconnection of this 50 Ohm plug may block a network completely.



Jabber Control

The jabber is a frame which is too long, that is, with a length greater than 1518 bytes. Theoretically, this type of fault should never occur in a healthy network. However, the original causes of instances in which the frame contains between 1500 and 3000 bytes or several tens of thousands of bytes vary.

In the first case, it is a superposition of two long frames involved in an undetected collision. The fact that a collision has not been detected reveals an important problem. If the CD (collision detection) is no longer guaranteed, the Ethernet layers 1 and 2 can no longer guarantee the same quality of service.

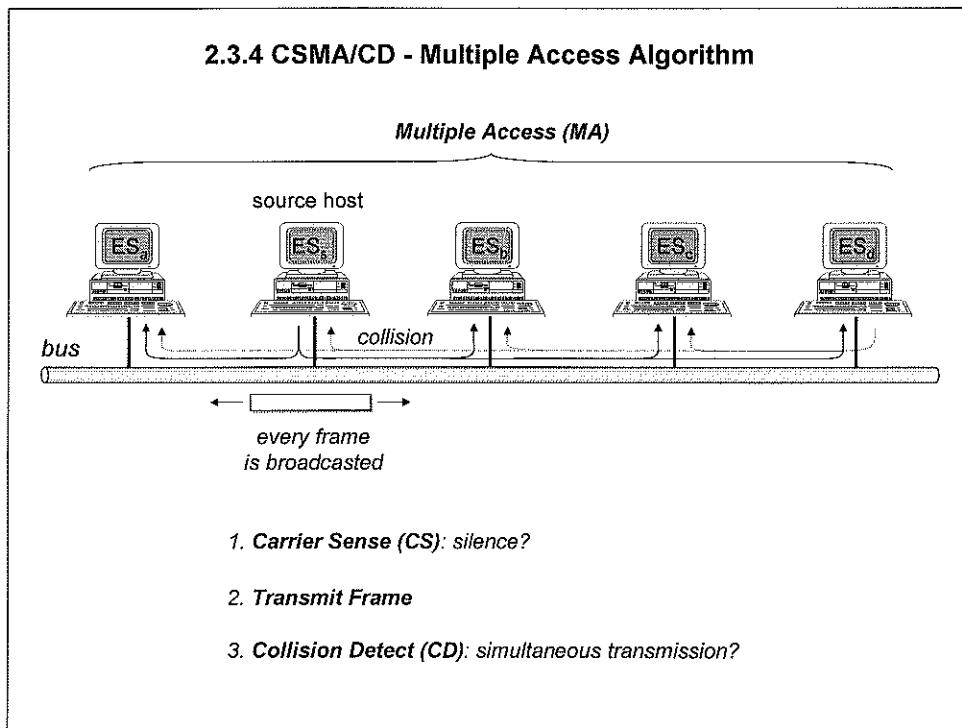
In the second case, the packet probably does not have a frame structure and must be produced by a defective component which has remained in transmission mode for far too long. There will be a tendency to accuse low-level elements (transceivers, repeaters, connector cards) rather than the software driver or the application used. This is because the problem appears rudimentary and directly linked to the production of the physical signals. This fault should be located and repaired quickly since it may be very injurious for the network. It is clear that an element which takes over to transmit for a long time blocks the network totally and unnecessarily for this time. It inhibits or corrupts all other transmissions. The jabber control isolates the transmission data path from the cable if certain defined time limits are violated. All frames transmitted on the medium have a defined maximum length. If this is exceeded, the jabber control inhibits further output data from reaching the medium.

Carrier Sense

Signals activity (transmission of frames) on the medium.

Collision Detect

Signals a collision of the transmitted frame with a frame transmitted by an other host.

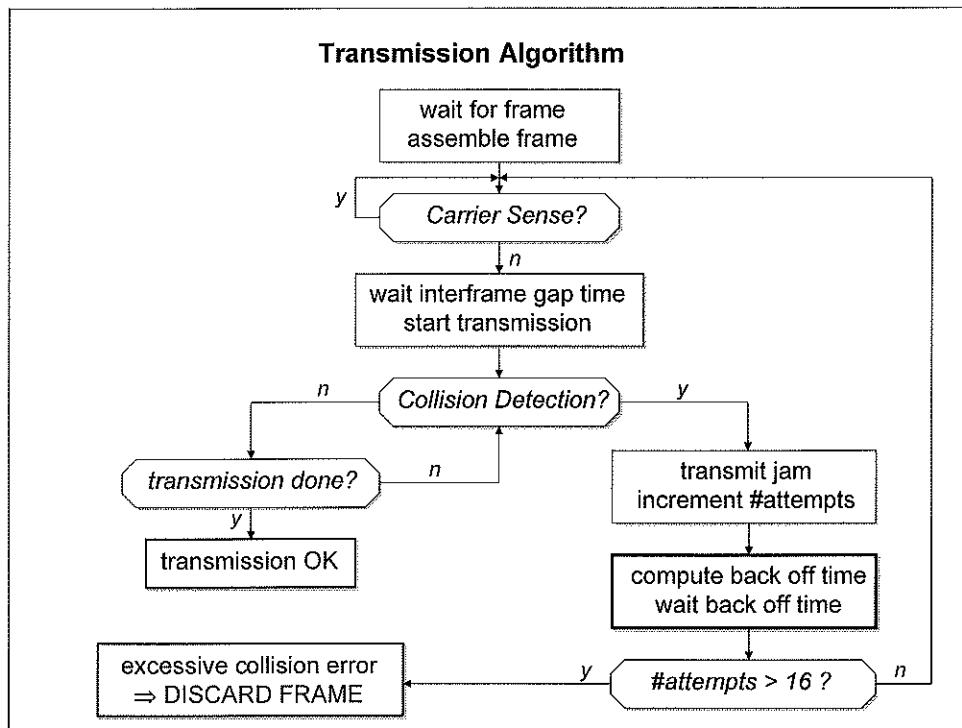


The CSMA/CD method is a particular version of the ALOHA type method, in which every transmitter is free to manage its transmissions as a function of its needs and the availability of the medium (random access).

When there is no traffic to transmit, the host remains silent and listens to the activity on the medium. (A host cannot determine the direction of the traffic.)

When the host needs to transmit one or more packets, it will act independently of the others. It only knows the activity on the medium. Since each host has the possibility of beginning a transmission autonomously at any time, the access method is distributed and is said to be a *Multiple Access (MA)* method. Thus, the host observes the medium in an attempt to detect a carrier (*carrier sense*, CS). If no frames are in transit (no carrier), the medium is free. The host then starts to transmit. It continues to listen for the result of its transmission for some time, in order to check that no other host has behaved in the same way as itself at the same time.

In fact, two different hosts may start to talk simultaneously, after each has checked immediately beforehand that no-one is transmitting. In this case, the signals interfere and are lost to everyone. With *collision detection* (CD), a host is able to detect a contention problem at the time it transmits and to stop with the intention of resending its packet later when it again has the right to talk. To minimize the risk of encountering a second collision with the same host, each waits for a *random delay* period before attempting to transmit again. This reduces the probability of successive collisions between a pair of hosts. However, so as not to saturate a network which is already heavily loaded, the host will not attempt indefinitely to retransmit a packet if, on every attempt, it finds itself in collision with another. After a certain number of fruitless attempts, the packet is deleted, which means that it does not cause the network to collapse (by not overloading it further); this action tells the higher layers that there is a problem, since the exchange was perturbed by the loss of a message.

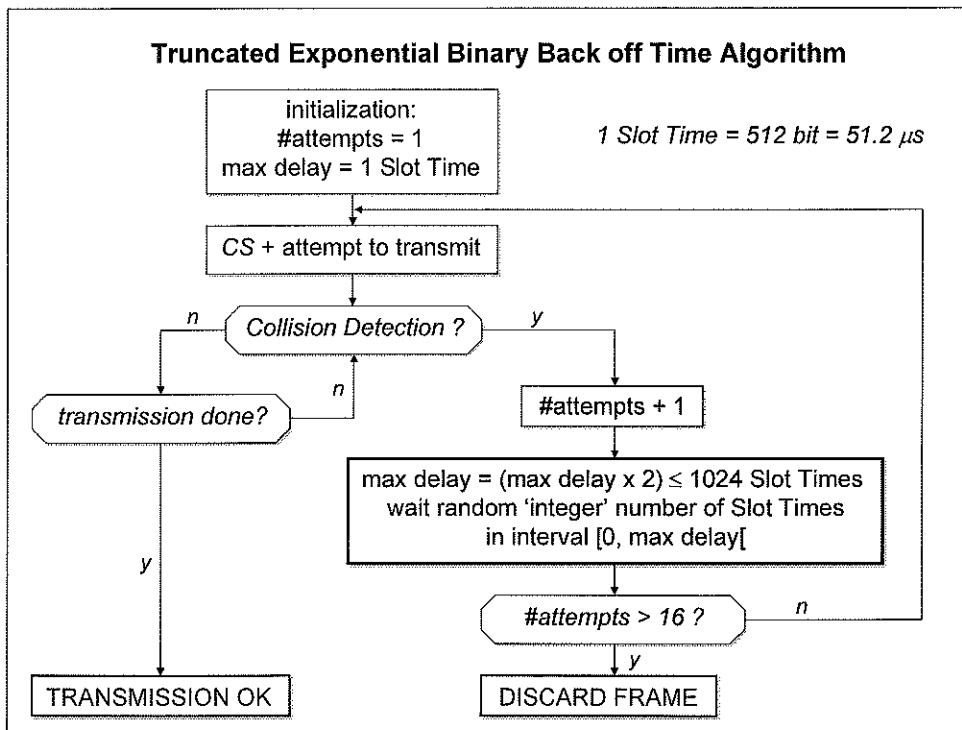


The access method used for Ethernet is of the persistent CSMA/CD type.

As the bit stream is transmitted, the transceiver simultaneously monitors the received signal to detect whether a collision has occurred. Assuming a collision has not been detected, the complete frame is transmitted and, after the FCS field has been sent, the MAC unit awaits the arrival of a new frame, either from the medium or from the controlling microprocessor. If a collision is detected, the transceiver immediately turns on the collision detect signal. This, in turn, is detected by the MAC unit which enforces the collision by transmitting the jam sequence to ensure that the collision is detected by all other hosts involved in the collision. After the jam sequence has been sent, the MAC unit terminates the transmission of the frame and schedules a retransmission attempt after a short randomly selected time interval.

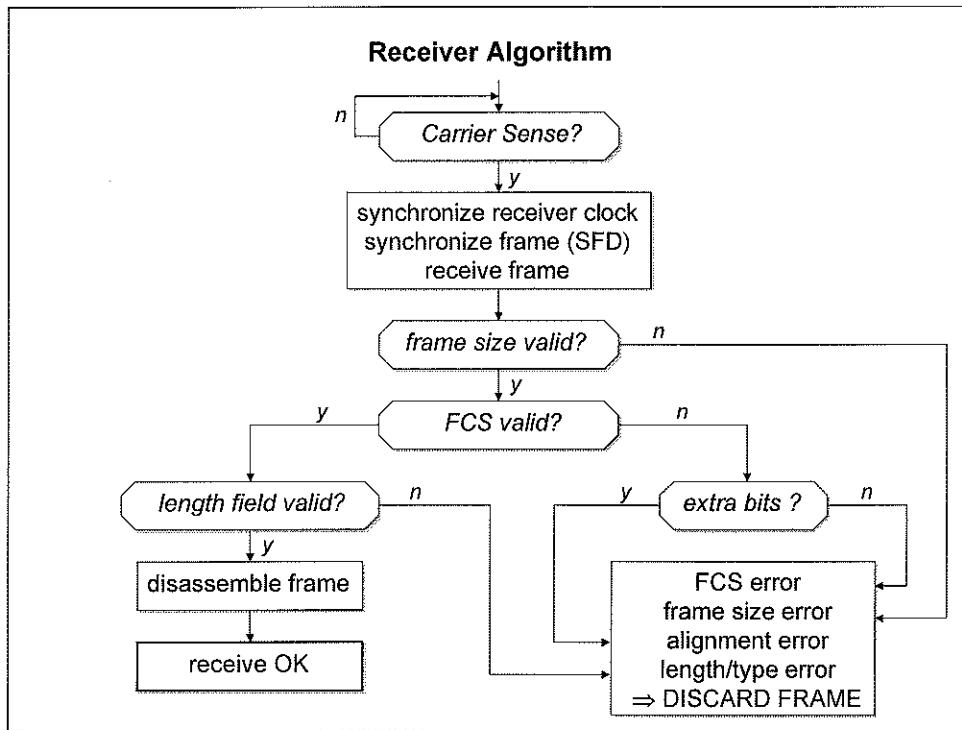
In the event of a collision, retransmission of the frame is attempted up to a defined maximum number of tries known as the attempt limit. Since repeated collisions indicate a busy medium, the MAC unit tries to adjust to the medium load by progressively increasing the time delay between repeated retransmission attempts (using a random distribution on an increasing time interval), to prevent overload of the medium by retransmissions. The scheduling of retransmissions is controlled by a process called *truncated binary exponential backoff*. When transmission of the jam sequence is complete, and assuming the attempt limit has not been reached, the MAC backs off (delays) a random integral number of Slot Times before attempting to retransmit the affected frame. A given host can experience a collision during the initial part of its transmission, the collision window, which is effectively twice the time interval for the first bit of the preamble to propagate to all parts of the medium (RTT). The *Slot Time* is thus the worst-case time delay a host must wait before it can reliably know a collision has occurred. It is defined as:

$$\text{Slot Time} = 2 \times (\text{transmission path delay}) + \text{safety margin}$$



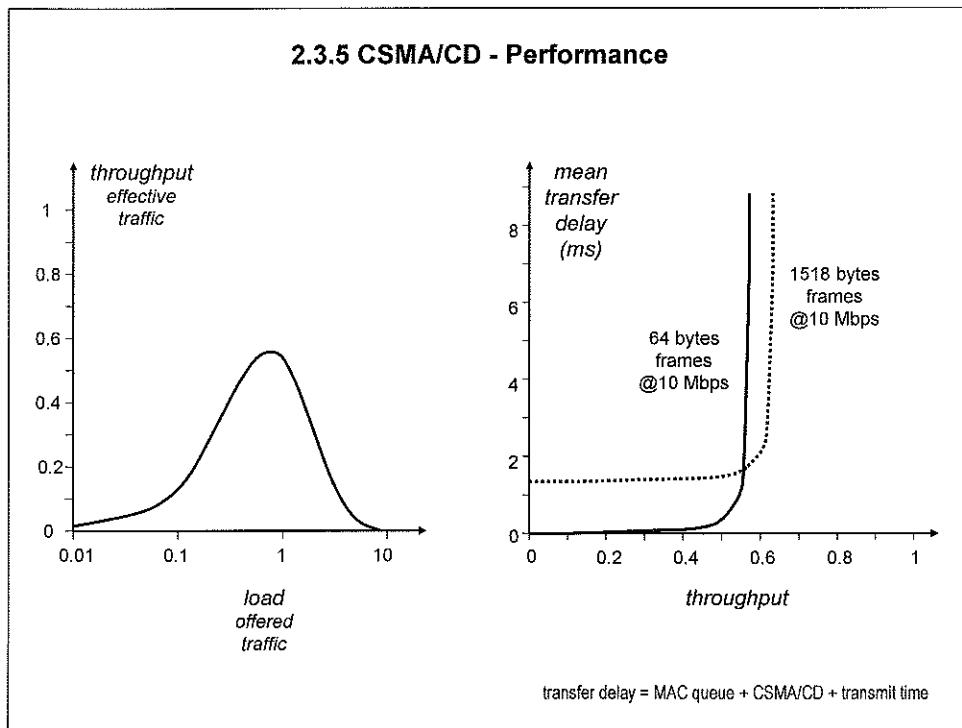
The *transmission path delay* is the worst-case signal propagation delay going from any transmitter to any receiver on the cable network. This includes any delays experienced in repeaters. The Slot Time is double this delay (to allow for the corrupted signal to propagate back to the transmitting host) plus a safety margin. The Slot Time used is made equal to this figure rounded up to be a multiple number of octets at the bit rate used. For example, for a 10 Mbps baseband coaxial cable network with a maximum of 2.5 km between any transmitter and any receiver, the Slot Time is equal to 512 bit times or 64 octets. The number of Slot Times before the Nth retransmission attempt, is then chosen as a uniformly distributed random integer R in the range $0 \leq R < 2^K$, where $K = \min(N, \text{backoff limit}=10)$.

When the transmission of a frame is perturbed by a collision, the host produces a jam, then ceases all transmissions until the medium becomes free again. After an interval, randomly determined so as not to restart simultaneously with its 'rival', it again tries to send its message and loops on the same transmission procedure as before. If a serious problem prevents all transmission, the Ethernet MAC level ensures that the host stops its series of attempts to transmit by simply deleting the frame to be transmitted from its buffers after 16 unsuccessful attempts. This shows the unreliability of the network, since, in certain cases, it authorizes the loss of a packet without informing the layers above. Moreover, if the collisions are due to the network load, the more the hosts try to retransmit their data rapidly, the more they load the network, resulting in an avalanche effect which tends towards a complete network blockage. To counter this, Ethernet uses a waiting time randomly chosen in an increasing window of possible delays. Thus, after the first collision, there are two possible choices for the window (0 or 1 Slot Times), which increases by a factor of two with every new attempt. Finally, after the tenth attempt, this window is bounded by 1024 Slot Times, which is already a relatively high value ($1024 \times 512 \times 0.1 \mu\text{s} = 52.4 \text{ ms}$). The attempts then continue, if necessary, up to the sixteenth time, with this same delay window.



On receipt a check is made to verify that the bit sequence can form a correct frame. Several criteria are used for this. The total number of bits should not be too small, otherwise it would be impossible to reconstitute all the fields of a complete frame. The destination address should be the physical address of the host, or should relate to a group of which the host is a member. The frame should not be too long, otherwise it would infringe a basic rule of Ethernet and could not be valid. The calculation of the parity control for the fields received should give a result identical to the value transported by the frame, otherwise there is a bit error (wrongly transmitted or wrongly received). The total number of bits received should be divisible by eight, so as to give an integral number of bytes, otherwise the reconstituted frame is invalid, since at least one of its fields has one or more bits too many (too few).

IEEE 802.3 introduced a length field into the frame by replacing the type field. This explains the presence of the additional test shown in the algorithm. In fact, this field provides additional information about the consistency of the frame received.



throughput

Load (offered traffic) and throughput (effective traffic) are normalized. They are expressed as a fraction of the available bit rate on the medium (10 Mbps).

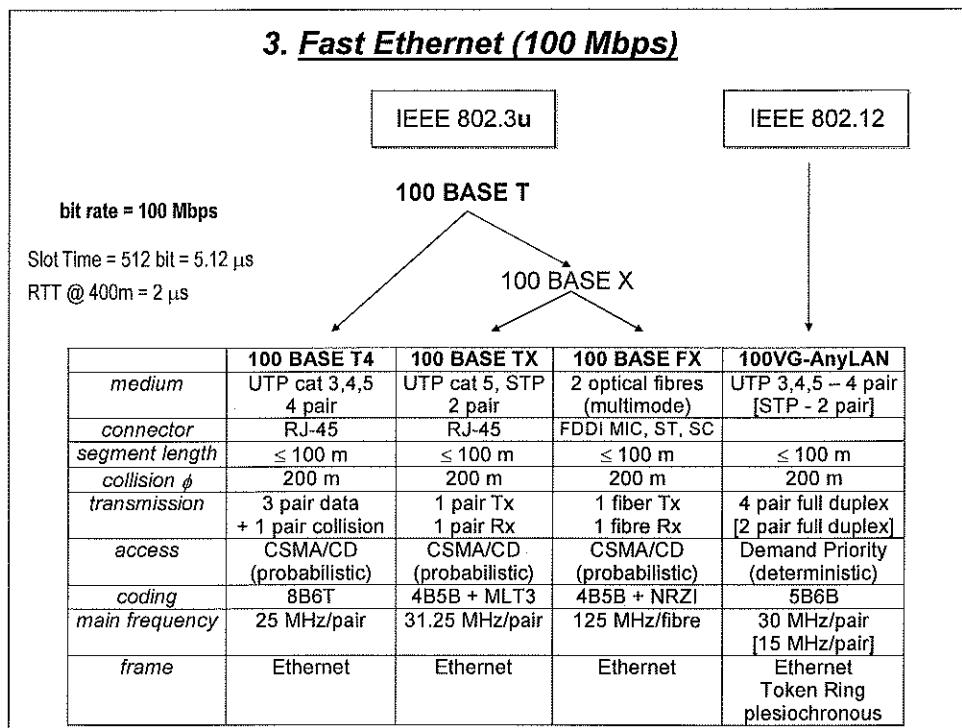
For small loads, the throughput increases proportional with the load. The number of collisions increases with the load. This overhead results in a 'less than proportional' increase of the throughput. If the number of packets to be transmitted (load) is actually too large ($\text{load} \approx 1$), an increase in the load even decreases the throughput. This is an unstable situation. Retransmissions increase further the load and reduces the throughput even more. A state of complete blockage of the network, with a drastic fall in throughput, might be reached (network collapse).

mean transfer delay

The transfer delay is defined as the time a frame is waiting in the MAC sublayer input queue + the delay associated with CSMA/CD + the time to transmit the frame.

The mean throughput is higher with a larger frame size (1518 byte frames vs. 64 byte frames). A larger number of frames will be generated with a smaller frame size (to transmit the same information). Hence the probability of a collision occurring at a particular throughput level is higher. Also the overhead associated with the recovery procedure increases.

The CSMA/CD method shows a tendency to slow down all transmissions on a heavily loaded network. It does not guarantee a minimum waiting time for the transmission of a frame (this is restrictive for real-time applications). The transmission of a frame is delayed because the medium is busy at the time the medium tries to send it, or because a collision occurred and a retransmission is needed. The more the traffic on the network increases, the more the mean transfer delay increases (exponential). The delay becomes unacceptable when the network is about to collapse.

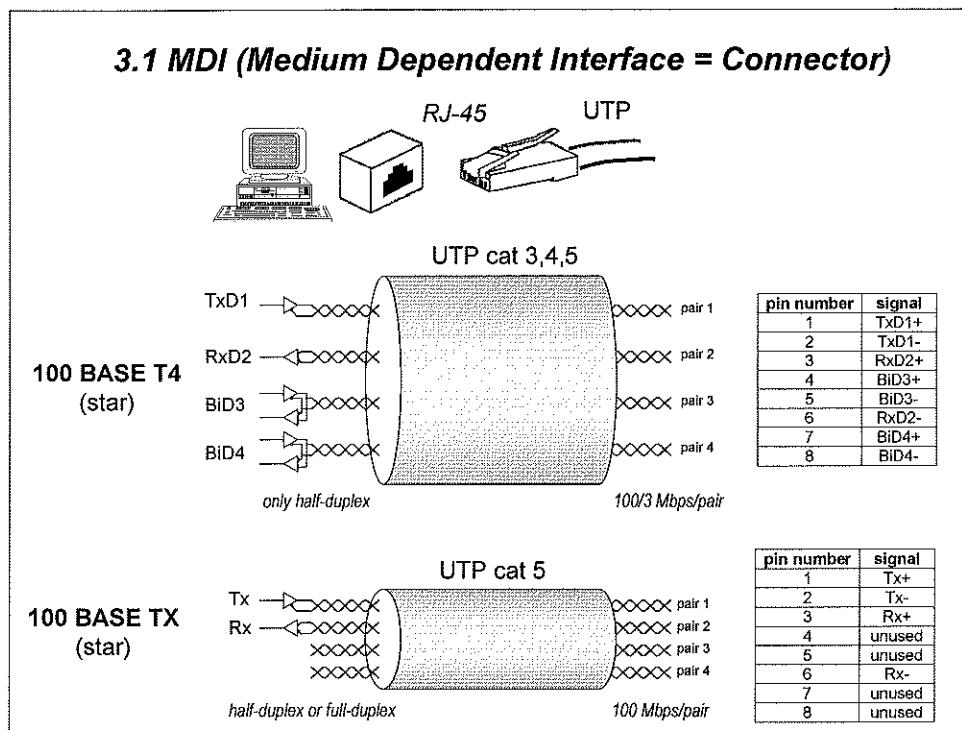


The aim of Fast Ethernet was to use the same shared, half-duplex transmission mode as Ethernet but to obtain a $\times 10$ increase in operational bit rate over 10 BASE T while at the same time retaining the same wiring systems, MAC method, and frame format. When using hubs with unshielded twisted-pair (UTP) cable, the maximum length of drop cable from the hub to a host is limited to 100 m by the driver/receiver electronics (signal attenuation and adaptive crosstalk canceller to overcome NEXT). Assuming just a single hub, this means that the maximum distance between any two hosts (the collision diameter) is 200 m and the worst-case path length for collision detection purposes is 400 m (with a velocity $v = 1 \text{ m} / 5 \text{ ns}$ this gives a RTT of 2 μs) plus the repeater delay in the hub. Therefore, a higher bit rate can be used while still retaining the same CSMA/CD MAC method and minimum frame size of 512 bits = 5.12 μs . The bit rate is set at 100 Mbps over existing UTP cable. Hence the standard is also known as **100 BASE T**.

The **100 BASE T4** is compatible with category 3 UTP cable used in 10 BASE T.

In addition to the 100 BASE T4 standard, a second Fast Ethernet standard is available, which is known as **100 BASE X**. Unlike 100 BASE T4 which was designed for use with existing category 3 UTP cable, 100 BASE X was designed for use with the higher quality category 5 cable (100 BASE TX) now being used in most new installations. In addition, it is intended for use with STP (Shielded Twisted Pair) and optical fiber cables (100 BASE FX). The use of various types of transmission media is the origin of the 'X' in the name. The 100 BASE X standard is adopted from the physical media standard ANSI X3T9.5 for the Fiber Distributed Data Interface (FDDI).

The **100VG-AnyLAN** (from HP) uses the medium access control mechanism 'demand priority' and is standardized under a new number: IEEE 802.12.



100 BASE T4

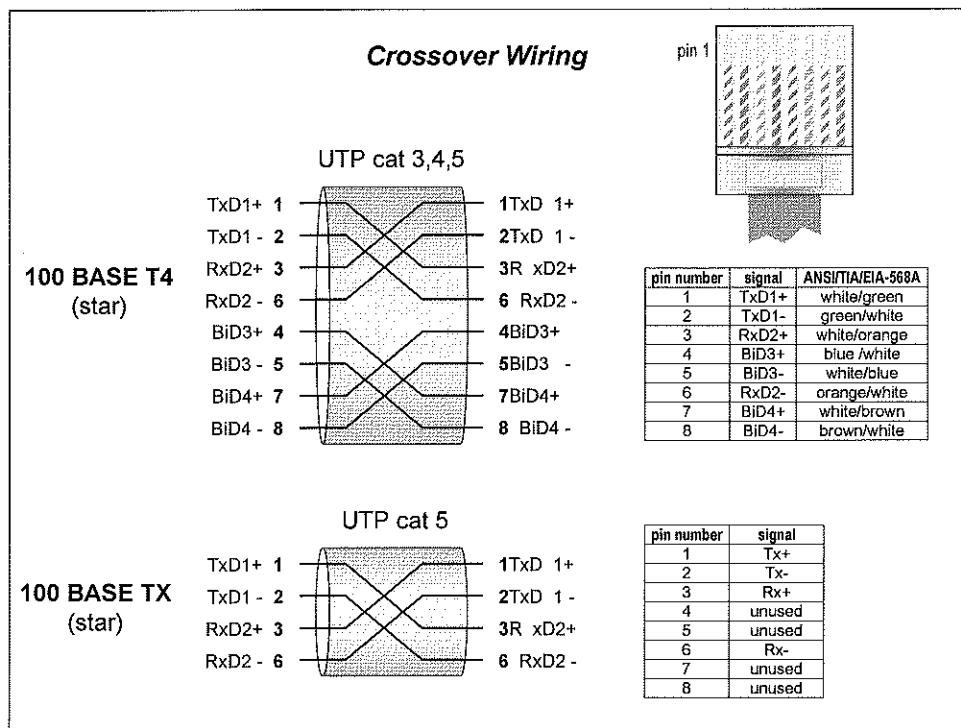
The major technological hurdle to overcome with Fast Ethernet was how to achieve a bit rate of 100 Mbps over 100 m of UTP cable. Category 3 UTP cable - as used for telephony, and the most widely installed - contains four separate twisted-pair wires. To reduce the bit rate used on each pair, all four pairs are used to achieve the required bit rate of 100 Mbps in each direction.

With the CSMA/CD access control method, in the absence of contention for the medium, all transmissions are *half-duplex*, that is, either host-to-hub or hub-to-host. In a 10 BASE T installation, just two of the four wire pairs are used for data transfers, one in each direction. Collisions are detected when the transmitting host (or hub) detects a signal on the receive pair (Rx+/-, pin 3&6) while it is transmitting on the transmit pair (Tx+/-, pin 1&2). Since the collision detect function must also be performed in the standard 100 BASE T4, the same two pairs are used for this function. The remaining two pairs are operated in a bidirectional mode.

The figure shows that data transfers in each direction utilize three pairs - pairs 1, 3, and 4 for transmissions between a host and the hub and pairs 2, 3, and 4 for transmissions between the hub and a host. Transmissions on pairs 1 and 2 are then used for collision detection and carrier sense purposes as with 10 BASE T. This means that the bit rate on each pair of wires need only be $100/3 = 33.33$ Mbps.

100 BASE TX

The 100 BASE TX makes use of two pairs of category 5 UTP cable, one pair used for transmission and one for reception. This means that the bit rate on each pair of wires must be 100 Mbps.



100 BASE T4

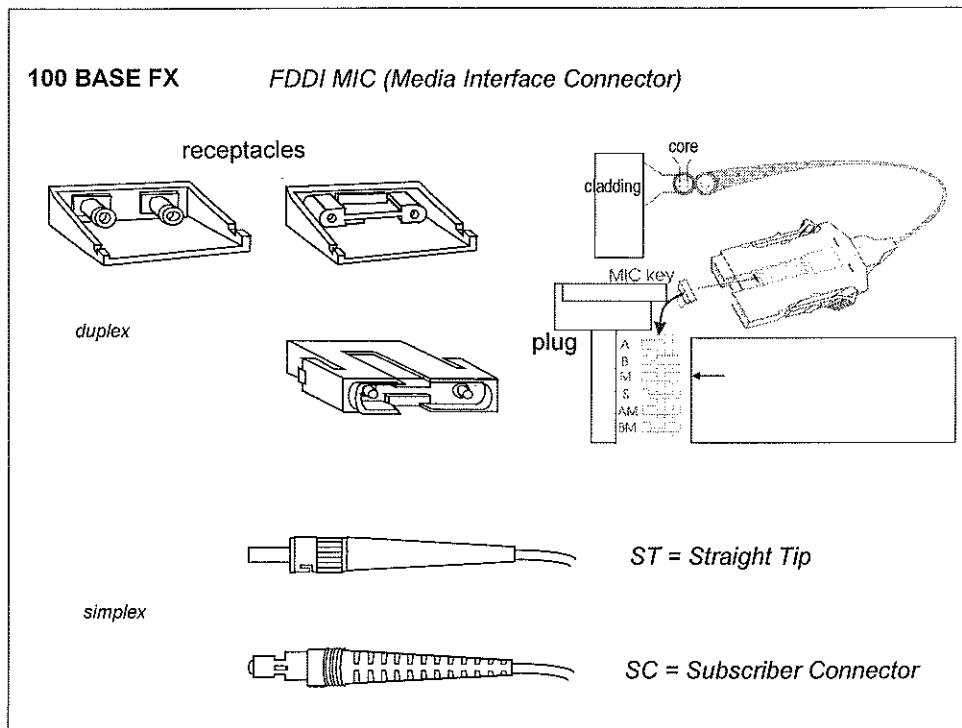
A signal crossover is required between hosts connected to a 100 BASE T4 segment, so that the Tx data pins and bidirectional data pins on the host at one end of the segment are connected to the Rx data pins and bidirectional data pins on the host at the other end, and vice versa. The standard recommends that the crossover be done internally in the repeater port. If the crossover function is done inside a repeater, then the port on the repeater must be marked with an 'X'.

If two hosts are linked together with a single 100 BASE T4 segment, then a special crossover cable must be provided. As shown in the figure, a crossover cable for a 100 BASE T4 segment must be wired so that the Tx pair at one end of the cable is connected to the Rx pair at the other end of the cable, and vice versa. In addition, the BiD3 pair at one end of the cable is connected to the BiD4 pair at the other end, and vice versa.

100 BASE TX

When connecting two hosts together over a segment, the Tx data pins of one MDI must be wired to the Rx data pins of the other MDI, and vice versa. For a single segment connecting only two hosts, this can be done by building a special crossover cable, with the transmit pins on the eight-pin plug at one end of the cable wired to the receive data pins on the eight-pin plug at the other end of the crossover cable.

When wiring multiple segments in a building, it's much easier to wire the cable connectors 'straight through' and not worry about whether the wires in the jumper cables or other twisted-pair cables in the building have been correctly crossed over. The way to accomplish this is to do the crossover wiring inside the repeater hub. The 100 BASE TX standard recommends this approach, and states that each port of the hub that is crossed over internally should be marked with an 'X'.



100 BASE FX

The MDI (Medium Dependent Interface) for a 100 BASE FX link may be one of three kinds of fiber optic connectors.

- **FDDI MIC (Media Interface Connector)**

This is a standard duplex connector used in FDDI LAN systems. MIC connectors are keyed in various ways, referred to as A, B, M and S, to make sure that the FDDI cabling is connected properly. If a FDDI MIC is used as a 100 BASE FX MDI, the specification state that it shall be keyed as an M receptacle. FDDI MIC connectors are just pushed into place and automatically complete the connection.

- **SC = Subscriber Connector**

This simplex connector is the recommended alternative in the standard. It is designed for ease of use. The SC connector is just pushed into place and automatically completes the connection. Two connectors are needed for bidirectional communication.

- **ST = Straight Tip**

This simplex connector is a spring-loaded bayonet-type connector that has a key on an inner sleeve of the ST plug with a corresponding slot on the ST receptacle. Then push the connector in and lock it in place by twisting the outer bayonet ring.

3.2 100 BASE T4

Line Code: 8B6T → $100/3 \times 6/8 = 25$ Mbaud/pair

Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword	Data byte	Codeword
00	- + 0 0 - +	20	- + + - 0 0	40	- 0 0 + 0 +	60	0 + + 0 - 0	80	- 0 0 + - +	A0	- + + 0 - 0	C0	- + 0 + - +	E0	- + + 0 - +				
01	0 - + - 0 -	21	+ 0 0 - + -	41	0 - 0 0 + +	61	0 + + 0 - 0	81	0 - 0 - + +	A1	+ - + - 0 0	C1	0 - + - + +	E1	+ - + 0 + +				
02	0 - + 0 - +	22	- + 0 - + +	42	0 - 0 - 0 +	62	0 + 0 + 0 - 0	82	0 - 0 + + +	A2	+ - + + 0 0	C2	0 - + + + +	E2	+ - + 0 + +				
03	0 - + + 0 -	23	+ - 0 - + +	43	0 - 0 + + 0	63	+ 0 + 0 + 0 -	83	0 0 + + +	A3	+ - + + 0 0	C3	0 + + + + +	E3	+ + + 0 + +				
04	- + 0 + 0 -	24	+ - 0 + 0 0	44	- 0 0 + + 0	64	0 + + 0 0 -	84	- 0 0 + + -	A4	- + + 0 0 -	C4	- + 0 + + +	E4	+ + + 0 + 0				
05	+ 0 - - + 0	25	- + 0 + 0 +	45	0 0 - + +	65	+ + 0 - 0 0	85	0 0 - - + +	A5	+ - + - 0 0	C5	+ 0 - + + +	E5	+ + - 0 + 0				
06	+ 0 - 0 - +	26	+ 0 0 - 0 0	46	0 0 - - + 0	66	+ + 0 0 - 0	86	0 0 - - + +	A6	+ - + - 0 0	C6	+ 0 - + + +	E6	+ + - 0 + +				
07	+ 0 - + 0 -	27	- + + - + -	47	0 0 - + + 0	67	+ + 0 0 0 -	87	0 0 - + + +	A7	+ - + + 0 0	C7	+ 0 + + + +	E7	+ + + 0 + -				
08	- + 0 + 0 - +	28	0 + + - 0 -	48	0 0 + 0 + 0	68	0 + + - + -	88	- 0 0 + 0 + 0	A8	- + + + - +	C8	+ 0 + + + 0	E8	+ + + 0 + +				
09	0 - + + + 0	29	+ 0 + 0 - -	49	+ + - 0 0 0	69	0 + + + - -	89	- 0 + 0 + 0	A9	- + + + + -	C9	0 - + + + 0	E9	+ + + 0 + -				
0A	0 - + 0 + - 0	2A	+ 0 + - 0 -	4A	+ - + 0 0 0	6A	+ 0 + + - -	9A	0 - 0 + 0 +	A10	+ + + + + +	C10	0 - + 0 + 0	E10	+ + + 0 + -				
0B	0 - + + - 0 +	2B	+ 0 + + - 0 -	4B	- + + 0 0 0	6B	0 + + + - +	9B	0 0 + 0 + 0	A11	+ + + + + +	C11	0 + + 0 + 0	E11	+ + + 0 + +				
0C	- + 0 + + - 0	2C	0 + + + - 0 -	4C	0 + + 0 0 0	6C	0 + + + - -	9C	- 0 0 + 0 + 0	A12	- + + + - +	C12	- 0 0 + 0 + 0	E12	- + + 0 + +				
0D	+ 0 - + + - 0	2D	+ 0 + 0 - 0 -	4D	+ 0 - 0 0 0	6D	+ + 0 + + -	9D	0 0 - 0 + 0	A13	- + + + + -	C13	- 0 0 + 0 + 0	E13	- + + 0 + +				
0E	+ 0 + 0 - + -	2E	+ 0 + 0 - 0 -	4E	0 - + 0 0 0	6E	+ + 0 + - -	9E	0 0 - 0 + 0	A14	- + + + + -	C14	- 0 0 + 0 + 0	E14	- + + 0 + +				
0F	+ 0 - 0 + + -	2F	+ 0 + 0 - 0 -	4F	- 0 + 0 0 0	6F	+ + 0 - + -	9F	0 0 - 0 + 0	A15	- + + + + -	C15	- 0 0 + 0 + 0	E15	- + + 0 + +				
10	0 - - + 0 +	30	- + 0 0 + +	50	+ - + 0 + 0	70	0 0 0 + + +	90	- + + + + +	B0	0 0 + 0 + 0	D0	0 + + + + +	F0	1 + 0 0 - +				
11	- 0 - 0 + 0 +	31	0 + - 0 + 0	51	- + - 0 + +	71	0 0 0 + + +	91	- + + + + +	B1	0 + 0 0 + 0	D1	0 + + + + +	F1	0 + 0 + 0 +				
12	- 0 - 0 + 0 +	32	0 + - 0 + +	52	- + - 0 + +	72	0 0 0 + + +	92	- + + + + +	B2	0 + 0 0 + 0	D2	0 + + + + +	F2	0 + 0 + 0 +				
13	0 - + + 0 +	33	0 + + - 0 +	53	- + + 0 + 0	73	0 0 + 0 + 0	93	- + + + + +	B3	0 + 0 0 + 0	D3	0 + + + + +	F3	0 + 0 + 0 +				
14	0 - - + + 0	34	- + 0 + + 0 -	54	+ - + + 0 +	74	0 0 0 + 0 +	94	- + + + + +	B4	+ 0 0 0 + 0	D4	+ 0 + + + +	F4	1 + 0 + 0 +				
15	- - 0 0 + +	35	- 0 + + - 0 +	55	- + + 0 + +	75	0 0 0 + + 0	95	- + + + + +	B5	0 0 + + 0 + 0	D5	- 0 + + + + +	F5	0 + 0 + + 0 +				
16	- - 0 + 0 +	36	- 0 + + + - 0	56	- + + 0 + +	76	0 0 0 - 0 + +	96	- + + + + +	B6	0 0 + 0 + 0	D6	- 0 + + + + +	F6	0 + 0 + + 0 +				
17	- - 0 + + 0 +	37	- 0 + 0 + +	57	- + + 0 + +	77	0 0 0 + 0 + +	97	- + + + + +	B7	0 0 + 0 + 0	D7	- 0 + + + + +	F7	0 + 0 + + 0 +				
18	- - 0 + + 0 +	38	- 0 + 0 + +	58	- 0 + + + 0	78	+ + + + - 0	98	- + + + + +	B8	+ 0 + 0 + 0 +	D8	+ 0 + + + + +	F8	+ 0 + 0 + + 0 +				
19	- + 0 - 0 + 0	39	0 + + - 0 + 0	59	0 - + + 0 +	79	+ + + + - 0	99	- + + + + +	B9	0 + 0 + 0 +	D9	0 + + + + +	F9	0 + 0 + + 0 +				
1A	- + + + 0 + 0	3A	0 + - 0 + + 0	5A	0 - + + 0 +	7A	+ + + + - 0	9A	- + + + + +	BA	0 + 0 + 0 +	DA	0 + + + + +	FA	0 + 0 + + 0 +				
1B	- + 0 + 0 + 0	3B	0 + + 0 + + 0	5B	0 - + 0 + + 0	7B	0 + + 0 + 0 +	9B	- + + + + +	BB	0 + 0 + 0 +	DB	0 + + 0 + 0 +	FB	0 + 0 + 0 +				
1C	+ 0 0 + 0 + 0	3C	+ 0 + 0 + + 0	5C	+ 0 - 0 + + 0	7C	- 0 + 0 + + 0	9C	+ + + + + +	BC	+ 0 + 0 + 0 +	DC	+ 0 + 0 + 0 +	FC	1 + 0 + 0 +				
1D	- + + + 0 + 0	3D	- 0 + + + 0 + 0	5D	- 0 + 0 + + 0	7D	- 0 + 0 + + 0	9D	- + + + + +	BD	0 0 + + + +	DD	- 0 + + 0 + 0	FD	0 + + + 0 +				
1E	- + 0 + 0 + 0	3E	- 0 + 0 + + 0 +	5E	- 0 + + 0 + + 0	7E	- + + + + +	9E	- + + + + +	BE	- 0 + 0 + 0 +	DE	- 0 + 0 + 0 + 0	FE	0 + 0 + 0 + -				
1F	- + 0 + 0 + 0	3F	- 0 + 0 + + 0 +	5F	- 0 + + 0 + + 0	7F	- + + + + +	9F	- + + + + +	BF	- 0 + 0 + 0 +	DF	- 0 + 0 + 0 +	FF	0 + 0 + 0 + -				

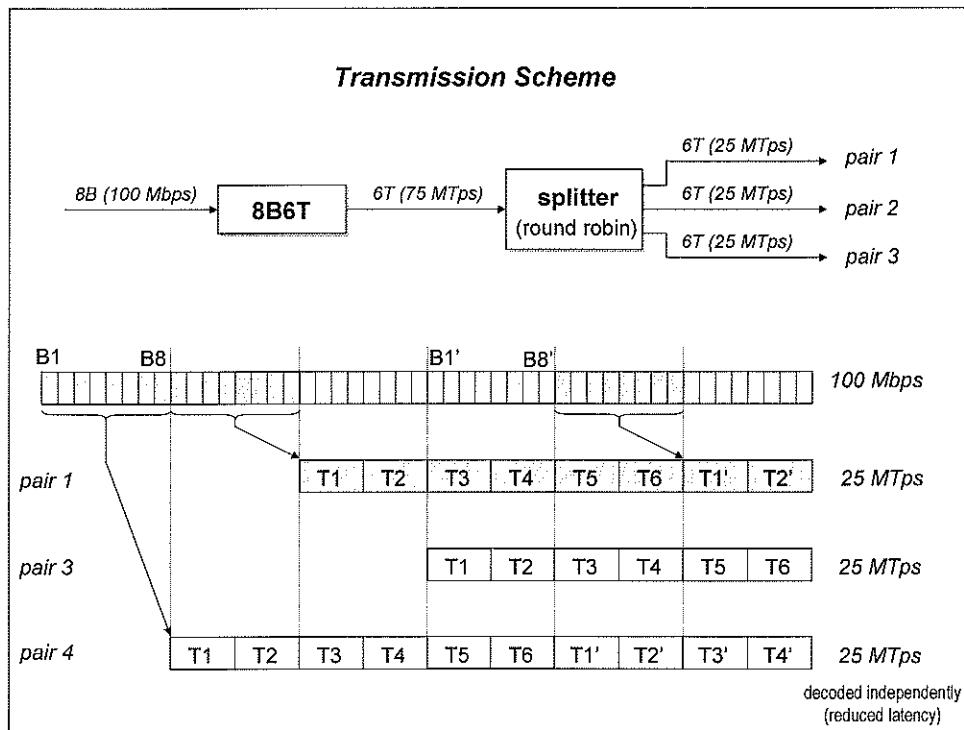
$2^8 = 256$ binary words → $3^6 = 729$ 'possible' ternary words

all selected codewords have a combined weight of 0 or +1 and at least two signal transitions

The bit rate on each pair of wires need only be $100/3 = 33.33$ Mbps. For a bit rate of 33.33 Mbps Manchester encoding requires 2 symbols per bit. This results in a baud rate of 66.66 Mbaud which exceeds the 30 Mbaud limit set for use with such cables, as above this, unacceptably high levels of crosstalk are obtained. To reduce the baud rate, a 3-level (ternary) code 8B6T is used instead of straight (2-level) binary coding. With the code 8B6T, prior to transmission, each set of 8 binary bits is first converted into 6 ternary (3-level) symbols. This block-coding method reduces the symbol rate by a factor $6/8$. This yields a symbol rate of: 33.3 Mbps $\times (6/8) = 25$ Mbaud, which is well within the set limit.

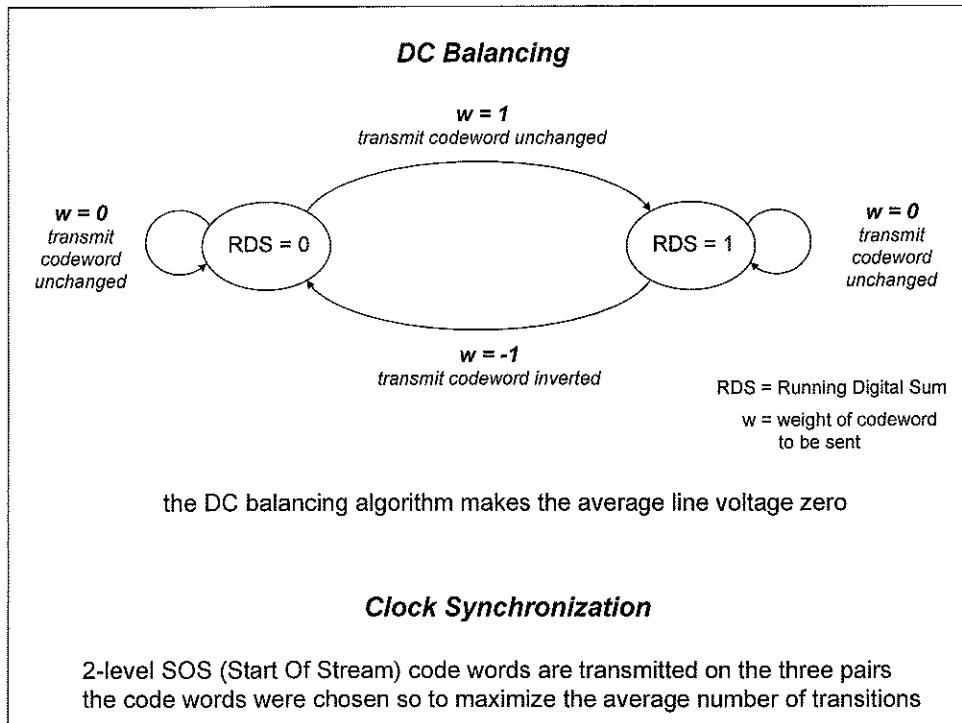
The three signal levels used are +V, 0, -V which are represented simply as +, 0, -. The codewords are selected such that the line is DC balanced, that is, the mean line signal is zero. This maximizes the receiver's discrimination of the three signal levels since these are then always relative to a constant 0 (DC) level. To achieve this, the inherent redundancy present in the use of 6 ternary symbols is exploited. The 6 ternary symbols means that there are 729 ($=3^6$) possible codewords. Since only 256 codewords are required to represent the complete set of 8-bit byte combinations, the codes used are selected, firstly, to achieve DC balance and secondly, to ensure all codewords have at least two signal transitions within them. This is done so that the receiver DLL (Digital Phase Locked Loop) maintains clock synchronization.

To satisfy the first condition, only those codewords are chosen with a combined weight of 0 or +1 and 267 codes meet this condition. To satisfy the second condition, those codes are eliminated with fewer than two transitions - five codes - and also those starting or ending with four consecutive zeros - six codes. This leaves the required 256 ($= 267 - 5 - 6$) codewords as shown in the table.



To reduce the latency during the decoding process, the 6 ternary symbols corresponding to each encoded byte are transmitted on the appropriate three wire pairs in the sequence shown in the figure above. This means that the sequence of symbols received on each pair can be decoded independently. Also, the frame can be processed immediately after the last symbol is received.

The symbols are filters before they are transmitted. This further reduces the frequency content of the signals ($> Rs/2 = 12.5 \text{ Msps}$, Nyquist).



DC Balancing

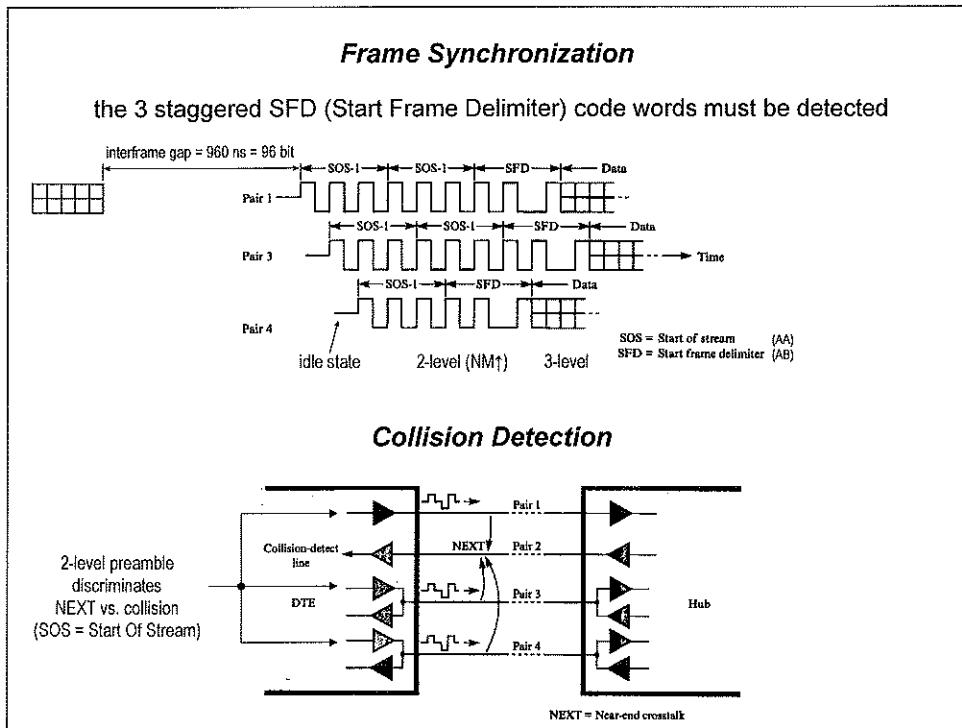
All the codewords selected have a combined weight of either 0 or +1. For example, the codeword $+---+00$ has a combined weight of 0 while the codeword $0+++--$ has a weight of +1. Clearly, if a string of codewords each of weight +1 is transmitted, then the mean signal level at the receiver will move away rapidly from the zero level, causing the signal to be misinterpreted. This is known as *DC wander* and is caused by the use of transformers at each end of the line. The presence of transformers means there is no path for DC (High Pass filter).

To overcome this, whenever a string of codewords with a weight of +1 is to be sent, the symbols in alternate codewords are inverted prior to transmission. For example, if a string comprising the codeword $0+++--$ is to be sent, then the actual codewords transmitted will be $0+++--$, $0 ---++$, $0 +++--$, $0 ---++$, and so on, yielding a mean signal level of 0. At the receiver, the same rules are applied and the alternative codewords will be reinverted into their original form prior to decoding. The procedure used for transmission is shown in a state transition diagram.

Clock Synchronisation

The preamble pattern on each pair is known as the *start of stream (SOS)* and is made up of two 2-level codewords, SOS-1 [AA = $(10101010)_B = (+- +- +-)_T$] and SFD [BB = $(10101011)_B = (+- +- -+)_T$].

To minimize any uncertainty the preamble at the start of each frame is encoded as a string of 2-level (as opposed to 3-level) symbols, that is, only positive and negative signal levels are present in each encoded symbol. This increases the signal-level amplitude variations which, in turn, helps the host/hub to discriminate between an induced NEXT signal and the preamble of a colliding frame.



Frame Synchronization and Collision detection

An example host-hub transmission without contention is shown in the figure above. Recall that a host detects a collision by detecting a signal on pair 2 while it is transmitting and, similarly, the hub detects a collision by the presence of a signal on pair 1. However the strong (unattenuated) signals transmitted on pairs 1, 3, and 4 from the host side each induce a signal into the collision detect - pair 2 - wire. This is near-end crosstalk (NEXT) and, in the limit, is interpreted by the host as a (collision) signal being received from the hub. (Also valid for transmissions from hub to host).

To minimize any uncertainty the *preamble* at the start of each frame is encoded as a string of 2-level (as opposed to 3-level) which helps the host/hub to discriminate between an induced NEXT signal and the preamble of a colliding frame. The complete pattern transmitted on each of the three pairs is shown on the figure. The SFD codeword on each pair is staggered by sending only a single SOS-1 on pair 4. This means that the first data byte of the frame is transmitted on pair 4, the next on pair 1, the next on pair 3, and so on. An acceptable start of frame requires all three SFD codes to be detected, and the staggering of them means that it takes at least four symbol errors to cause an undetectable start-of-frame error.

On *detecting a collision*, a host transmits the jam sequence and then stops transmitting. At this point, the host must be able to determine when the other host(s) involved in the collision cease transmitting in order to start the retry process. This is relatively easy since, in the nontransmitting (idle) state with 8B6T encoding, a zero signal level is present on the three data wires. This means that there is no induced NEXT signal in the collision detect wire which, in turn, enables the completion of the jam sequence from the hub side to be readily determined. To improve the utilization of the cable, the interframe gap time is reduced from 9.6 μ s to 960 ns (96 bits). The collision diameter is also reduced from 2.5 km to 200m (factor 10).

3.3 100 BASE FX			
<i>Line Code: 4B5B + NRZI</i>			
<i>data symbol</i>	<i>4B</i>	<i>5B</i>	<i>NRZI</i>
D0	0000	11110	
D1	0001	01001	
D2	0010	10100	
D3	0011	10101	
D4	0100	01010	
D5	0101	01011	
D6	0110	01110	
D7	0111	01111	
D8	1000	10010	
D9	1001	10011	
DA	1010	10110	
DB	1011	10111	
DC	1100	11010	
DD	1101	11011	
DE	1110	11100	
DF	1111	11101	

<i>control symbol</i>	<i>5B</i>	<i>NRZI</i>	<i>usage</i>
Q	00000		Quiet line state
I	11111		Idle line state, preamble
H	00100		Halt line state
J	11000		Start-of-Stream delimiter 1
K	10001		Start-of-Stream delimiter 2
T	01101		End-of-Stream delimiter 1
R	00111		End-of-Stream delimiter 2
S	11001		Reset status
			Set status

control = transparent to frame content

inversion of signal levels allowed

clock synchronization

- continuous transitions in Idle state
- at least two 1's in the 5B code, results in at least two transitions in the NRZI code

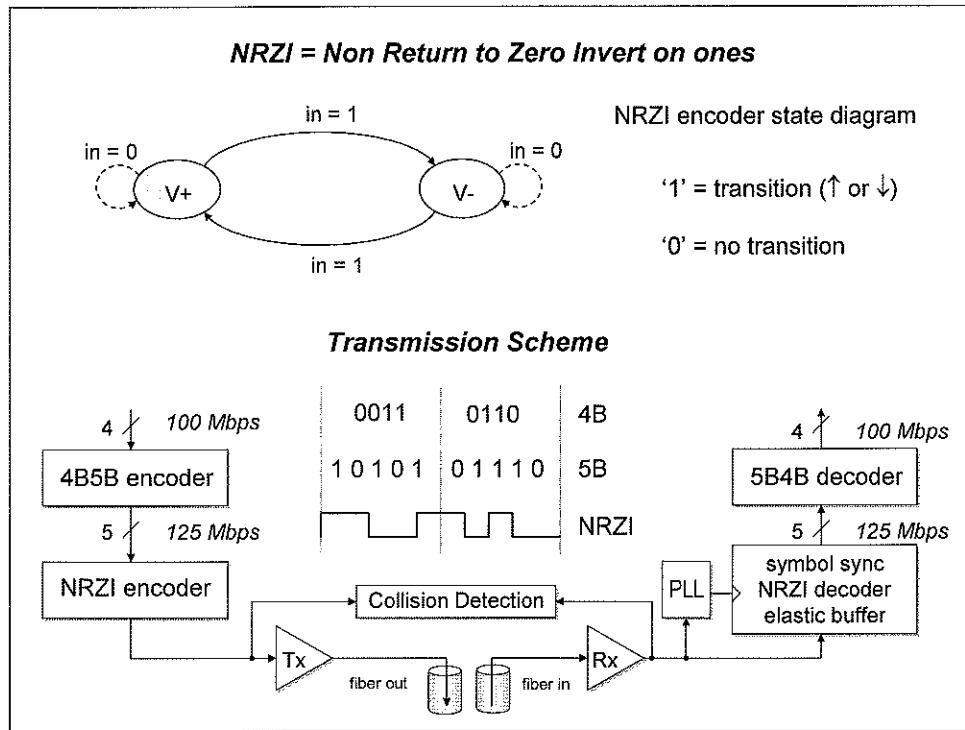
4B/5B - NRZI

This scheme, which is actually a combination of two encoding algorithms, is used both for 100 BASE-X and FDDI.

In the *4B/5B code scheme*, encoding is done four bits at a time; each four bits of data are encoded into a symbol with five *code bits*, such that each code bit contains a single signal element. The block of five code bits is called a code group. In effect, each set of 4 bits is encoded as 5 bits. The efficiency is thus reduced to 80%; 100 Mbps is achieved with 125 Mbaud.

To ensure synchronization, there is a second stage of encoding: each code bit of the 4B/5B stream is treated as a binary value and encoded using *Nonreturn to Zero Inverted (NRZI)*. In this code, a binary 1 is represented with a transition at the beginning of the bit interval (toggle), and a binary 0 is represented with no transition at the beginning of the bit interval (no toggle); there are no other transitions. The advantage of NRZI is that it employs differential encoding. In differential encoding, the signal is decoded by comparing the polarity of adjacent signal elements rather than the absolute value of a signal element. A benefit of this scheme is that it is generally more reliable in detecting a transition in the presence of noise and distortion than in comparing a value to a threshold. Inversion of the signal levels has no influence on the decoding process.

In the table the *4B/5B + NRZI symbol encoding* is shown. Each 5-bit code group pattern is shown, together with its NRZI realization. Because four bits are encoded with a 5-bit pattern, only 16 of the 32 possible patterns are needed for data encoding. The codes selected to represent the 16 4-bit data blocks are such that a transition is present at least twice for each 5-code group code (at least two 1's in the 5B code). No more than three zeros in a row are allowed across one or more code groups. This guarantees adequate clock synchronization.



Symbols not used to represent data are either declared invalid or are assigned special meaning as control symbols (see table on previous page). Some nondata symbols:

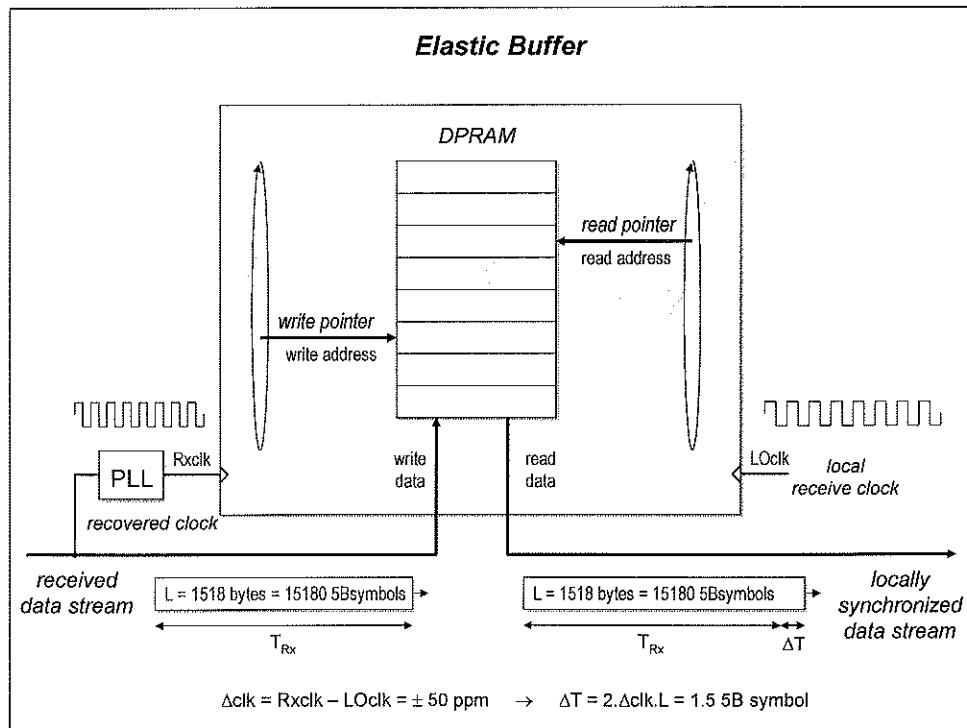
- *Idle*. The idle symbol is transmitted between data transmission sequences. It consists of a constant flow of binary ones, which in NRZI comes out as a continuous alternation between the two signal levels. This continuous fill pattern establishes and maintains clock synchronization at the receiver and is used in the CSMA/CD protocol to indicate that the shared medium is idle.
- *Start-of-stream delimiter (J and K)*. Used to delineate the starting boundary of a data transmission sequence. It consists of a J symbol followed by the K symbol (JK symbol pair). They are unique and transparent to the frame content.
- *End-of-stream delimiter (T and R)*. Used to terminate normal data transmission sequence. It consists of two different symbols T and R that are unique and transparent to the frame content.

Symbol Synchronization

The local clock used in the physical interface is 125 MHz which, because of 4B/5B encoding, yields a data rate of 100 Mbps. Since all transitions are encoded into 5-bit symbols, each 5-bit symbol must first be buffered at the receiver before it can be decoded. However the use of two symbols (J and K, the Start-of-stream delimiters) to establish correct symbol boundaries, means that a 10-symbol buffer is used at the receiver. This is the latency (or elastic) buffer, since it introduces 8 data bits of delay (latency).

Collision Detection

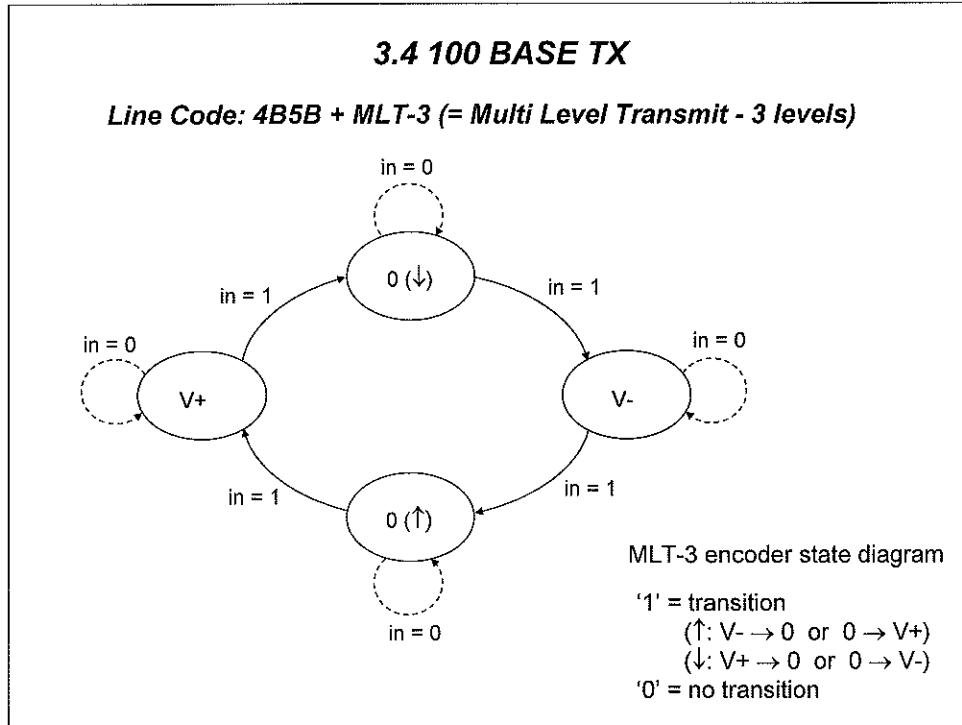
The cable comprises two fibers, one for host-hub and one for hub-host. Collisions are detected if a (colliding) signal is present on the receiving fiber during the period the host is transmitting.



As every host has its own local clock, there will always be a slight difference between the local clock frequency of the transmitting host, and that of the receiving host. As the clocks are totally independent, the chances are that they can be completely out of phase. What this means is that there is a slight timing difference between the incoming recovered clock, and the local clock within the receiver of the host.

For this purpose an *elastic buffer* is implemented within the receiver portion of the host, which is a FIFO (first in, first out) based temporary storage (based on a DPRAM = Dual Port Random Access Memory) for the incoming recovered bit stream. The elastic buffer is placed before the 5B4B decoding function in the receiver. The buffer compensates for the difference between the transmit and receive clock frequencies by clocking data into the FIFO at the recovered clock rate (derived by the PLL from the incoming data), and out of the FIFO at the host's local receive clock rate. Most hardware VLSI implementations perform this function at symbol-pair level at the byte clock rate (12.5 MHz).

In between frame transmissions, the FIFO's write pointer (input) is repositioned to the centre of the FIFO (half of the FIFO capacity) as a drift can occur in either direction. Certain rules govern cases where this may not occur, such as an incorrect number of inter-frame symbols being detected. Assume a local clock crystal accuracy of 50 p.p.m. (parts per million). Then for a maximum Ethernet frame size of 1518 bytes (= 12.144 bits = 15.180 5Bsymbols), a possible 1.2 bit ($12.144 \text{ bits} \times 50 \times 10^{-6} \times 2$) difference can occur. This assumes both host clocks are out by +50 and -50 p.p.m. respectively (hence the factor of 2). To cater for a drift of the clocks in either direction (low Tx clock and high Rx clock and vice versa), then at least 3 bits are in theory needed for an elasticity buffer. In practice, a larger size is used (as many as 10 bits) to cater for differences beyond 50 p.p.m.



Although 4B/5B-NRZI is effective over optical fiber, it is not suitable for use over twisted pair. The reason is that the signal energy is concentrated in such a way as to produce undesirable radiated transmissions from the wire. MLT-3, which is used on both 100 BASE TX and the twisted pair version of FDDI, is designed to overcome this problem. The following steps are involved:

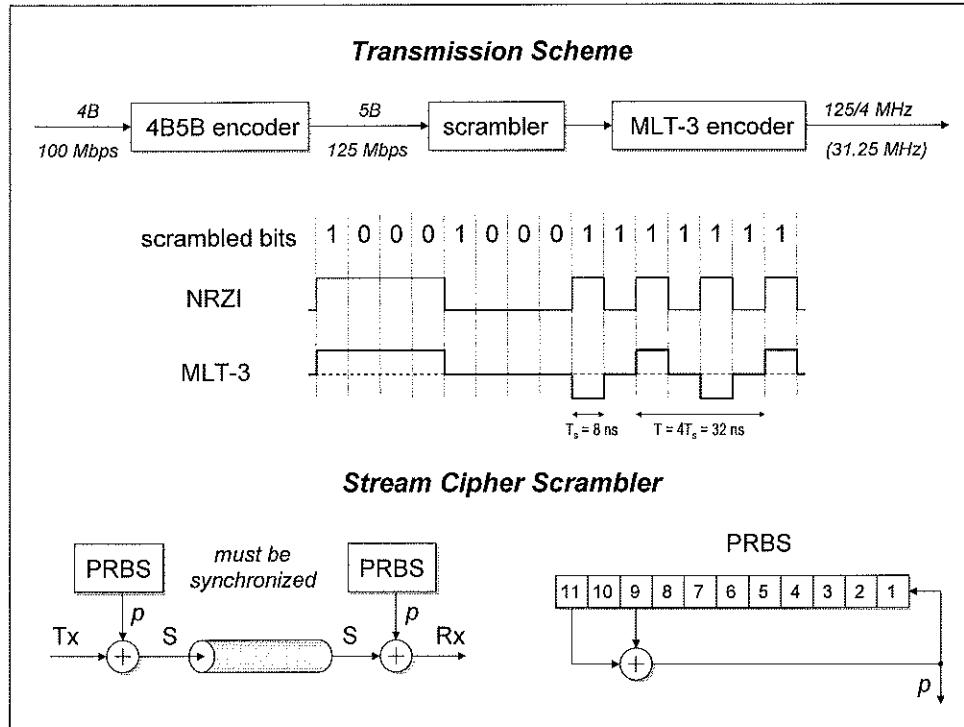
1. *4B/5B conversion*: The 4B/5B conversion is done without NRZI.
2. *Scrambling*: The bit stream is scrambled to produce a more uniform spectrum distribution for the next stage. The peaks in the energy spectrum are reduced.
3. *Encoder*: The scrambled bit stream is encoded using a scheme known as MLT-3.
4. *Driver*: The resulting encoding is transmitted.

The effect of the MLT-3 scheme is to concentrate most of the energy in the transmitted signal below 30 MHz, which reduces radiated transmissions.

The MLT-3 encoding produces an output that has a transition for every binary one that uses three levels: a positive voltage (+V), a negative voltage (-V) and no voltage (0). The encoding rules explained with reference to the encoder state diagram:

1. If the input bit is '0', then the output value is the *same as the preceding value*.
2. If the input bit is '1', then the output value involves a *transition*:
 - a. If the preceding output value was either +V or -V, then the next output value is 0.
 - b. If the preceding output value was 0, then the next output value is nonzero, and that output is of the opposite sign to the last nonzero output.

Every time there is an input of 1, there is a transition (similar to NRZI). The direction of the transition (\uparrow or \downarrow) depends on both the current level, and the direction of the last transition. With the occurrences of +V and -V, the direction reverses.



Maximum Transmission Frequency

When there is a string of '1's, the maximum frequency is attained. The data stream has a period of 4 bits, instead of 2, as in the NRZI example above it. This means that the maximum fundamental frequency is $125 \text{ MHz}/4 = 31.25 \text{ MHz}$. Thus lowering the highest frequency component of the data bit stream. This is one of the main reasons that the MLT-3 encoding allows the distance requirements of UTP to be met.

Stream Cipher Scrambler

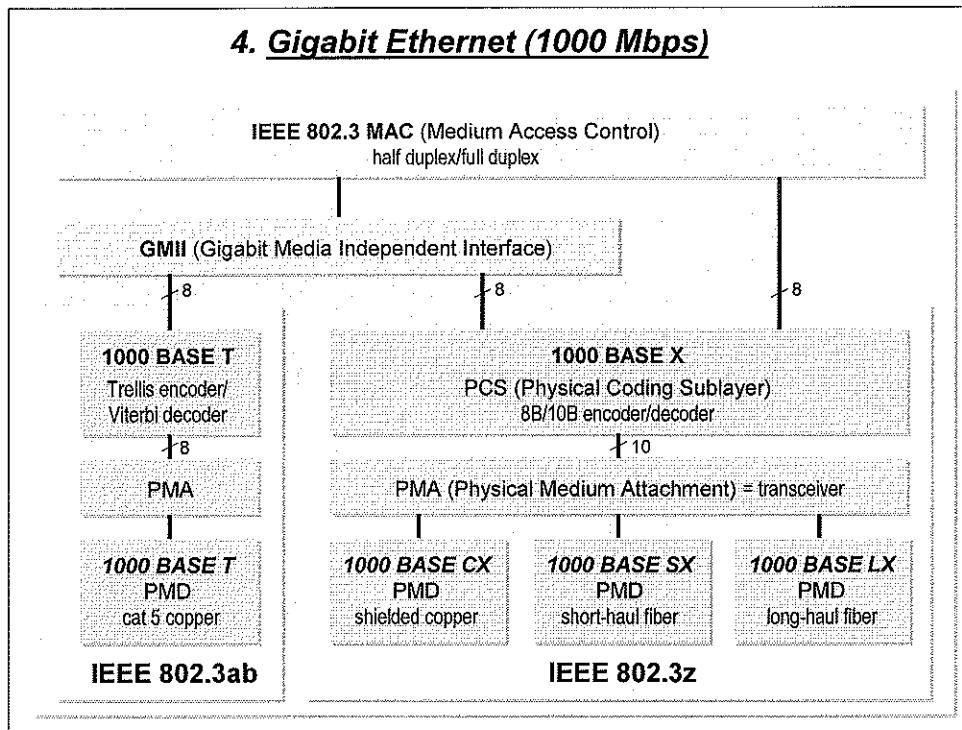
The 4B/5B serial data stream is modified by the addition of a random component, to reduce energy peaks within the MLT-3 encoded streams. Some 0's are changed to 1's, some 1's are changed to 0's and some bits are left the same. The result is a reduced run-length of the same-value bits, increased transition density and easier clock recovery. This makes the NEXT quasi random, while noise is common mode.

This random component is removed at the receiving end to re-create the original data bit stream. The PRBS generator (feedback shift register) runs at the bit transmission rate.

In order to reconstruct the original data stream, the receiving descrambler must be synchronized to the transmitting scrambler. This is accomplished by the use of control symbols that produce predictable patterns in the scrambled data stream. Once the descrambler is synchronized, its random bit stream may be XORed with the scrambled data to reproduce the original data stream. This is based on the basic property of the XOR function that data XORed with itself, produces a zero result.

$$\text{Transmitter: } S = Tx \oplus p$$

$$\text{Receiver: } Rx = S \oplus p = (Tx \oplus p) \oplus p = Tx \oplus (p \oplus p) = Tx \oplus 0 = Tx$$



The strategy for Gigabit Ethernet is the same as that for 100-Mbps Ethernet. While defining a new medium and transmission specification, Gigabit Ethernet retains the carrier sense multiple access collision detect (CSMA/CD) protocol and frame format of its 10- and 100-Mbps predecessors. So it is compatible with the slower Ethernets, providing a smooth migration path. As more organizations move to 100-Mbps Ethernet, putting huge traffic loads on backbone networks, demand for Gigabit Ethernet is intensifying.

The overall protocol architecture for Gigabit Ethernet is shown.

MAC: The Media Access Control layer is an enhanced version of the basic 802.3 MAC algorithm.

GMII: A separate Gigabit Medium-Independent Interface has been defined and is *optional* for all the medium options except unshielded twisted-pair (UTP). The GMII defines independent 8-bit-parallel transmit and receive synchronous data interfaces. It is intended as a chip-to-chip interface that lets system vendors mix MAC and physical sublayer (PHY) components from different manufacturers.

PCS: Two signal encoding schemes are defined at the physical layer. The 8B/10B scheme is used for optical fiber and shielded copper media, and the pulse amplitude modulation (PAM)-5 is used for UTP.

4.1 PMD (*Physical Medium Dependent*)

	<i>medium</i>	<i>modal BW (MHz.km)</i>	<i>range (m)</i>	<i>coding</i>	<i>frequency (per pair)</i>
1000 BASE T	4 pair cat 5(E) UTP	/	100	Viterbi 5-level PAM	125 Mbaud
1000 BASE CX	2 pair STP (twinax)	/	25	8B/10B	1.25 Gbaud
1000 BASE SX (short λ)	2 pair fiber $\lambda = 770-860 \text{ nm}$ 62.5 μm MM 62.5 μm MM 50 μm MM 50 μm MM	160 200 400 500	2-220 2-275 2-500 2-550	8B/10B	1.25 Gbaud
1000 BASE LX (long λ)	2 pair fiber $\lambda = 1270-1355 \text{ nm}$ 62.5 μm MM 50 μm MM 50 μm MM 10 μm SM	500 400 500 > 5000	2-550 2-550 2-550 2-5000	8B/10B	1.25 Gbaud

MM = Multi-Mode fiber SM = Single-Mode fiber

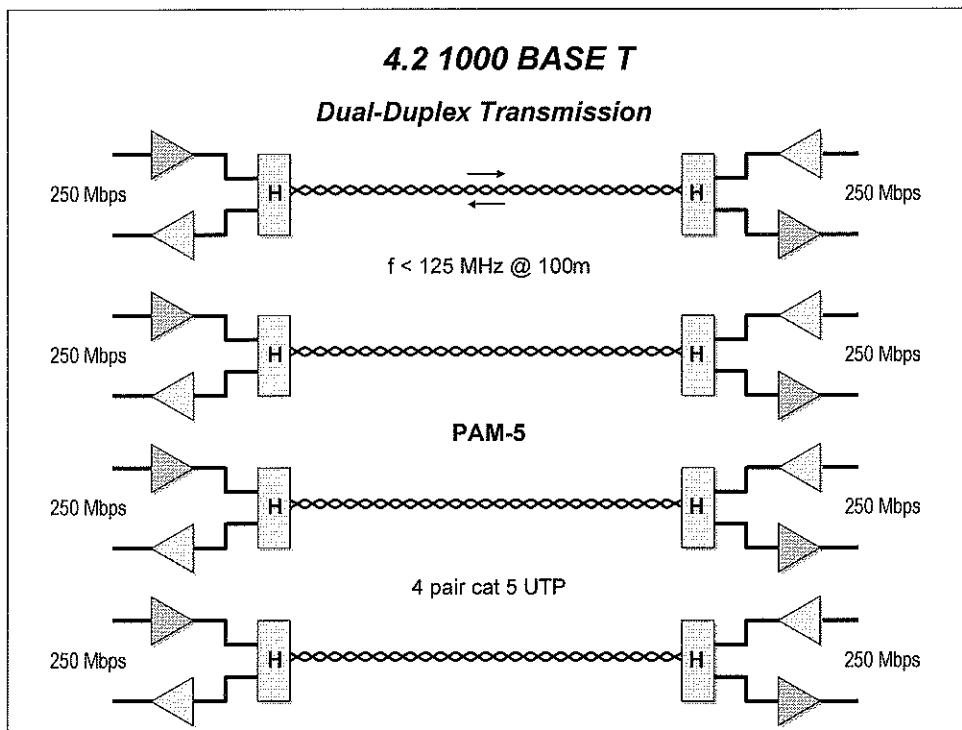
The Physical Layer of Gigabit Ethernet uses a mixture of proven technologies from the original Ethernet and the ANSI X3T11 Fiber Channel Specification. Gigabit Ethernet supports 4 physical media types. These are defined in 802.3z (1000 BASE X) and 802.3ab (1000 BASE T).

The 1000 BASE X standard is based on the Fiber Channel Physical Layer. This is an interconnection technology for connecting workstations, supercomputers, storage devices and peripherals. Fiber Channel has a 4 layer architecture. The lowest two layers FC-0 (interface and media) and FC-1 (Encode/Decode) are used in Gigabit Ethernet. Since Fiber Channel is a proven technology, re-using it, greatly reduces the Gigabit Ethernet standard development time.

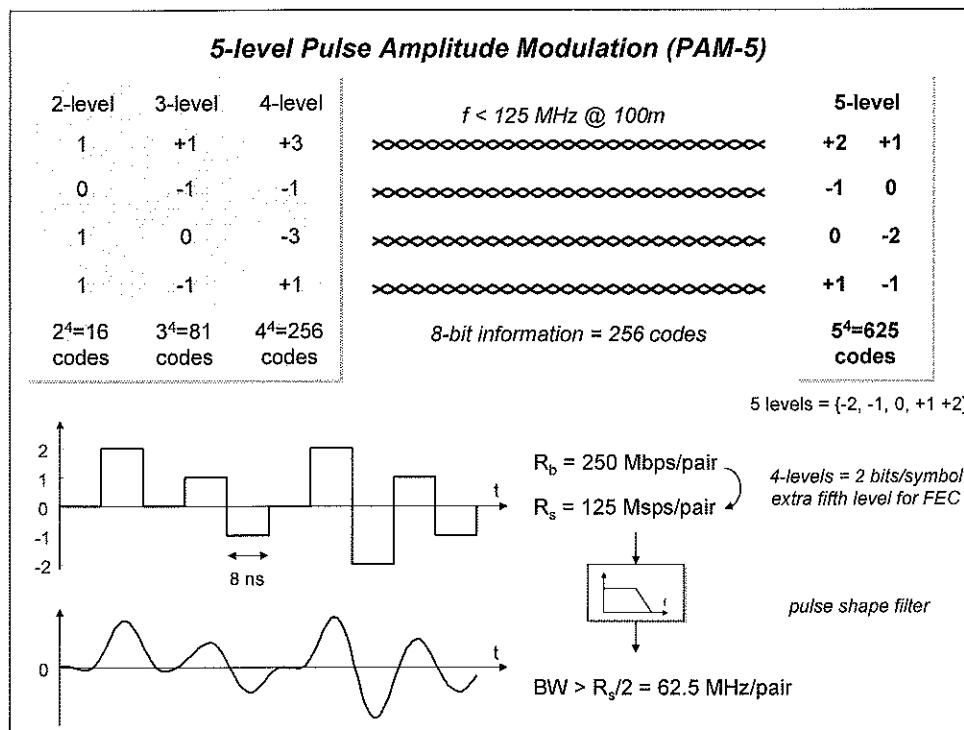
Three types of media are included in the 1000 BASE X standard:

- 1000 BASE SX 850 nm (short wave) laser on multi mode fiber. It supports duplex links of up to 275m using 62.5- μm multi-mode or up to 550m using 50- μm multi-mode fiber (50- μm fiber has a larger modal bandwidth than 65- μm).
- 1000 BASE LX 1300 nm (long wave) laser on single mode and multi-mode fiber. It supports duplex links of up to 550m of 62.5- μm or 50- μm multimode fiber or up to 5km of 10- μm single-mode fiber .
- 1000 BASE CX short haul copper "twinax" STP (Shielded Twisted Pair) cable. It supports 1-Gbps links among devices located within a single room or equipment rack, using copper jumpers (specialized STP cable that spans no more than 25 m) .

1000 BASE T is a standard for Gigabit Ethernet over long haul copper UTP. The standard allows 1 Gbps transmission up to 100 m over 4 pairs of category 5 UTP. This standard is developed by the 802.3ab task force.

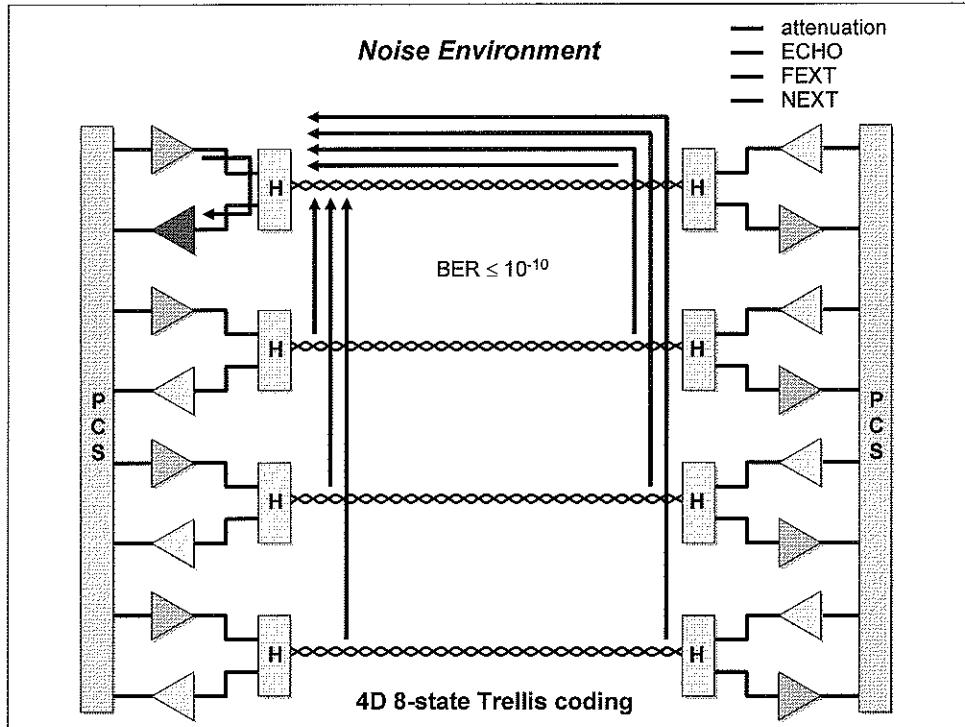


Gigabit Ethernet uses bidirectional transmission. The signals are transmitted simultaneously in both directions on the same wire pair (full duplex); that is, both the transmit and receive pair occupy the same wire pair. 1000 BASE T uses bidirectional transmission on four wire pairs. Bidirectional data transmission on a single pair is enabled by devices called hybrids (H): duplex by echo cancellation. The hybrid stops the local transmitted signals from being mixed with the local received signals.



For 1000 BASE T, the encoding scheme used is PAM-5, over four twisted-pair links. Therefore, each link must provide a data rate of 250 Mbps. PAM-5 provides better bandwidth utilization than simple binary signaling by using five different signaling levels. Each signal element can represent two bits of information (using four signaling levels). In addition, a fifth signal level is used in a forward error correction scheme.

From the MAC layer, frames of 8 bits are transmitted to the physical coding sublayer (PCS) through the Gigabit Media Independent interface (GMII). To encode eight GMII bits, $2^8 = 256$ codes are needed. A two-level signal used on each of the four pairs of transmission would enable the coding of $2^4 = 16$ data codes. Similarly a three-level signal would give $3^4 = 81$ codes. A five-level signal gives $5^4 = 625$ potential codes and has been chosen by the IEEE for implementation. The specific five-level signal used by 1000 BASE T is Pulse Amplitude Modulation 5 (PAM-5).



The transmit signal is subject to impairments introduced by the cabling and external noise sources. In order for the receiver to operate reliably, the impairments to the transmit signal need to be controlled.

Attenuation is a reduction in signal power due to cabling losses and a function of frequency and cable length.

Echo is the result of bidirectional transmission on the same wire. Echo is the combined effect of the cabling return loss and the hybrid function. Return loss is a measure of the reflected energy caused by impedance mismatches in the cabling system.

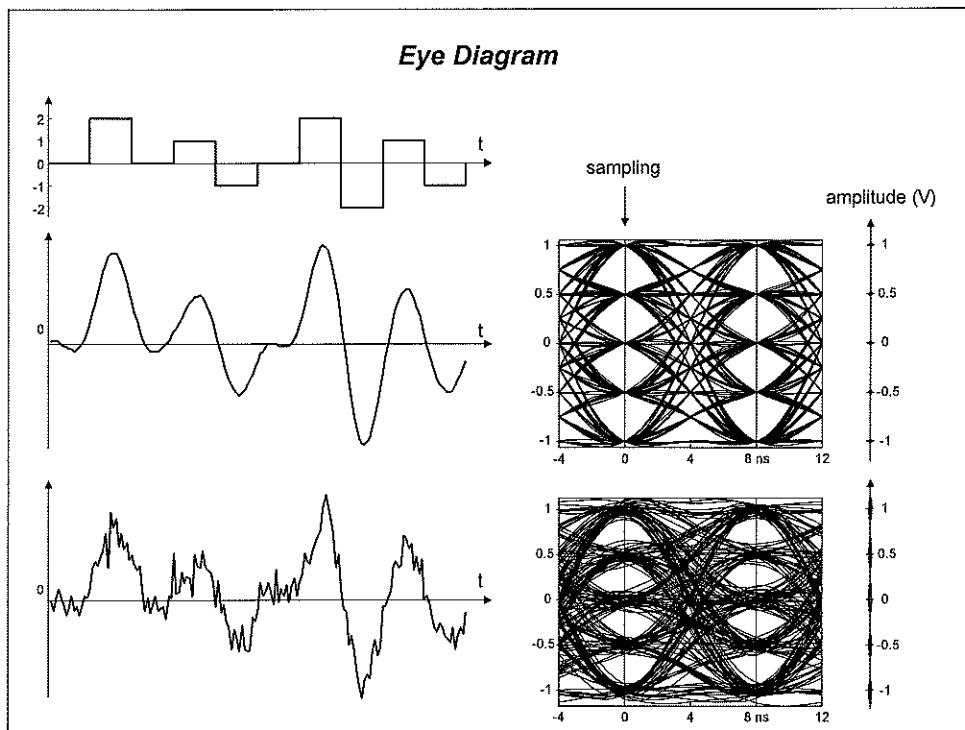
FEXT is the noise induced by a transmitter at the near-end into a far-end receiver due to unwanted signal coupling (typical for multi-pair transmission).

NEXT is the noise induced by a transmitter to a neighbouring receiver due to unwanted signal coupling.

Delay skew is the difference in delay between pairs.

The signal-to-noise ratio (SNR), the ratio between the impairments (typically referred to as noise) and the transmit signal, is maintained in order to achieve an acceptable bit error rate (BER) of 10^{-10} .

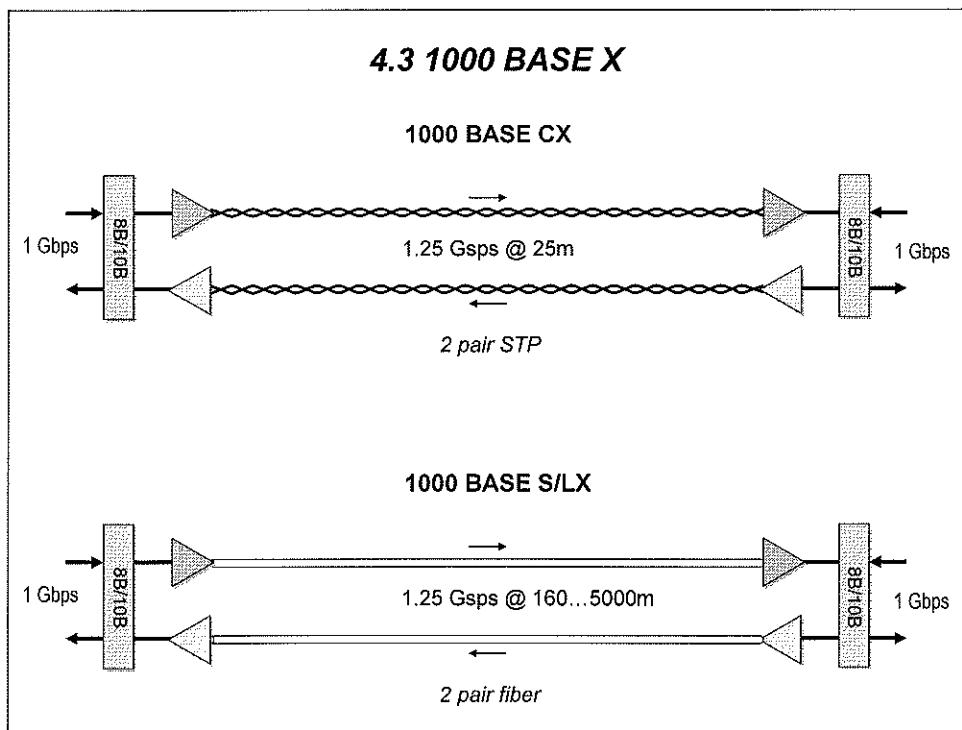
All Ethernet twisted-pair technologies are subject to signal impairment. But in the case of 1000 BASE T, these disturbances are cancelled. Echo, FEXT, NEXT is countered by echo/crosstalk cancellation. The effect of attenuation is countered at the receiver by the equalization of the signal, which compensates for the cabling losses. All these actions provide additional immunity to noise but require complex DSP algorithms (a proven technology also used in voice band telephone modems).



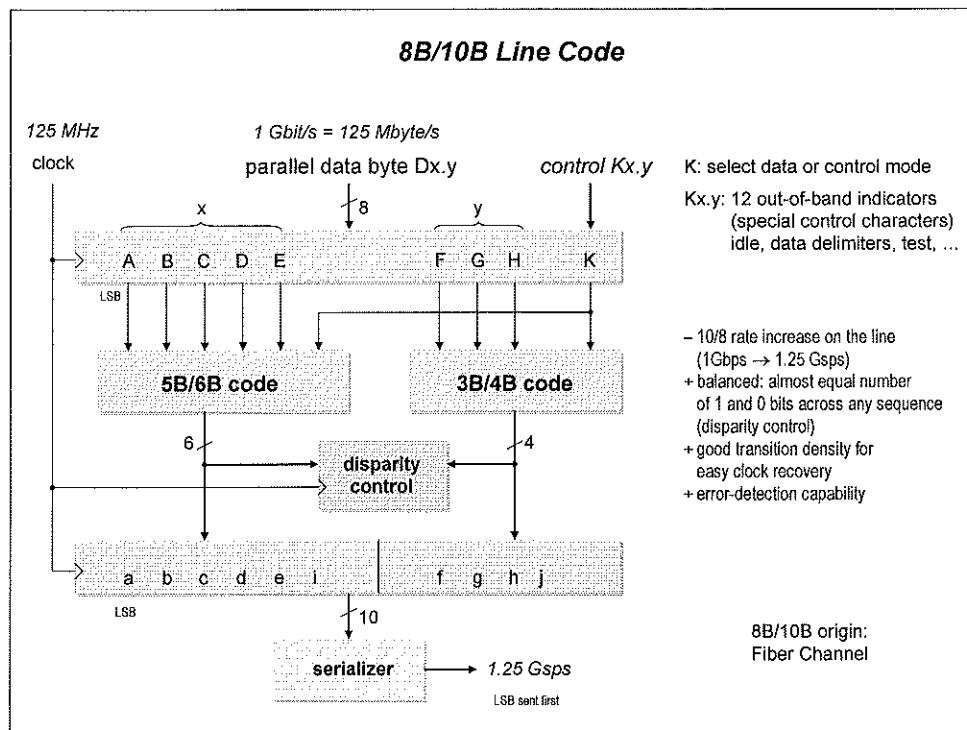
The eye patterns of 1000 BASE T signaling are shown. The eye pattern illustrated was produced by a modulated random-data waveform, with each symbol period tracing from left to right and starting in the same place on the left. As shown in the figure the PAM-5 modulation of 1000 BASE T provides closer consecutive levels of signals (5 levels, separation of 0.5 V) and hence a greater sensitivity to transmission distortions than 100 BASE TX (3 levels, separation of 1 V). That is, it has a reduced signal-to-noise margin compared to 100 BASE T. (100 BASE TX uses three-level signaling: MLT-3).

The reduced noise margin lost at the level of PAM-5 is recovered thanks to the use of convolution coding. Convolution coding implemented by 1000 BASE T (called Trellis coding) allows error detection and correction by the receiver (through Viterbi decoding). These are established, proven technologies defined and used in modems. In comparison, 100 BASE TX uses block coding (4B5B coding, four bits coded by five symbols). Block coding uses simple codes that do not offer error detection or correction.

In fact, the use of Trellis coding and Viterbi decoding makes 1000 BASE T even more resilient to external noise than 100 BASE TX, since 1000 BASE T transmits 'uncorrelated' symbols in the transmitted symbol stream; no correlation is allowed between symbol streams travelling in both directions on any pair combination, and no correlation is allowed between symbol streams on each pair. External noise pickup is generally correlated (common mode!) to each pair. External noise can be cancelled statistically, providing improvements in noise immunity that are not available in the technology of 100 BASE TX.



The 1000 BASE X standard uses unidirectional transmission. Signals are transmitted in one direction on a single wire pair. By using two pairs, full-duplex transmission is supported.

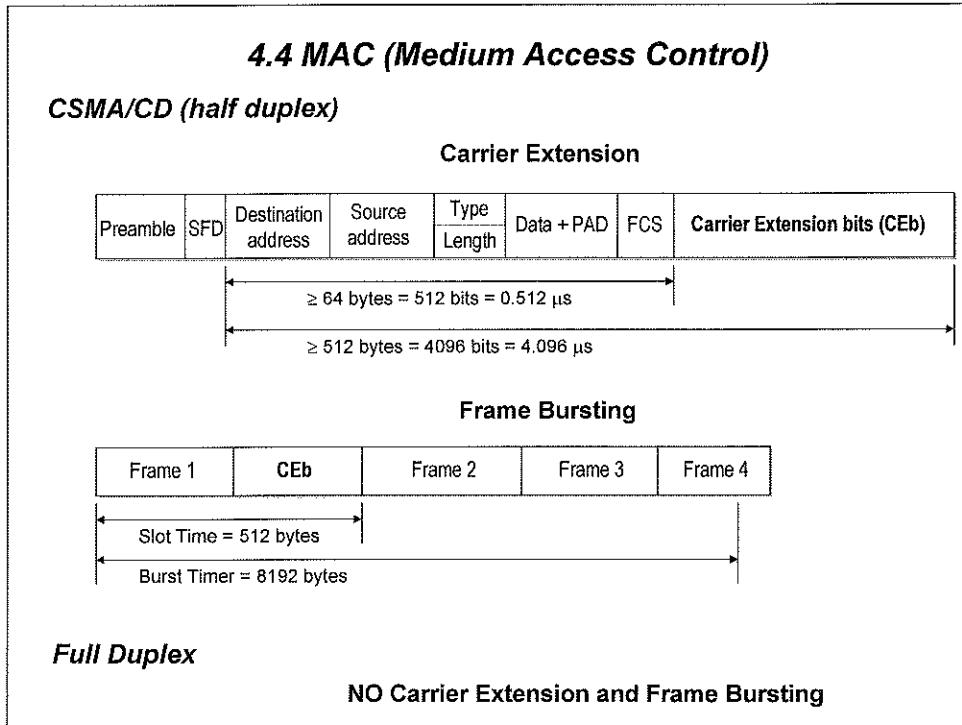


The encoding scheme used for all the Gigabit Ethernet options except twisted-pair is 8B/10B. This scheme is also used in Fiber Channel. With 8B/10B, each 8 bits of data is converted into 10 bits for transmission. The 8B/10B scheme was developed and patented by IBM (200-megabaud ESCON interconnect system). Advantages:

- It is well balanced, with minimal deviation from the occurrence of an equal number of 1 and 0 bits across any sequence (minimum DC content on the serial line).
- It provides good transition density for easier clock recovery.
- It provides useful error-detection capability.

The 8B/10B code is an example of the more general mBnB code, in which m binary source bits are mapped into n binary bits for transmission. Redundancy is built into the code to provide the desired transmission features by making $n > m$. The 8B/10B code actually combines two other codes, a 5B/6B code and a 3B/4B code. The use of these two codes is simply an artefact that simplifies the definition of the mapping and the implementation; the mapping could have been defined directly as an 8B/10B code. In any case, a mapping is defined that maps each of the possible 8-bit source blocks into a 10-bit code block. *Disparity control* keeps track of the excess of zeros over ones or ones over zeros. An excess in either direction is referred to as a disparity. If there is a disparity, and if the current code block would add to that disparity, then the disparity control block complements the 10-bit code block. This complement has the effect of either eliminating the disparity or at least moving it in the opposite direction of the current disparity. Disparity determines the DC component of a serial line.

The encoding mechanism also includes a control line input, K , which indicates whether the lines A through H are data or control bits. In the latter case, a special nondata 10-bit block is generated. A total of 12 of these nondata blocks are defined as valid in the standard. These blocks are used for synchronization and other control purposes.



Media Access Layer

Half Duplex

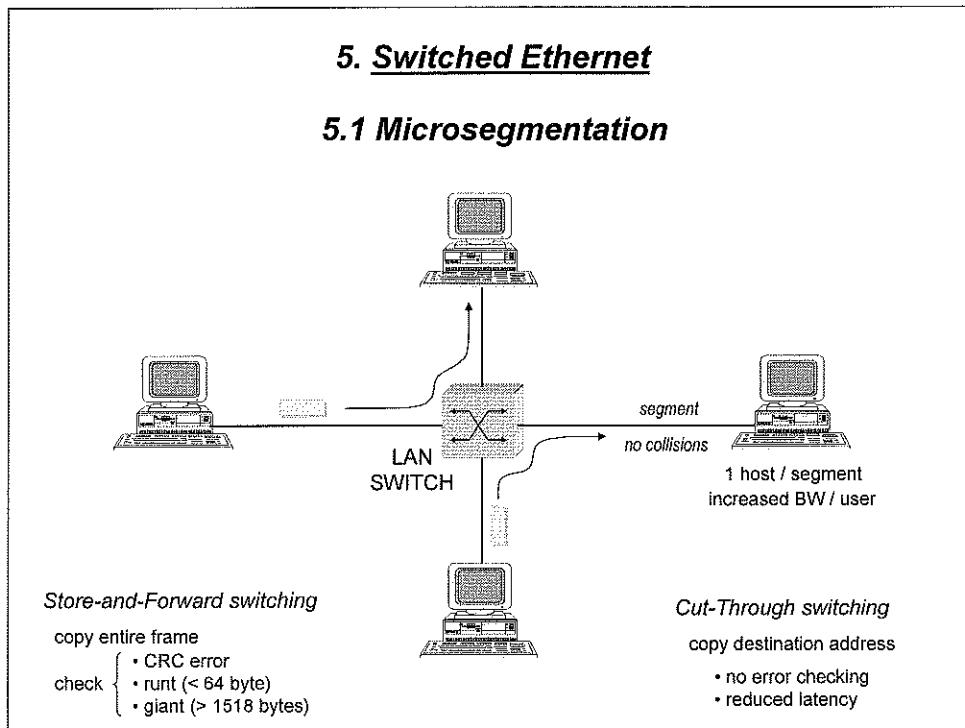
For traditional Ethernet hub operation, in which only one station can transmit at a time (half-duplex), the basic CSMA/CD scheme has two enhancements:

- **Carrier extension:** Carrier extension appends a set of special symbols to the end of short MAC frames so that the resulting block is at least 4096 bit-times in duration, up from the minimum 512 bit-times imposed at 10 and 100 Mbps. This extension makes the frame length of a transmission longer than the propagation time at 1 Gbps over 100m. When this is not done, the cable length would have to be reduced to about 10m.
- **Frame bursting.** This feature allows for multiple short frames to be transmitted consecutively, up to a limit, without relinquishing control for CSMA/CD between frames. The first packet is padded to the slot time if necessary using carrier extension. Subsequent packets are transmitted back to back (no idle periods on the medium, which inhibits any waiting host transmitting) until a burst timer (8192 bytes) expires. Frame bursting avoids the overhead of carrier extension when a single station has a number of small frames ready to send.

Full Duplex

With a LAN switch (full-duplex operation), which provides dedicated rather than shared access to the medium, the carrier extension and frame bursting features are not needed. They are unnecessary because data transmission and reception at a host can occur simultaneously without interference and with no contention for a shared medium. All the gigabit products on the market use a switching technique, and so do not implement the carrier extension and frame bursting.

The *pause protocol* (defined for 100-Mbps Ethernet) is expanded by allowing asymmetric flow control. Using the auto negotiation protocol, a device may indicate that it may send pause frames to its link partner but will not respond to pause frames from its partner.



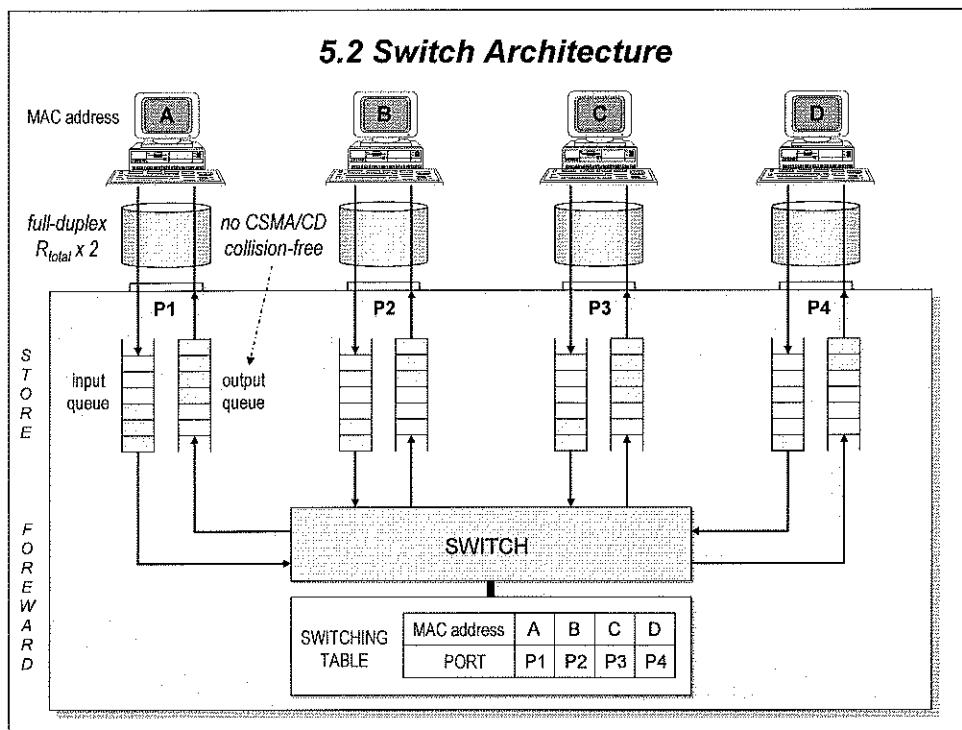
A LAN switch is a device that provides much higher port density at a lower cost than traditional bridges. For this reason, LAN switches can accommodate network designs featuring fewer users per segment, thereby increasing the average available bandwidth per user.

The trend toward fewer users per segment is known as *microsegmentation*. Microsegmentation allows the creation of private or dedicated segments, that is, one user per segment. Each user receives instant access to the full bandwidth and does not have to contend for available bandwidth with other users. As a result, collisions (a normal phenomenon in shared-medium networks employing hubs) do not occur. A LAN switch forwards frames based on either the frame's layer 2 address (Layer 2 LAN switch), or in some cases, the frame's layer 3 address (multi-layer LAN switch, router functions are integrated). A LAN switch is also called a frame switch because it forwards layer 2 frames, whereas an ATM switch forwards cells.

LAN switches can be characterized by the forwarding method they support.

With the *Store-and-Forward switching* method, the LAN switch copies the entire frame into its onboard buffers and computes the cyclic redundancy check (CRC). The frame is discarded if it contains a CRC error or if it is a *runt* (less than 64 bytes including the CRC) or a *giant* (more than 1518 bytes including the CRC). If the frame does not contain any errors, the LAN switch looks up the destination address in its switching table, determines the outgoing interface and forwards the frame toward its destination.

With the *Cut-Through switching* method, the LAN switch copies only the destination address (the first 6 bytes following the preamble) into its onboard buffers. It then looks up the destination address in its switching table, determines the outgoing interface, and forwards the frame toward its destination. A cut-through switch provides reduced latency because it begins to forward the frame as soon as it reads the destination address and determines the outgoing interface.



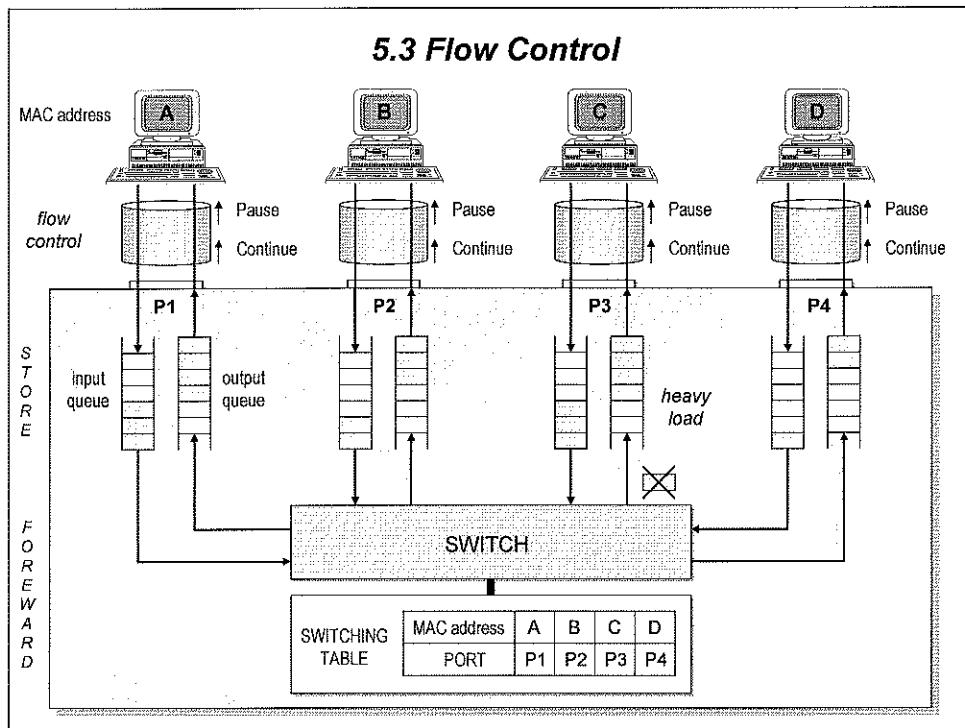
With CSMA/CD the throughput for a host reduces with increased load. A Fast Ethernet switch allows multiple access/transfers to be in progress concurrently by the use of a *switched hub*, and *duplex* working over the circuits that connect the hosts to the hub.

The general architecture of a switching hub is shown. Each host is connected to the hub by means of a pair of (duplex) lines which are implemented as two UTP pairs or two multimode fiber cables, in the case of 100 BASE X. Each UTP pair or each fiber is used to transmit at 100Mbps one in each direction of transmission. *Full-duplex* working doubles the throughput.

With a switching hub, CSMA/CD is not used. Dedicated *collision-free communication* between network devices increases throughput.

Because each host can transmit frames simultaneously, a frame may be received at multiple input ports of the hub - and hence require processing - simultaneously. Similarly, two or more frames may require the same output line simultaneously. Hence associated with each input and output line is a memory buffer (queue) that can hold several frames waiting to be either processed (input) or transmitted (output). The frames - memory pointers to the start of the frame in practice - are stored in an input FIFO queue. The switch then reads the pointer to the frame at the head of each input queue in turn, obtains the destination MAC address from its head, and forwards the frame pointer to the tail of the required output queue to await transmission.

In order to retain the same connectionless mode of operation of the other LAN types, when the switch is first brought into service - and subsequently at periodic intervals - the switch enters a *learning state*. This is similar to that used in transparent bridges. Hence when in the learning state, the switch simply initiates the onward transmission of a copy of each frame received from an input line onto *all* output lines. Prior to doing this, however, the switch reads the source address from the head of the frame and keeps a record of this, together with the input port number on which the frame was



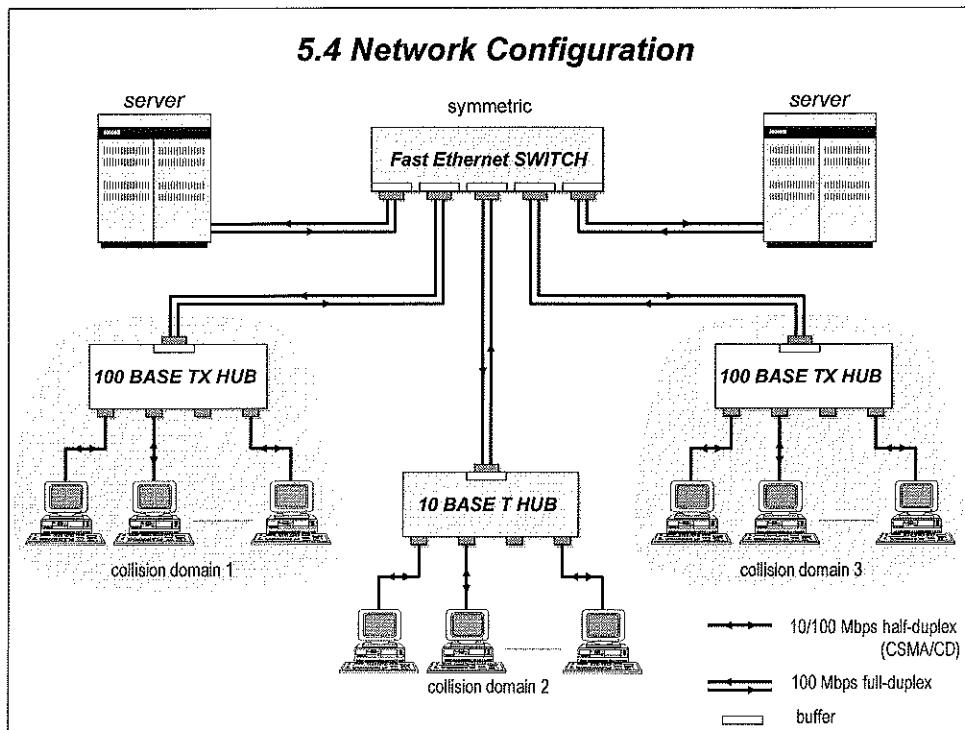
received, in a *switching table* (*forwarding table*). The contents of the table are then subsequently used to switch each received frame to a specific output port. There is a store-and-forward delay associated with a switch. Also, as with a bridge, the FCS at the tail of each frame is used to check for the presence of transmission errors prior to the frame being forwarded and corrupted frames are discarded.

Flow Control

Full-duplex operation requires concurrent implementation of the optional flow-control capability that allows a receiving node (such as a network switch port) that is becoming congested to request the sending node (such as a file server) to stop sending frames. Under heavy load conditions it is possible for all the frame buffers within the switch to become full. The switch must discard any new frame(s). Alternatively flow control can be used. When the level of memory in use reaches a defined threshold, the switch initiates the transmission of what is called a *Pause frame* (with a selected short period of time) on all of its input ports. On receipt of a Pause frame, the attached host must then stop sending any further frames to the switch until either a *defined time has expired* or it receives a notification from the switch that the overload condition has passed. Having sent a Pause frame, the switch monitors the level of memory in use and, when this falls below a second level, it sends out a *Continue frame* (pause frame with a zero time-to-wait value) on all input ports to inform the attached hosts that they can now resume sending new frames.

Pause frames are identified as MAC control frames by an *exclusive assigned (reserved) length/type value*. They are also assigned a reserved destination address value to ensure that an incoming pause frame is never forwarded to upper protocol layers or to other ports in a switch.

The full-duplex and flow control options are enabled on a link-by-link basis, assuming that the associated physical layers are also capable of supporting the options.



Network Configuration example

In order to obtain a high level of throughput, the two servers are connected directly to the switch by means of duplex 100 Mbps lines. All the end-user hosts then gain access to the servers through either a 10 BASE T or a 100 BASE TX hub. Both these types of hub operate in the half-duplex repeating mode. Hence the duplex uplink port connecting each hub to the switch has bridging circuitry within the hub to temporarily buffer all frame transfers to and from the switch and to perform the CSMA/CD MAC protocol associated with the shared medium hub ports.

Switching Bandwidth

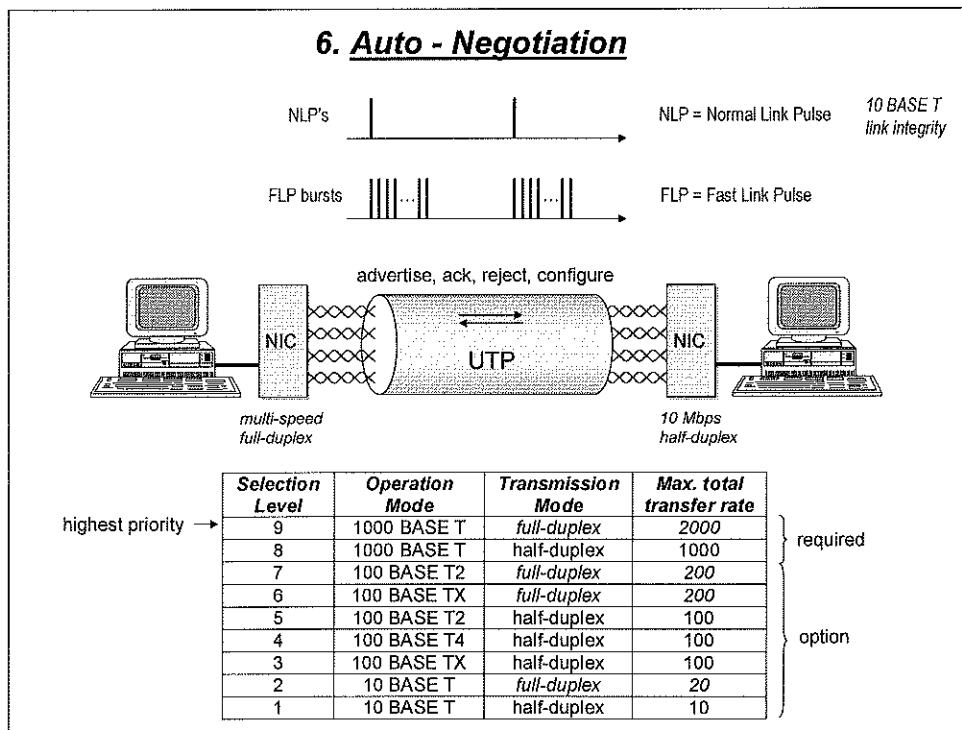
LAN switches can be characterized according to the proportion of bandwidth allocated to each port. Symmetric switching provides evenly distributed bandwidth to each port, while asymmetric switching provides unequal bandwidth between some ports.

An *asymmetric LAN switch* provides switched connections between ports of unequal bandwidths, such as a combination of 10 BASE T and 100 BASE T. This is also called 10/100 switching. Asymmetric switching is optimized for client-server traffic flows where multiple clients simultaneously communicate with a server, requiring more bandwidth dedicated to the server port to prevent a bottleneck at that port.

A *symmetric LAN switch* provides switched connections between ports with the same bandwidth, such as all 10 BASE T or all 100 BASE T. Symmetric switching is optimized for a reasonably distributed traffic load (peer-to-peer desktop environment).

Collision Domain

A collision domain is defined as a single CSMA/CD network in which there will be a collision if two hosts attached to the system both transmit at the same time. A LAN switch separates collision domains since they do not forward collision signals.



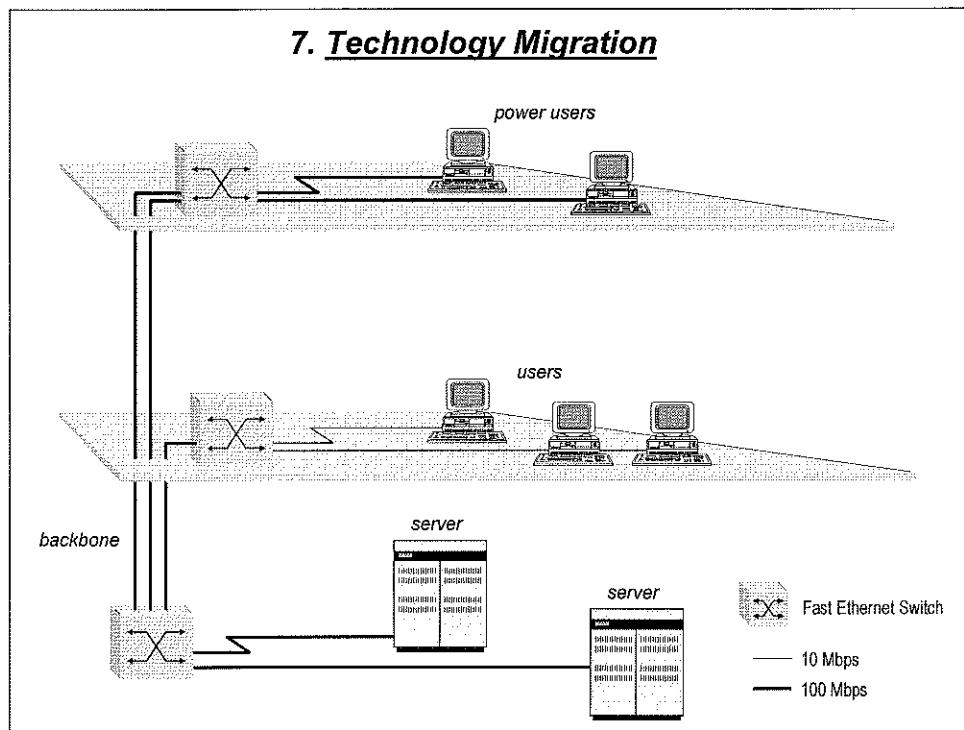
The purpose of auto negotiation is to find a way for two NICs that share a *UTP link* to communicate with each other, regardless of whether they both implemented the same Ethernet version or option set. It allows the NIC's to achieve the best possible mode of operation over the UTP link: automatic speed matching for multi-speed NIC's, automatic half-duplex/full-duplex mode matching.

Auto negotiation is performed totally within the physical layers *during link initiation*, without any additional overhead either to the MAC or to higher protocol layers. Auto negotiation allows UTP-based NICs to do the following:

- advertise their Ethernet version and optional capabilities to the other NIC
- acknowledge receipt and understand the operational modes that both NICs share
- reject any operational modes that are not shared
- configure each NIC for highest-level operational mode that both NICs can support

Auto negotiation is an option for 10 BASE T, 100 BASE TX, and 100 BASE T4, but it is required for 1000 BASE T implementations. The table shows the defined selection priority levels (highest level = top priority) for UTP-based Ethernet NIC's.

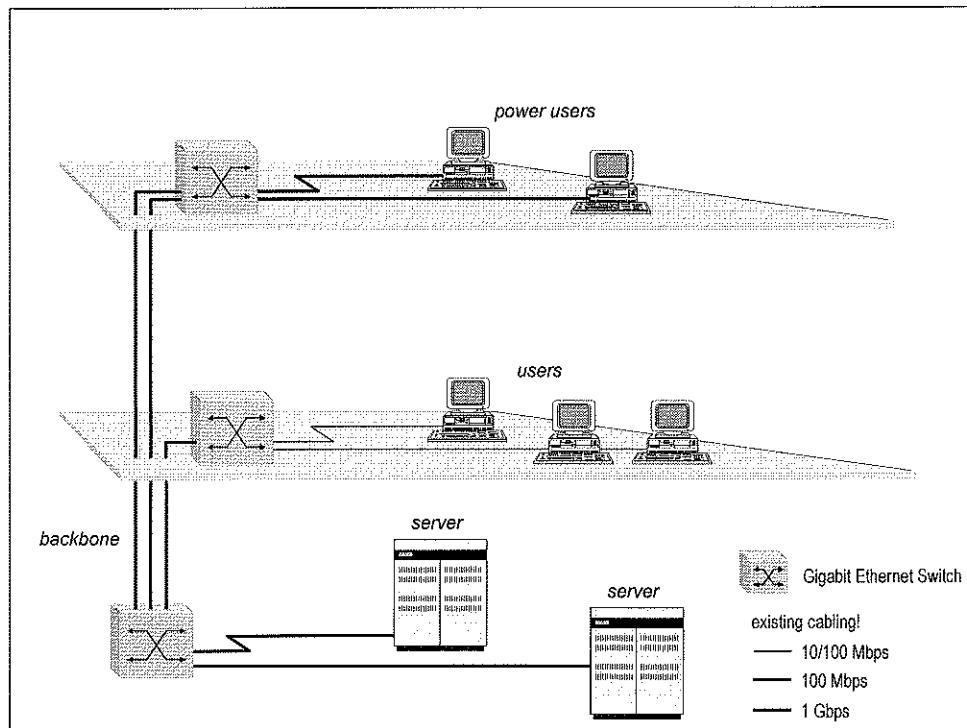
The auto negotiation function in UTP-based NIC's uses a modified 10 BASE T link integrity pulse sequence in which the Normal Link Pulses (NLP's) are replaced by bursts of Fast Link Pulses (FLP's), generated automatically at power-up. Each FLP burst is an alternating clock/data sequence in which the data bits in the burst identify the operational modes supported by the transmitting NIC and also provide information used by the auto negotiation handshake mechanism. If the NIC at the other end of the link is a compatible NIC but does not have auto negotiation capability, a parallel detection function still allows it to be recognized. A NIC that fails to respond to FLP bursts and returns only NLP's is treated as a 10 BASE T half-duplex NIC.



1000 BASE T allows a simple performance boost to support exploding bandwidth requirements on networks. 1000 BASE T is best suited for unclogging network bottlenecks that occur in three main areas:

- connection to high-speed servers
- workgroup aggregation
- desktop connections

In the network configuration shown, the initial building backbone is 10/100 Mbps Ethernet/Fast Ethernet. Several segments are aggregated in a 10/100 Mbps switch. Also several servers are connected to the switch with dedicated 10/100 Mbps connections. Some power users have dedicated 10/100 Mbps switched connections to their hosts.

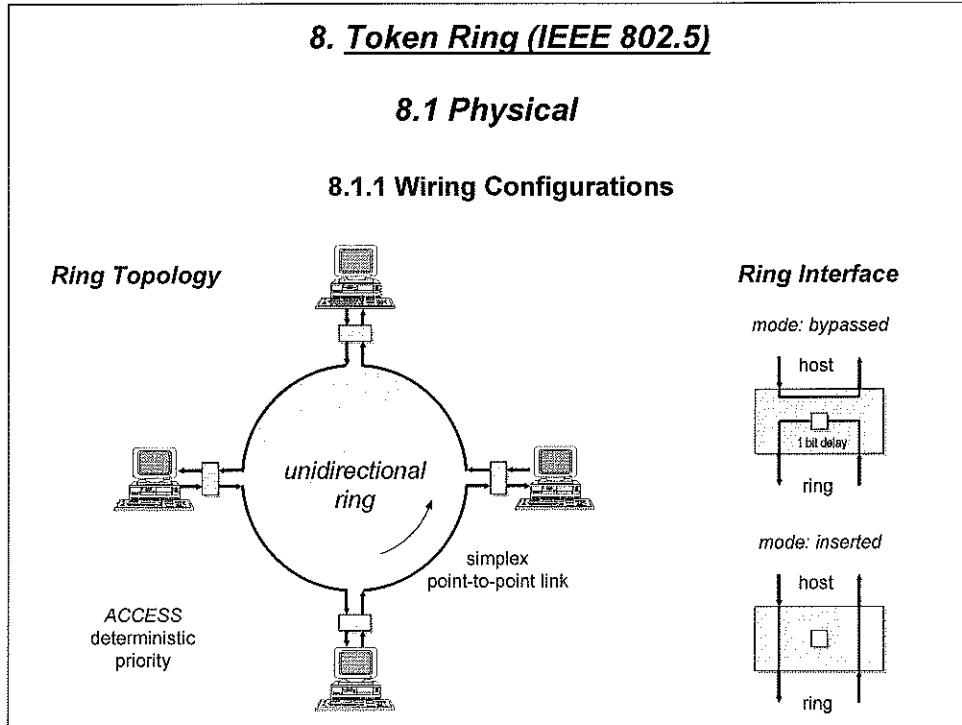


When users are starting to experience slow response times, an upgrade can be implemented in three areas:

- upgrading the backbone with a 1 Gbps Gigabit Ethernet Switch
- upgrading the workgroup switches that support power users or large workgroups with Gigabit Ethernet downlink modules
- implementing Gigabit Ethernet NICs in key servers

As a result of these measures, the speed of the backbone increases tenfold to accommodate the overall increase in network bandwidth demand, while the investment in existing workgroup switches, host NICs and existing cabling is preserved.

In a second phase the power user can be upgraded to Gigabit Ethernet NICs, giving them full access to the resources of the network.



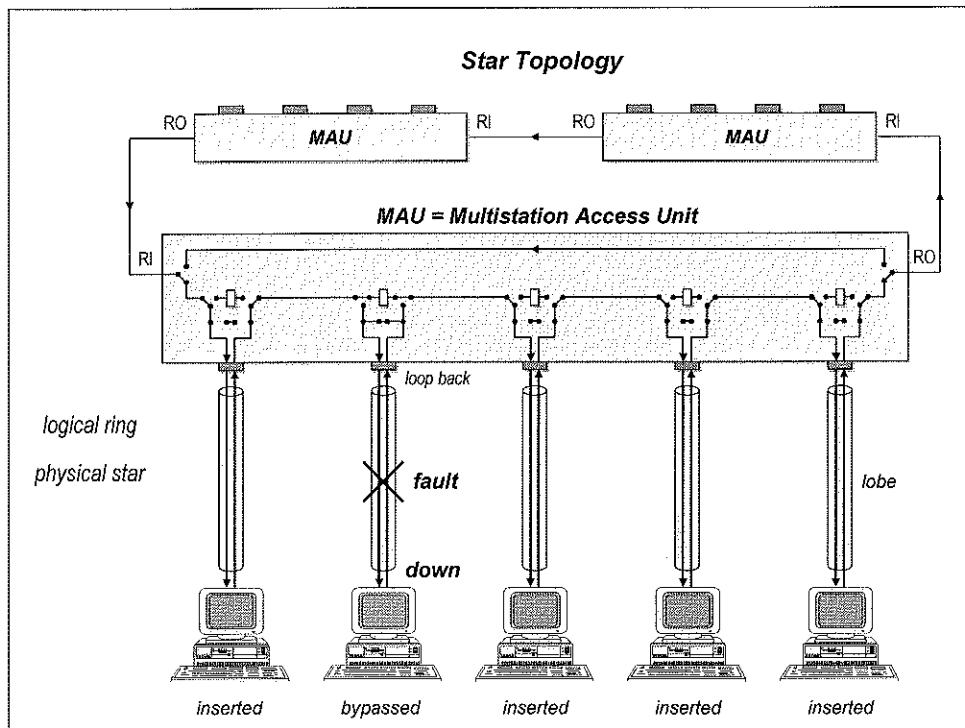
Ring networks are used for both local and wide area networks. Among their many attractive features is the fact that a ring is not really a broadcast medium, but a collection of individual point-to-point links that happen to form a circle. Point-to-point links involve a well-understood and field-proven technology and can run on twisted pair, coaxial cable, or fiber optics. Ring engineering is also almost entirely digital, whereas IEEE 802.3, for example, has a substantial analog component for collision detection. A ring is also fair and has a known upper bound on channel access.

With an Ethernet LAN the time to transmit a frame is non-deterministic since, during heavy load conditions when collisions are likely to be frequent, the transmission of a frame may be delayed and, in the limit, unsuccessful. In industrial environments such as manufacturing and process control, this is unacceptable. Hence although token ring LAN's also use a high bit rate shared/broadcast transmission medium, in order to provide a deterministic service, they utilize a completely different MAC method.

Also, frames can be assigned different priorities. The MAC method, therefore, also contains a priority control algorithm to ensure higher priority frames (for example those containing alarm messages) are transmitted before lower priority frames.

For these reasons, IBM chose the ring as its LAN and IEEE has included the *token ring* standard as IEEE 802.5.

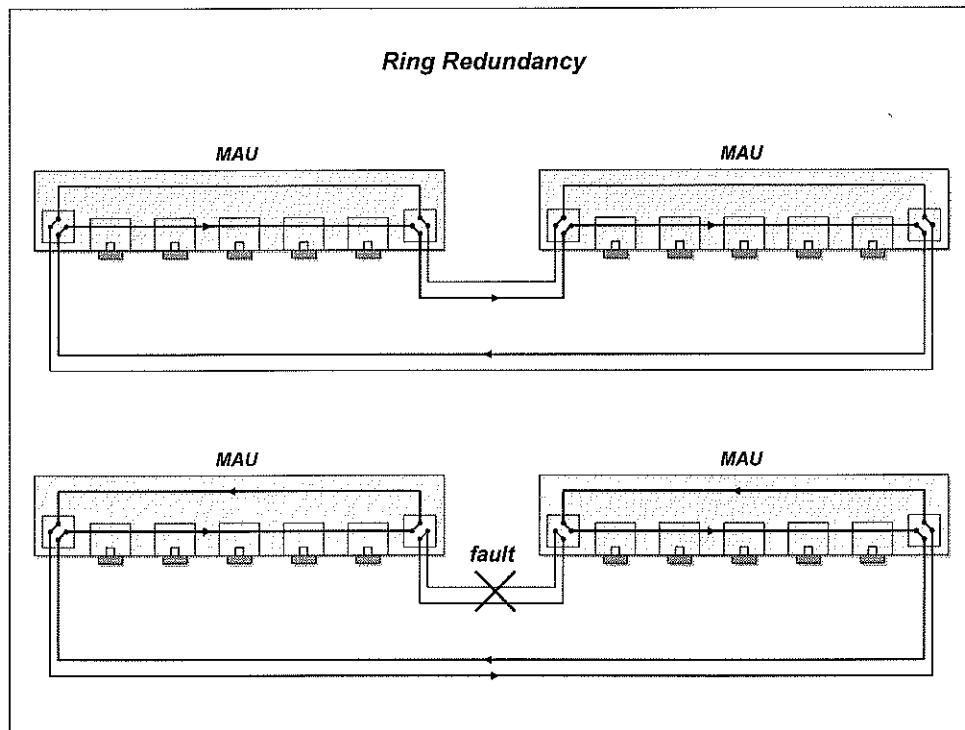
A ring really consists of a collection of *ring interfaces* connected by *unidirectional point-to-point links*. Each bit arriving at an interface is *copied* into a 1-bit buffer and then copied out onto the ring again. While in the buffer, the bit can be inspected and possibly modified before being written out. This copying step introduces a 1-bit delay at each interface.



Ring interfaces have two operating modes, listen and transmit. In *listen mode*, the input bits are simply copied to output, with a delay of 1 bit time. In *transmit mode*, which is entered only after the host has permission to transmit, the interface breaks the connection between input and output, entering its own data onto the ring. To be able to switch from listen to transmit mode in 1 bit time, the ring interface usually needs to buffer one or more frames itself rather than having to fetch them from the host on such short notice.

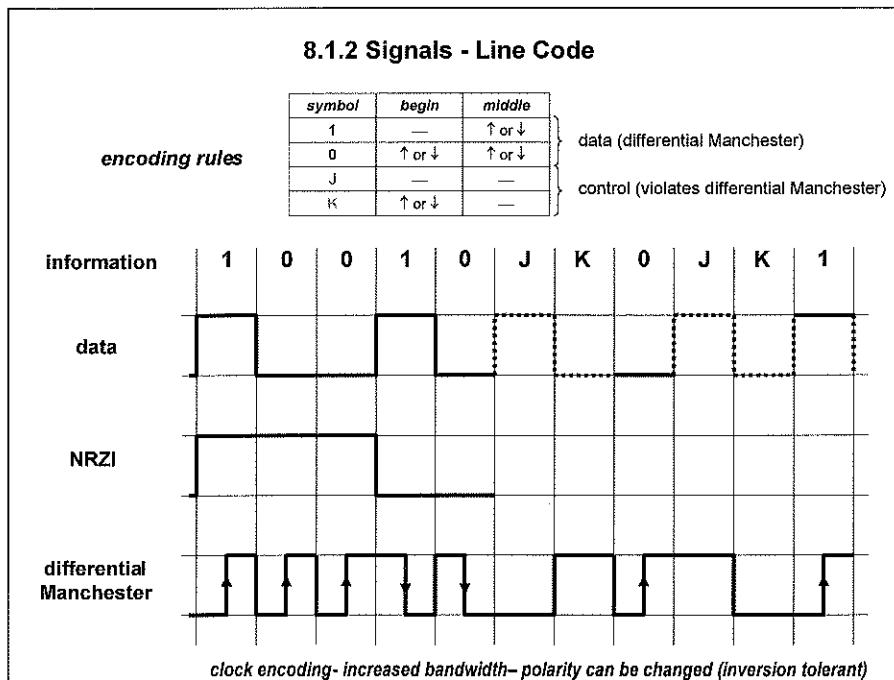
One problem with a ring network is that if the cable breaks somewhere, the ring dies. This problem can be solved by the use of a *MAU (Multistation Access Unit)* or hub/concentrator. While logically still a ring, physically each host is connected to the MAU by a cable containing (at least) two cable pairs (UTP, fiber), one for data to the host and one for data from the host.

Inside the MAU are bypass relays that are energized by current from the hosts. If the ring breaks or a host goes down, loss of the drive current will release the relay and bypass the host. The relays can also be operated by software to permit diagnostic programs to remove hosts one at a time to find faulty hosts and ring segments. The ring can then continue operation with the bad segment bypassed. Although the IEEE 802.5 standard does not formally require this kind of ring, often called a star-shaped ring, most IEEE 802.5 LAN's, in fact, do use MAU's to improve their reliability and maintainability. When a network consists of many clusters of hosts far apart, a topology with multiple MAU's can be used. Just imagine that the cable to one of the hosts were replaced by a cable to a distant MAU. Although logically all the hosts are on the same ring, the wiring requirements are greatly reduced.



Most token ring networks are installed with redundant main rings. As illustrated in the figure, each MAU is connected through both the RI and RO ports. Each main ring connection contains two pairs: one which is used as the primary data path and a second which serves as a back-up pair. When the ring is functioning correctly, data travels only on the main pair. In the figure the arrows indicate how data travels on only the main pair.

In the case of a cable failure on the main ring path, some MAU's will automatically cause the main ring to wrap around and use the backup pair, preserving ring integrity. As illustrated in the figure, only one pair is continuous, and the data path is self-looped at the RI/RO ports. The arrows indicate how the data travels on both the primary and the back-up pair.



Differential Manchester linecode

This is a code in which a binary 1 is represented by a mid-interval voltage transition in the opposite direction from the previous bit. A binary 0 is represented by a mid-interval voltage transition in the same direction as the previous bit.

The differential Manchester line code can be read correctly by inverting the direction.

It guarantees one transition (rising or descending edge) per clock pulse, which is equivalent to transporting a synchronization signal. The DC component is always constant and may therefore be chosen to be zero. The signal spectrum is bunched between $1/2T_b$ and $1/T_b$ with zero power at frequency zero, but it has a total width twice that of the NRZ (Non Return to Zero). The code has an efficiency of 50% since two levels are used to encode one bit of data.

High and low levels are positive and negative signals of absolute magnitude 3 to 4.5 V.

Token ring also uses a violation of the differential Manchester encoding to transmit the two unique control symbols (J and K) used to distinguish the Start Delimiter (SD) and Ending Delimiter (ED) fields in tokens and frames.

Unlike binary data which always provides a voltage transition in the middle of a bit-time, the J and K symbols do not change voltage for an entire bit-time. A J symbol is represented by continuing the same voltage as the previous bit. A K symbol is represented by the opposite voltage of the last bit.

8.1.3 Signals - Medium				
Medium	Host-to-MAU	MAU-to-MAU	Maximum number of repeaters	
UTP	100 m	120 m	72	<i>medium general guidelines</i>
STP	100 m	200 m	250	
fiber	-	2000 m	-	

Data Rate	Line Code	max Frame Size	Access Control	
4 Mbps	differential Manchester	4550 bytes	TP or DTR	
16 Mbps	differential Manchester	18.200 bytes	TP or DTR	TP = Token Passing
100 Mbps	MLT-3 or 4B5B/NRZI	18.200 bytes	DTR	DTR = Dedicated Token Ring (MAU = switch)

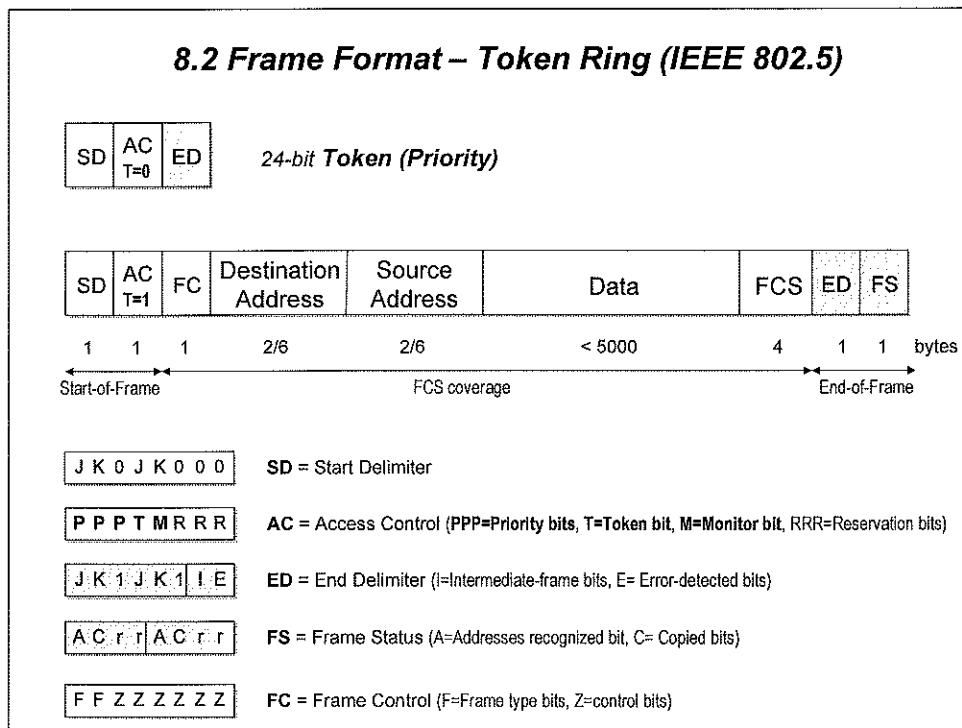
Medium

Unlike the relatively self-explanatory cabling guidelines for Ethernet 802.3 networks, token ring guidelines are more flexible and dependent on a variety of factors. Recommended cabling distances and numbers of stations per ring vary considerably from vendor to vendor, and the most reliable sources are the specifications and installation guides provided by manufacturers for their products. These typically include charts that give allowable cable lengths for type of media, number of MAU's, number of stations, and number of wiring closets in use.

The table lists general guidelines for the maximum size of token ring networks. However, actual limitations for specific networks and products may vary greatly and manufacturers of specific token ring MAU's may guarantee better performance.

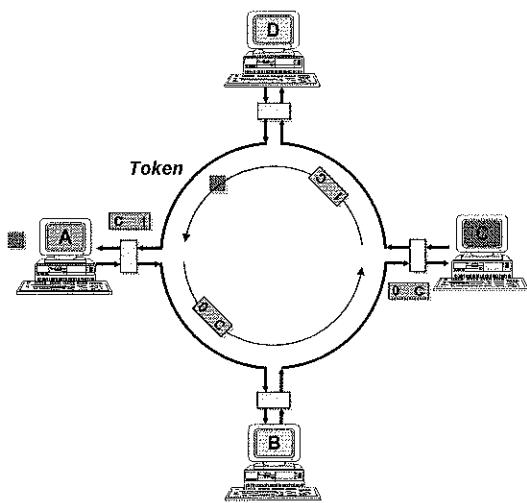
Dedicated Token Ring

The 1997 update to IEEE 802.5 introduced a new access control technique known as dedicated token ring (DTR). A ring can be configured in a star topology by use of a MAU (hub or concentrator). The token-passing (TP) algorithm can still be used so that the ring capacity is still shared and access control is determined by the token. However, it is also possible to have the central hub function as a *switch*, so that the connection between each station and the switch functions as a full-duplex point-to-point link. The DTR specification defines the use of stations and concentrators in this switched mode. The DTR concentrator acts as a frame-level relay rather than a bit-level repeater, so that each link from concentrator to station is a dedicated link with immediate access possible; token passing is not used.



- **Starting delimiter (SD).** Indicates start of frame. The SD consists of signaling patterns that are distinguishable from data. It is coded as follows: JK0JK000, where J and K are nondata symbols. The actual form of a nondata symbol depends on the signal encoding on the medium.
 - **Access control (AC).** Has the format PPPTMRRR, where PPP and RRR are 3-bit priority and reservation variables, and M is the monitor bit. T indicates whether this is a token or data frame. In the case of a token frame, the only remaining field is ED.
 - **Frame control (FC).** Indicates whether this is an LLC data frame. If not, bits in this field control operation of the token ring MAC protocol.
 - **End delimiter (ED).** Contains the error-detection bit (E), which is set if any repeater detects an error, and the intermediate bit (1), which is used to indicate that this is a frame other than the final one of a multiple-frame transmission.
 - **Frame status (FS).** Contains the address recognized (A) and frame-copied (C) bits, whose use is explained below. Because the A and C bits are outside the scope of the FCS, they are duplicated to provide a redundancy check to detect erroneous settings. If a host detects its own MAC address, it sets the A bit to 1; it may also copy the frame, setting the C bit to 1. This allows the originating host to differentiate three results of a frame transmission:
 - Destination host nonexistent or not active (A = 0, C = 0)
 - Destination host exists but frame not copied (A = 1, C = 0)
 - Frame received (A = 1, C = 1)

8.3 Medium Access Control (MAC): Token Passing



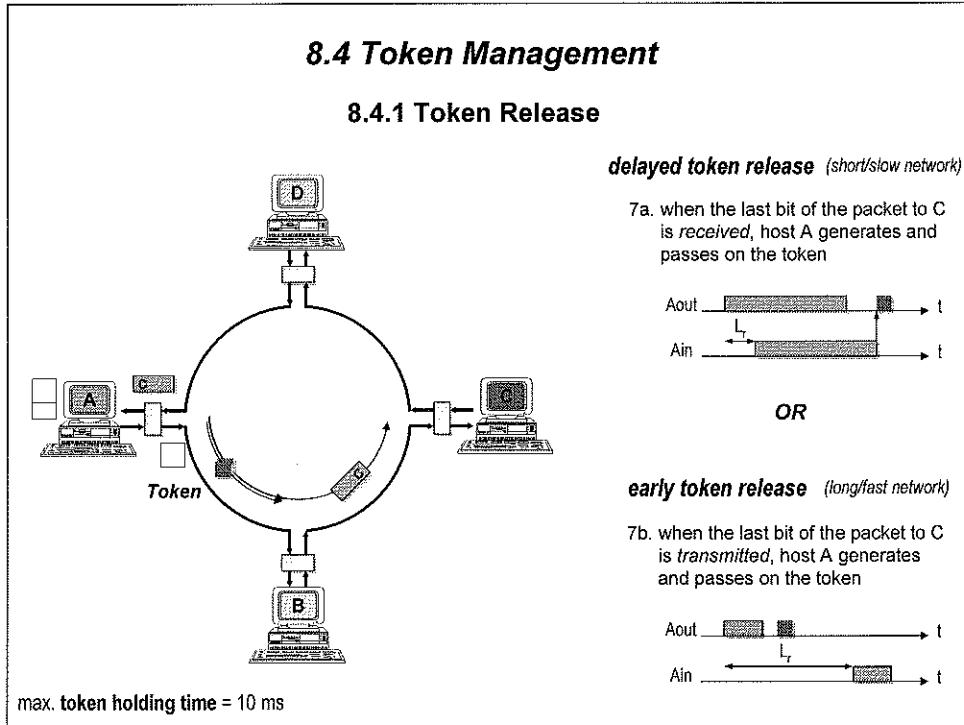
1. host A waits for the token from its upstream neighbor D
2. host A seizes the token ($T \rightarrow 1$)
3. host A sends a packet to C by passing it to its downstream neighbor B
host B repeats the packet
4. host C copies packet addressed to it
5. host C sets Frame Status ($A=C=1$)
packet continues around the ring
host D repeats the packet
6. host A receives packet for C and removes it from the ring

The token ring technique is based on the use of a small frame, called a *token*, that circulates when all hosts are idle. A host wishing to transmit must wait until it detects a token passing by, as indicated by a token bit $T=0$ in the AC field. The host *seizes the token* by setting the token bit to 1. This changing of the T bit in the token, transforms it from a token into a start-of-frame sequence for a data frame (the SD and AC fields of the received token function are the first two fields of the outgoing frame). The host then appends the address of the intended recipient at its head and transmits the remainder of the fields needed to construct a data frame.

The frame is *repeated* (that is, each bit is received and then retransmitted) by all the hosts in the ring until it circulates back to the initiating host, where it is *removed*. In addition to repeating the frame, each host reads and stores the frame contents. The intended recipient - indicated by the destination address in the frame header - retains a copy of the frame and indicates that it has done this by setting the Frame Status (A,C) bits at the tail of the frame.

The frame on the ring will make a round trip. Under normal conditions, the first bit of the frame will go around the ring and return to the sender before the full frame has been transmitted. Only a very long ring will be able to hold even a short frame. Consequently, the transmitting host must drain the ring while it continues to transmit. This means that the bits that have completed the trip around the ring come back to the sender and are removed.

When a host seizes a token and begins to transmit a data frame, there is no token on the ring, so other hosts wishing to transmit must wait. The transmitting host will insert a new token on the ring. Once the new token has been inserted on the ring, the next host down-stream with data to send will be able to seize the token and transmit.



The host transmits one or more frames, continuing until either its supply of frames is exhausted or a token-holding timer expires. When the AC field of the last transmitted frame returns, the host sets the token bit to 0 and appends an ED field, resulting in the insertion of a new token on the ring.

A host may hold the token for the *token-holding time*, which is 10 ms unless an installation sets a different value. If there is enough time left after the first frame has been transmitted to send more frames, these may be sent as well. After all pending frames have been transmitted or the transmission of another frame would exceed the token-holding time, the host regenerates the 3-byte token frame and puts it out onto the ring.

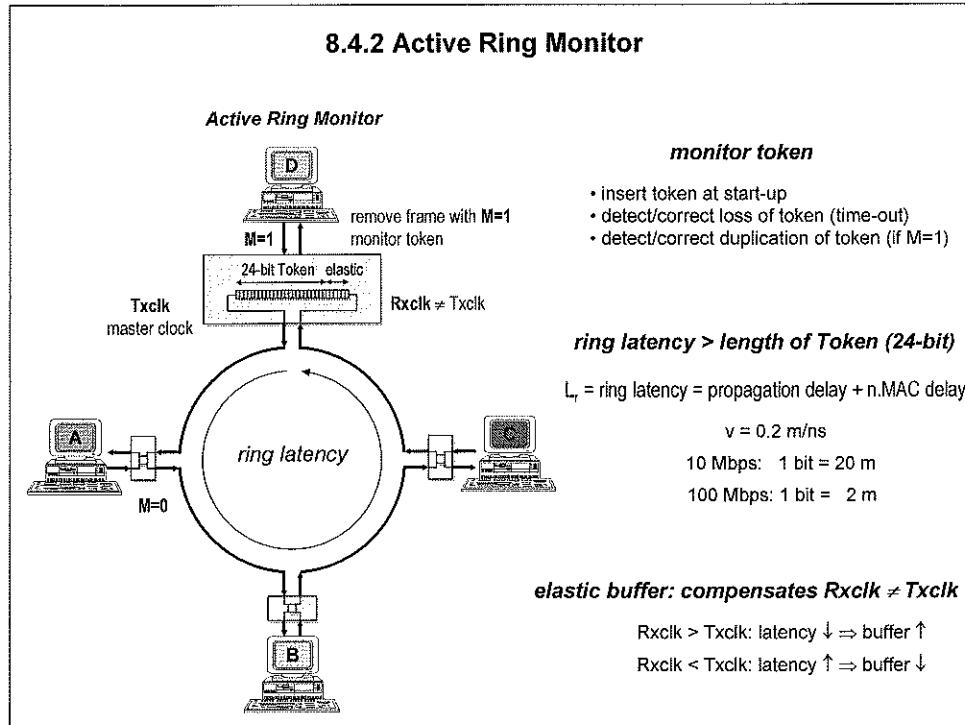
Short (slow) network:

When a station issues a frame, if the bit length of the ring is less than that of the frame, the leading edge of the transmitted frame will return to the transmitting station before it has completed transmission; in this case, the station may issue a token as soon as it has finished frame transmission.

Long (fast) network:

If the frame is shorter than the bit length of the ring, then after a station has completed transmission of a frame, it must wait until the leading edge of the frame returns before issuing a token. In this latter case, some of the potential capacity of the ring is unused. *ETR (early token release)* allows a transmitting station to release a token as soon as it completes frame transmission, whether or not the frame header has returned to the station.

Stations that implement ETR are compatible and interoperable with those that do not.

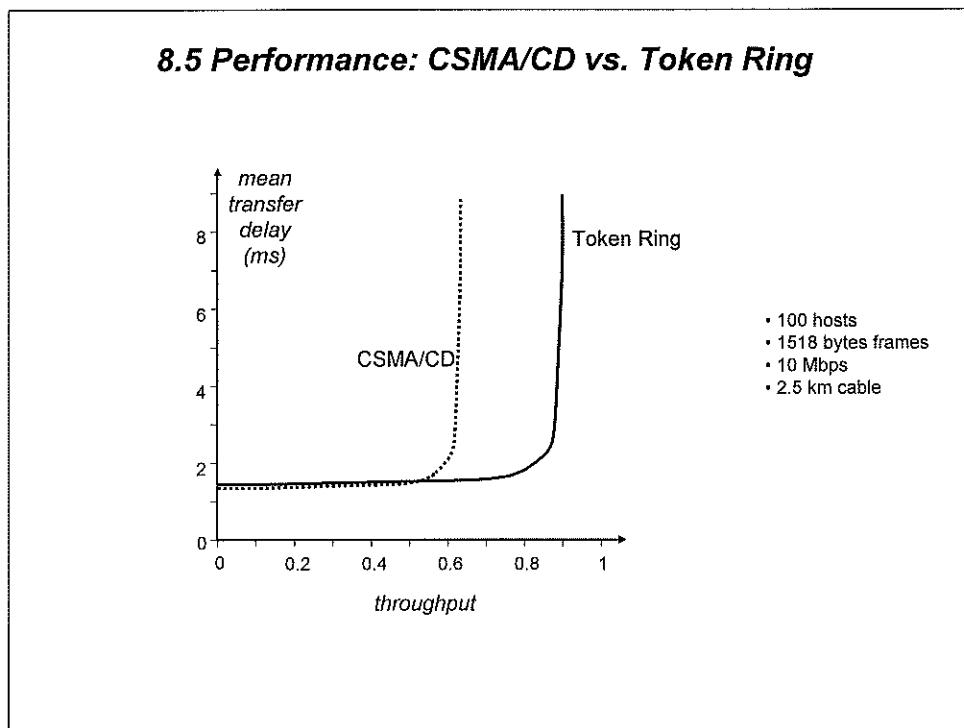


Associated with the ring is a single master station known as the **active ring monitor** that supplies the master clock for the ring. All stations are capable of performing this function and the active monitor is selected using a bidding process involving all the stations that are currently inserted into the ring. All other inserted stations in the ring frequency and phase lock their own clock using a DPLL and the incoming bit stream.

The active ring monitor ensures the ring has a *minimum latency time*. This is the time, measured in bit times at the ring data transmission rate, taken for a signal to propagate once around the ring. The ring latency time includes the signal propagation delay through the ring transmission medium together with the sum of the delays through each MAC unit. For the control token to circulate continuously around the ring when none of the stations requires to use the ring (that is, all stations are simply in the repeat mode), the ring must have a minimum latency time of at least the number of bits in the token sequence to ensure that the token is not corrupted. The token is 24 bits long, so the active ring monitor provides a fixed 24-bit buffer, which effectively becomes part of the ring to ensure its correct operation under all conditions.

To maintain a constant ring latency, an additional *elastic (variable) buffer* with a length of six bits is added to the fixed 24-bit buffer. The resulting 30-bit buffer is initialized to 27 bits. If the received signal at the master MAC unit is faster than the master oscillator, the buffer is expanded by a single bit. Alternatively, if the received signal is slower, the buffer is reduced by a single bit. In this way the ring always comprises sufficient bits to allow the token to circulate continuously around the ring in the quiescent (idle) state.

Token maintenance is necessary. Loss of the token prevents further utilization of the ring. Duplication of the token can also disrupt ring operation. One host must be selected as a monitor to ensure that exactly one token is on the ring and to ensure that a free token is reinserted, if necessary.



Monitor

An orphaned frame is one that was transmitted correctly on the ring but whose 'parent' died; that is, the sending station went down before it could remove the frame from the ring. These are detected using the Monitor bit M. This is 0 on transmission and set to 1 the first time the packet passes the monitor. If the monitor sees a packet with this bit set, it knows the packet is going by for the second time and it drains the packet off the ring.

Performance

When *traffic is light*, the token will spend most of its time idly circulating around the ring. Occasionally a host will seize it, transmit a frame, and then output a new token. There is some inefficiency with token ring because a host must wait for the token to come around before transmitting.

However, when the *traffic is heavy*, so that there is a queue at each host, as soon as a host finishes its transmission and regenerates the token, the next host downstream will see and remove the token. In this manner the permission to send rotates smoothly around the ring, in round-robin fashion. The network efficiency can begin to approach 100 percent under conditions of heavy load. The ring functions in a round-robin fashion, which is both efficient and fair. After host A transmits, it releases a token. The first host with an opportunity to transmit is B. If B transmits, it then releases a token and C has the next opportunity, and so on.

In the cases where the load is significant, the token passing access method is superior compared to CSMA/CD.



1. Internetworking Terms

Subnetwork : constituent homogeneous network

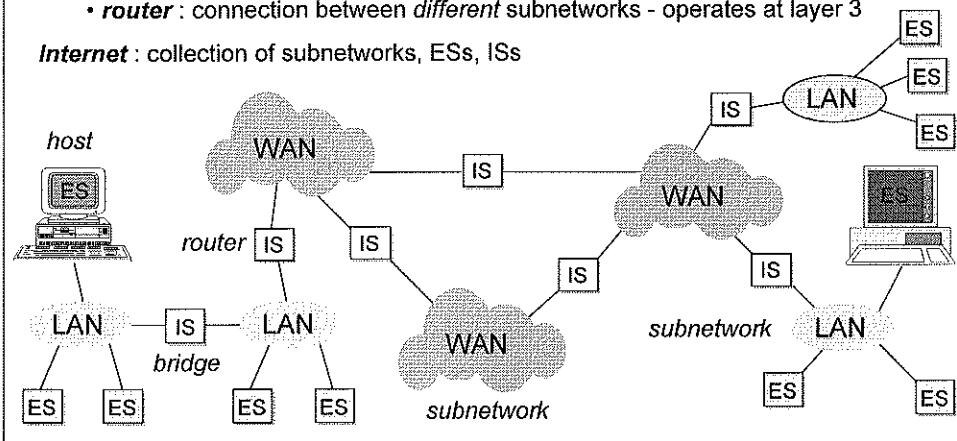
- **WAN** = Wide Area Network
- **LAN** = Local Area Network

ES = End System : connected to subnetwork, support end-user applications/services

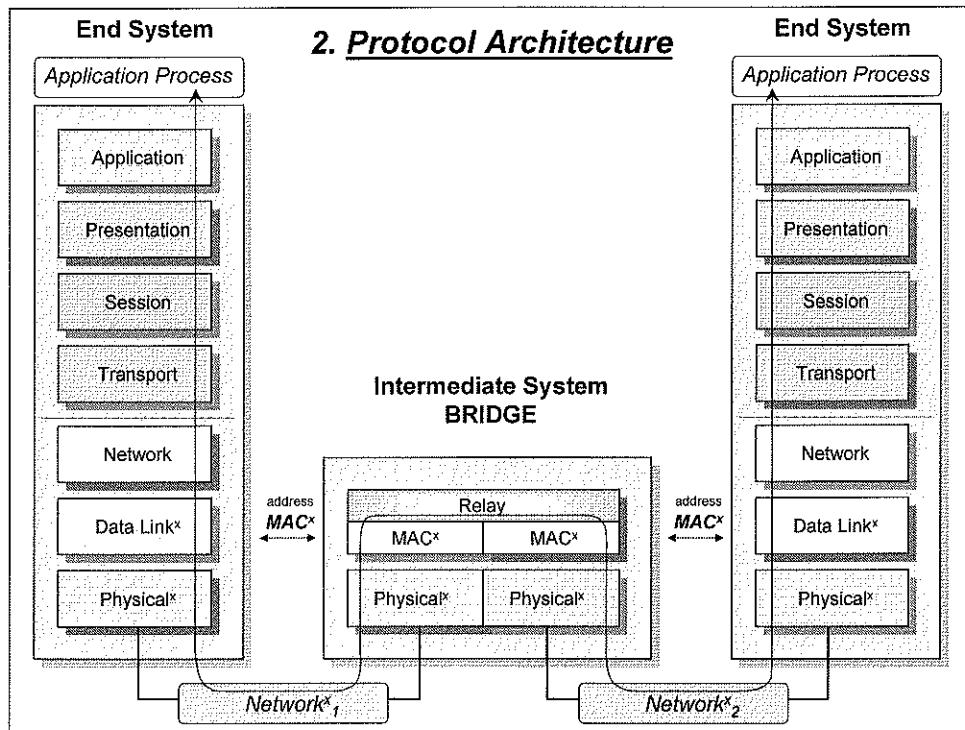
IS = Intermediate System : connected between networks, support communication

- **bridge** : connection between *like* subnetworks - operates at layer 2
- **router** : connection between *different* subnetworks - operates at layer 3

Internet : collection of subnetworks, ESs, ISs



Internet: *catenet model* = a concatenation of networks



Bridge

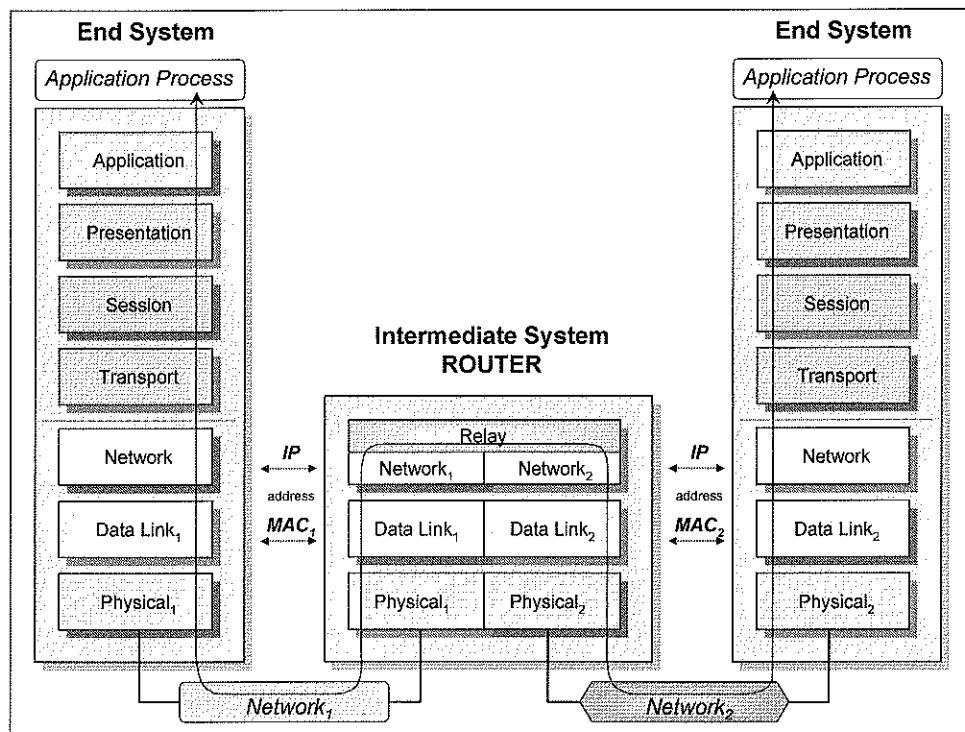
The most basic form of *transparent bridge* is one that attaches to two or more LANs (each attachment to a LAN is known as a port), listening to every MAC-frame transmitted and storing each received frame until it can be transmitted on the LANs other than the one on which it was received.

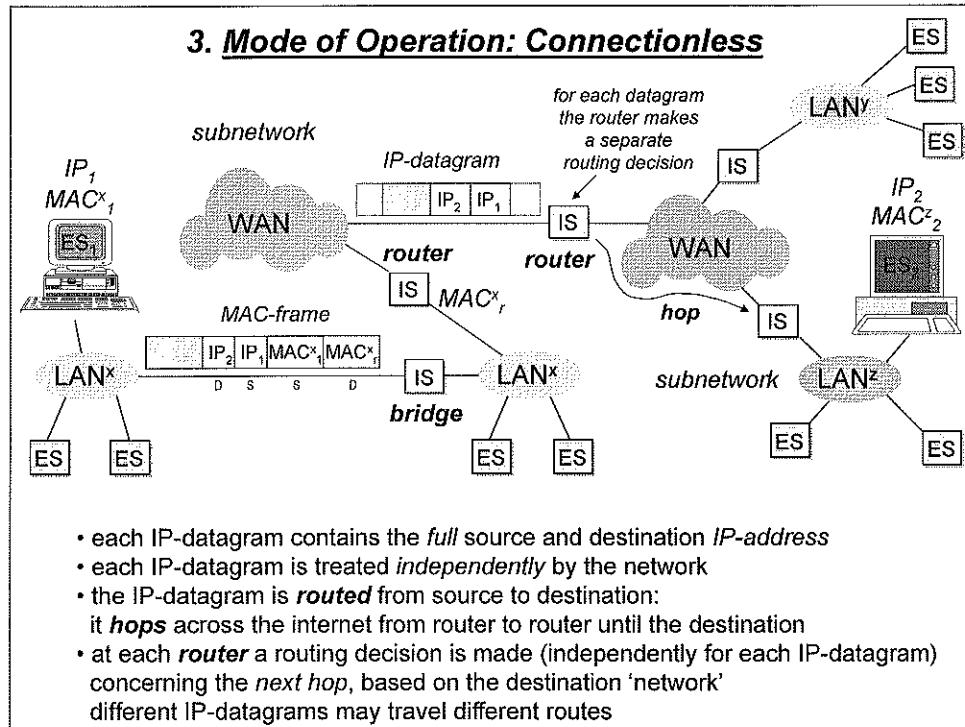
The bridge makes *no modification* to the content or format of the MAC-frames it receives, nor does it encapsulate them with an additional header . Each frame to be transferred is simply *copied* from one LAN and repeated with exactly the same bit pattern as the other LAN. (The two LANs use the same LAN protocol.)

The bridge should contain enough *buffer space* to meet peak demands. Over a short period of time, frames can arrive faster than they can be transmitted (because the other LAN is busy).

The bridge does an *interpretation of the MAC addresses*. At a minimum, the bridge must know which addresses are on each network in order to know which frames to pass.

The bridge provides an *extension to the LAN* that requires no modification to the communications software in the stations attached to the LANs. It appears to all stations on the two (or more) LANs that there is a single LAN on which each station has a unique address. The station uses that unique address and need not explicitly discriminate between stations on the same LAN and stations on other LANs; the bridge takes care of that.





4. IP = Internet Protocol (Connectionless)

- ***Connectionless***

Each packet is treated *independently* from all others. A sequence of packets (datagrams) sent from one host to another may travel over different paths, or may be lost while others are delivered.

There is *no reservation of resources* in advance of sending a datagram.

- ***Unreliable***

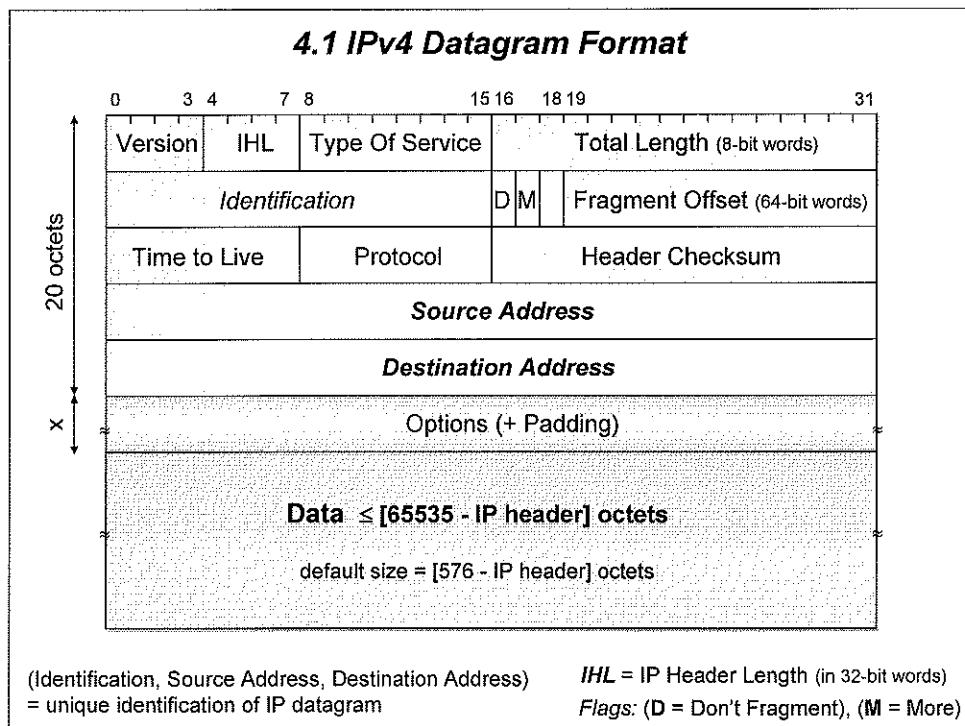
Delivery is *not guaranteed*. The packet may be lost, duplicated, delayed, delivered out of order, but the IP service will not detect such conditions, nor will it inform the sender or receiver (no acknowledgment).

- ***Best-effort delivery***

The IP layer makes an earnest attempt to deliver packets. Unreliability arises only when *routers are overloaded* (congested) or underlying *networks fail*.

- ***Routing***

Based on the IP address the routers select a path that the IP datagram will follow through the network. Routing is based on a '*destination network*', not on a destination host.



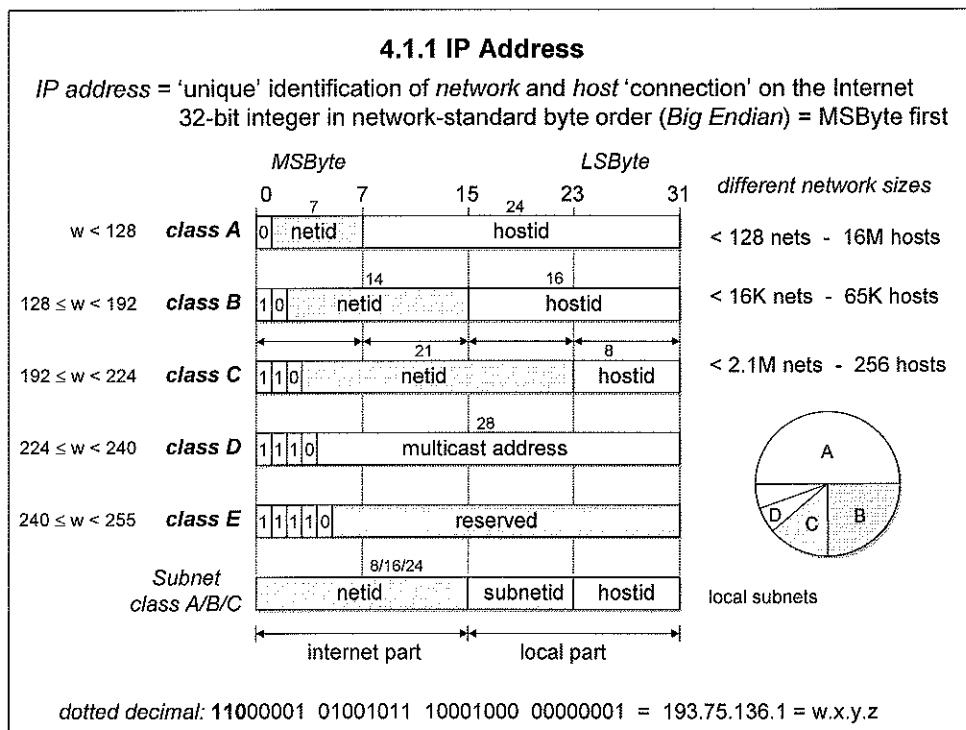
Mostly datagram processing occurs in software, than the contents and format are not constrained by any hardware.

Version (IP version4), the first 4-bit field in the datagram, is used to verify that the sender, receiver, and any routers in between them agree on the format of the datagram. This first bits of the IP datagram define the format of the header bits that will follow. If standards change, hosts will reject datagrams with protocol versions that differ from theirs, preventing them from misinterpreting datagram contents according to an outdated format. The current IP protocol version is 4, but a version 6 is defined.

IHL, the *IP Header Length* field, also 4 bits, gives the datagram header length measured in 32-bit words. IHL is needed because the header is of variable length. All fields in the header have fixed length except for the IP *Options* and corresponding *Padding* fields. Padding (= insert 0's) is used, if necessary, to extend the header to a multiple of 32 bits. The most common header, which contains no options and no padding, measures 20 octets and has a IHL field of 5. The maximum header length is $(2^4-1=15) \times 4\text{bytes} = 60$ bytes.

The *Total Length* field gives the length of the IP datagram measured in octets, including octets in the header and data. The size of the data area can be computed by subtracting the length of the header (IHL) from the Total Length. Because the Total Length field is 16 bits long, the maximum possible size of an IP datagram is $2^{16}-1$ or 65.535 octets. In most applications this is not a severe limitation. It may become more important in the future when higher speed networks can carry data packets larger than 65.535 octets. The default IP datagram length is 576. It is required that all hosts and routers service IP datagrams up to 576 octets in length without any need for fragmentation.

To give room to fragmentation flags (D,M) the *Fragmentation Offset* is reduced with 3 bits and has a maximum value of $2^{13}-1 = 8191$ 8-byte words = 65528 octets.



IP address

Each host 'connected' on a TCP/IP internet is assigned a unique 32-bit IP address, that is used in all communication with that host connection.

NIC = Network Information Center = addressing authority: assigns unique netid

Three different address formats (classes) are defined to allow for the different sizes of network to which a host may be attached. The *three primary classes* are A, B, and C; each is intended for use with a different size of network. The address class can be determined from the position of the first zero bit in the first four bits. The remaining bits specify two subfields - a *network identifier (netid)* and a *host identifier (hostid)*. The subfield boundaries are located on byte boundaries to simplify decoding.

IP address = (classid+netid,hostid)

The *netid* identifies a network, the *hostid* identifies a host on that network (hierarchy).

Routers base routing on the netid, this makes routing efficient.

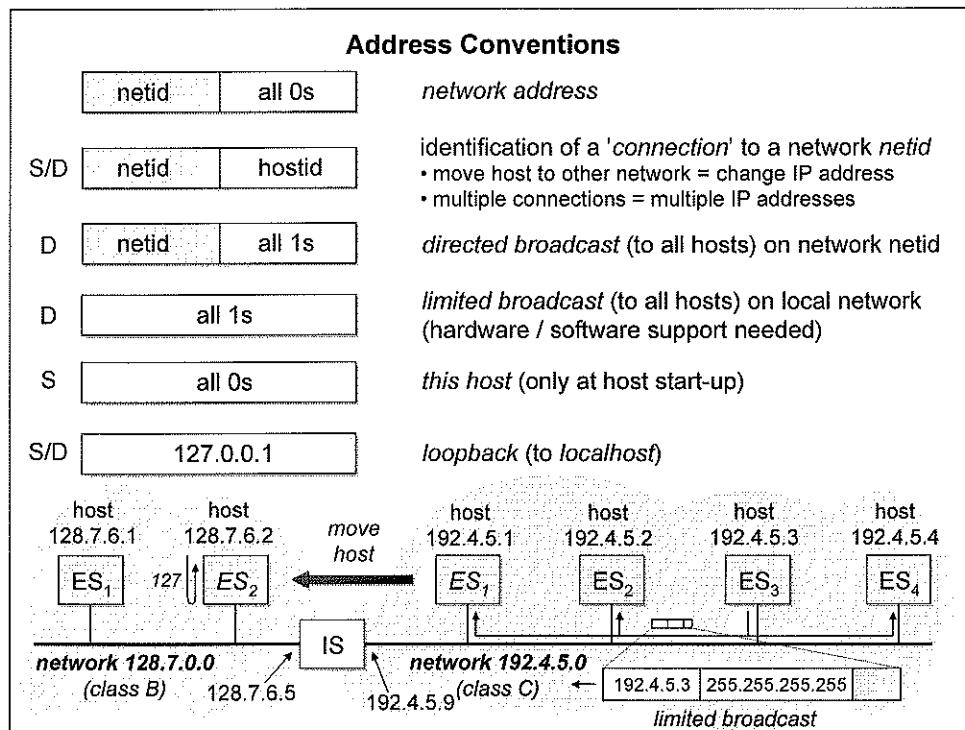
Class D: IP-Multicast is a form of multi-point delivery. Unlike broadcasting, multicasting allows each host to choose whether it wants to participate in a multicast. It allows transmission of an IP datagram to a set of hosts (spread across separate physical networks) that form a multicast group. Membership is dynamic: a host may *join / leave a multicast group* at any time with the IGMP (Internet Group Management Protocol).

Dotted Decimal Notation

When communicated in technical documents or through application programs, IP addresses are written as four decimal integers separated by decimal points (w.x.y.z), where each integer gives the value of one octet (0 .. 255) of the 32-bit IP address.

Big Endian

The *internet standard for byte order* specifies that integers are sent most significant byte first: *Big Endian* style (\leftrightarrow Little Endian).



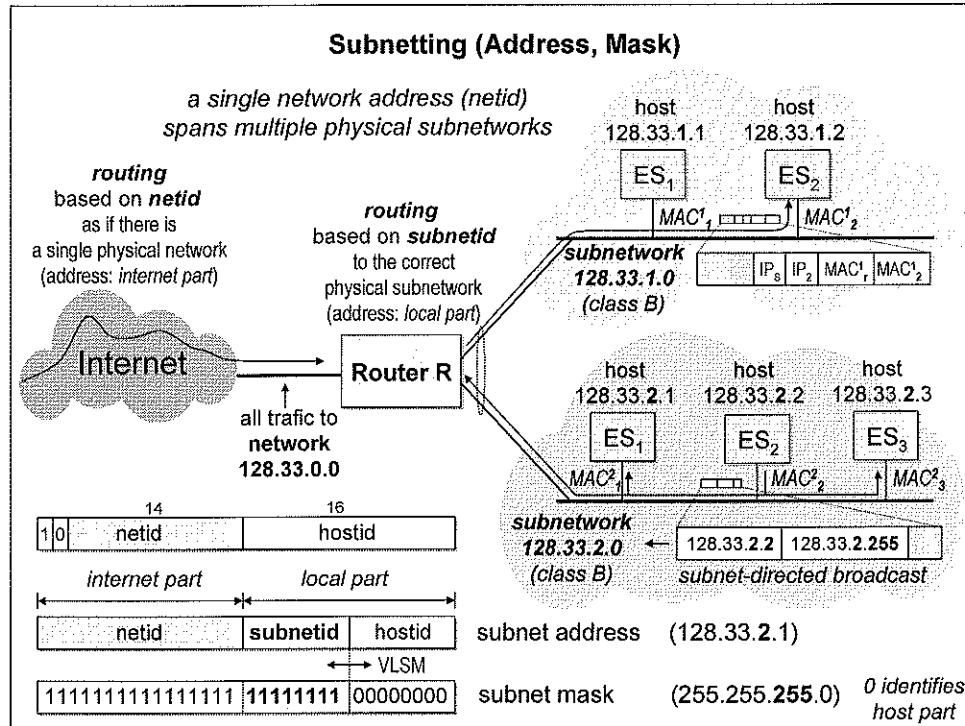
An IP address encode both a network and a host on that network, *it does not specify an individual machine*, but a 'connection' to a network. *Routers and multi-homed hosts use an IP address per connection to a physical network*. If a host moves from one network to another, its IP address must change.

IP addresses can be used to refer to networks as well as individual hosts. By convention, the *network address has hostid with all bits 0*. Hostid 0 is never a host.

A *directed broadcast address* (only destination!) refers to all hosts on a specified physical network, uniquely identified by its *netid*. A broadcast address has hostid with all bits 1. On many LAN network technologies (e.g., Ethernet) broadcasting can be as efficient as normal transmission (IP broadcast maps on Ethernet hardware broadcast). Some networks like WAN's do not support broadcast at all (telephone network). Thus, having an IP broadcast address does not guarantee the availability or efficiency of broadcast delivery. Directed broadcast addresses provide a powerful (and somewhat dangerous) mechanism that allows a remote system to send a single packet that will be broadcast on the specified network. Disadvantage of directed broadcast is that it requires knowledge of the network address.

A *limited broadcast address* or *local network broadcast address* (only destination!), provides a broadcast address for the local network independent of the assigned IP address. The local broadcast address consists of thirty-two 1s (hence, it is sometimes called the "all 1s" broadcast address). A host may use the limited broadcast address as part of a start-up procedure before it learns its IP address or the IP address for the local network. After startup the host should use directed broadcast. As a general rule, TCP/IP protocols restrict broadcasting to the smallest possible set of hosts.

Most TCP/IP implementations support a *loopback interface* that allows a client and server on the same host to communicate through TCP/IP. The class A address 127 is reserved. Most systems assign the IP address 127.0.0.1 and the name *localhost*. An IP datagram sent to the loopback interface must not appear on any network.



Subnet addressing, subnet routing, or subnetting allows a single network address to span multiple physical networks. It hides the internal network organization to external routers. Subnetting is now a required part of IP addressing.

In the shown example a site has a single class B IP network address 128.33.0.0 assigned to it, but it has two physical networks. Only the *local router R* knows that there are multiple physical nets and how to route traffic among them; routers in other autonomous systems route all traffic as if there were a single physical network. Once a packet reaches R, it must be sent across the correct physical network to its destination. The local site uses the third octet of the address to distinguish between the two networks. The manager assigns hosts on one physical net addresses of the form 128.33.1.X, and hosts on the other et addresses of the form 128.33.2.X, where X represents an integer used to identify a specific host. R examines the third octet of the destination address and routes datagrams with value 1 to the network labeled 128.33.1.0 and those with value 2 to the network labelled 128.33.2.0.

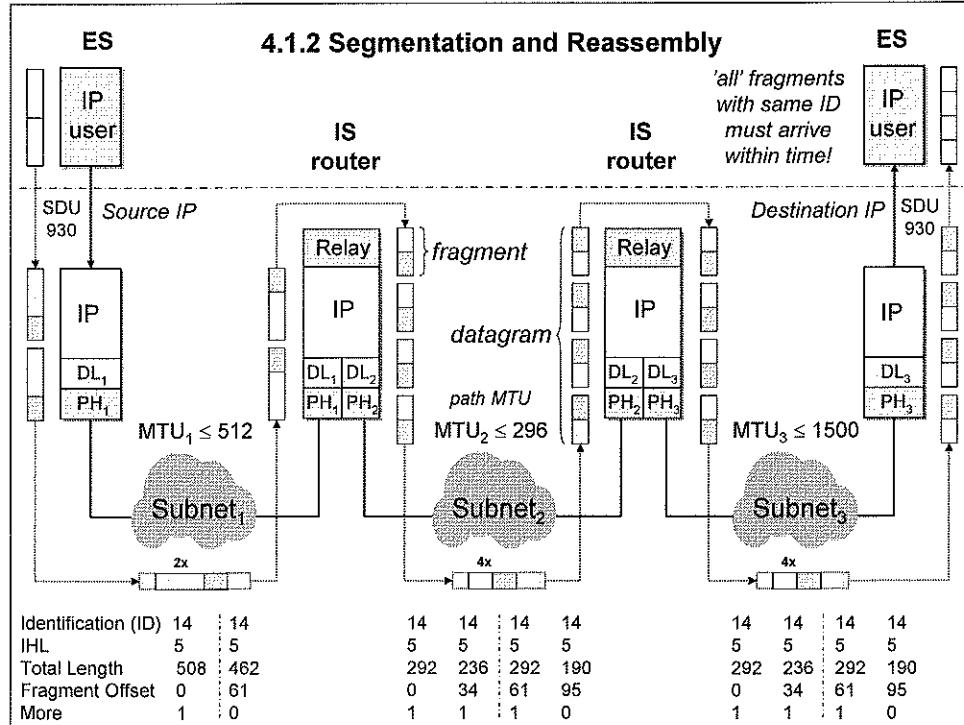
Instead of dividing the 32-bit IP address into a networdid and a hostid, subnetting divides the address into a *internet part* and a *local part*. The interpretation of the *internet part* remains the same as for networks that do not use subnetting. The interpretation of the *local part* as an identification of a host on a physical network is left up to the site. The result is a form of *hierarchical addressing* that leads to corresponding *hierarchical routing*. The top level of the routing hierarchy (i.e., other autonomous systems in the internet) uses the *internet part* when routing, the next level (e.g., the local site) uses the additional *subnetid*. Finally, the lowest level (i.e., delivery across one physical network) uses the entire address (IP to MAC).

The TCP/IP subnet standard is flexible: the local part partitioning can be chosen independently for each physical network (VLSM=Variable Length Subnet Mask)

ex: 16 bit local part = 8 bit subnetid + 8 bit hostid or 3 bit subnetid + 13 bit hostid or ..

The subnet is used by the host to detect if the destination is on the same subnet. If positive, the packet can be transmitted directly on the subnet. If negative, the host transmits the packet to router connected to the subnet.

The *subnet-directed broadcast address* has hostid all 1's and the specific subnetid.



Transmit IP-datagram with 930 bytes data (SDU=Service Data Unit)

IHL = 5 (32-bit words) = 20 bytes (copied in each fragment)

Identification = 16-bit sequence number ('unique' identification for each IP datagram; ID wrap-around: ID is too small for high speed networks: 2^{16} datagrams . 2^{10} bytes = 64 Mbyte)

Unlike datagrams, fragments are not independent. Each has its own copy of the IP-header. All fragments must arrive and within time or the 'entire' datagram is discarded.

Strategy: each fragment except for the last has maximum length

- The subnetwork_1 Maximum Transfer Unit MTU₁ is restricted to 512 bytes.

With a 20-byte IP header, this allows 492 data bytes per fragment.

The number of bytes in all except the final fragment must be divisible by 8 (fragment offset is specified in terms of 64-bit words = 8 byte words): $492 / 8 = 61.5$ bytes

max. number of data bytes / fragment is adjusted downward to: $61 \times 8 = 488$ bytes

maximum length fragment: Total Length = $488+20 = 508$ bytes (\rightarrow offset = 61)

final fragment: Total Length = $(930-488)+20 = 462$ bytes

- The subnetwork_2 Maximum Transfer Unit MTU₂ is restricted to 296 bytes (smallest MTU in the path, called path MTU). This requires *repeated fragmentation*.

With a 20-byte IP header, this allows 276 data bytes per fragment.

The number of bytes in all except the final fragment must be divisible by 8 :

max. number of data bytes / fragment is adjusted downward to: $34 \times 8 = 272$ bytes

maximum length fragment: Total Length = $272+20 = 292$ bytes (\rightarrow offset = 34)

final fragment: Total Length = $(488-272)+20 = 236$ bytes

- Reassembly is only done in the ES (host), not in the IS (router)! This is done based on the unique *Identification* of the fragments. If one of the fragments is missing, the IP-datagram is discarded, without an attempt to recover the datagram. Buffering is needed for the reassembly because *fragments do not necessarily arrive in order*.

+ The ID allows the fragments to be routed independently (connectionless).

- In subnetwork_3 the packets are much smaller than the allowed payload.

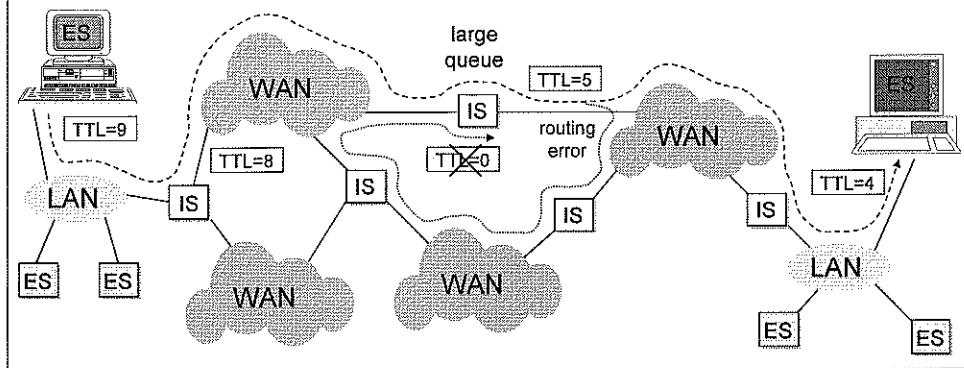
Transmission is not optimal in this subnetwork (header overhead).

4.1.3 Time To Live (TTL)

Counter used to *limit lifetime* of IP datagrams (maximum = 255 = $2^8 - 1$).

- decremented at each router (hop counting)
- decremented multiple times when queued for a long time in a router
- when reached zero, the router discards the IP datagram and sends an error message to the source (ICMP)

TTL guarantees that an IP datagram cannot travel around forever.
(even if a routing table is corrupt or routers route the datagrams in a circle)



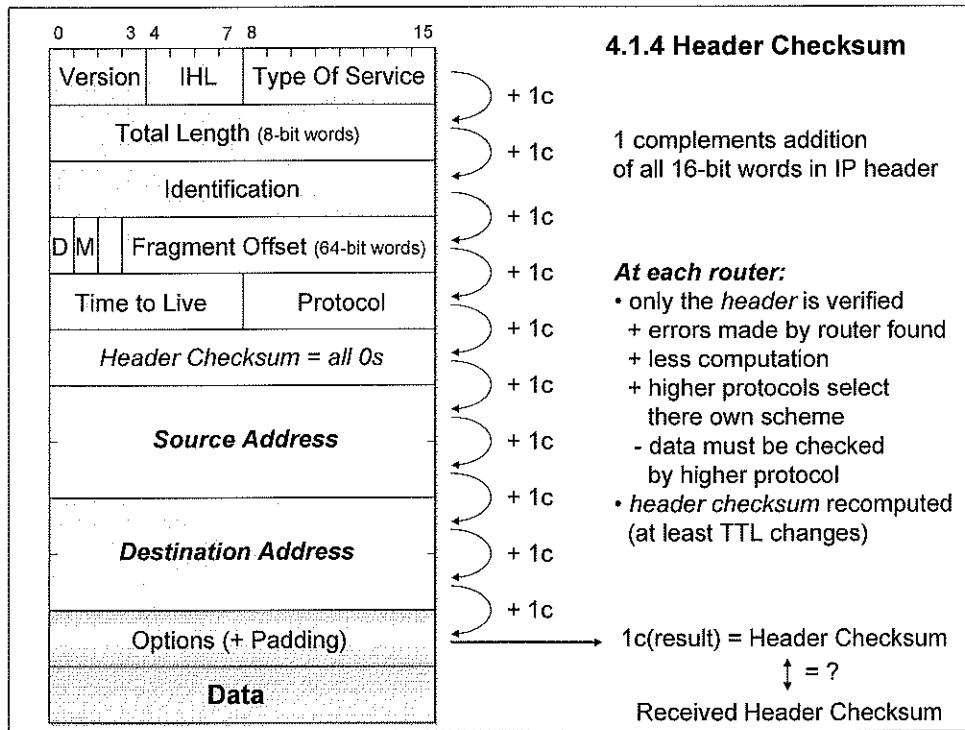
The field *Time To Live (TTL)* specifies how long (number of network hops) the datagram is allowed to remain in the internet system (default value of 64). The idea is both simple and important: whenever a host injects a datagram into the internet, it sets a maximum time that the datagram should survive. *Routers and hosts that process datagrams must decrement the Time To Live field as time passes and remove the datagram from the internet when its time expires.*

Estimating exact times is difficult because routers do not usually know the transit time for physical networks. A few rules simplify processing and make it easy to handle datagrams without synchronized clocks.

Most routers hold a datagram for less than a second. In this case the router along the path from source to destination is required to *decrement the Time To Live field by 1 hop* when it processes the datagram header. The TTL effectively becomes a hop counter, decremented by one by each router.

To handle cases of *congested (overloaded) routers* that introduce long delays, each router records the local time when the datagram arrives and leaves. The Time To Live field is *decremented by the number of seconds* that the datagram remained inside the router waiting for service [RFC 1009].

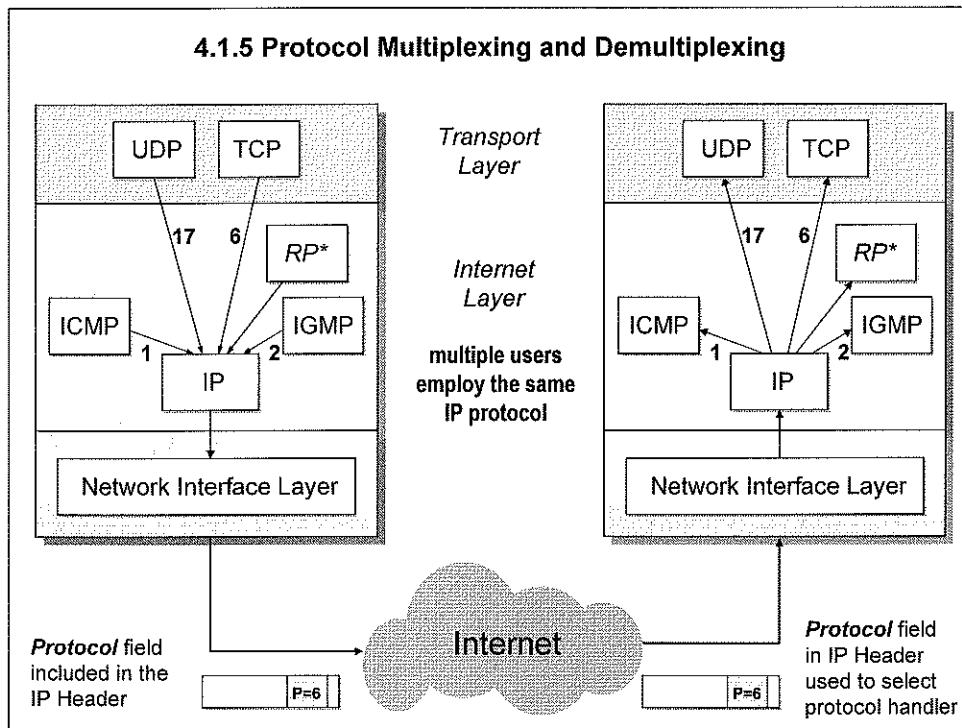
Whenever a *Time To Live field reaches zero*, the router discards the datagram and sends an error message (ICMP) back to the source. The idea of keeping a timer for datagrams is interesting because it guarantees that datagrams cannot travel around an internet forever, even if routing tables become corrupt and routers route datagrams in a circle. It prevents datagrams from ending up in infinite loops.



Field *Header Checksum* ensures integrity of header values. The IP checksum is formed by treating the header as a sequence of 16-bit integers (in network byte order, Big Endian), adding them together using one's complement arithmetic (end-around carry: a carry-out of the high-order bit position must be added to the sum as a 1 in the LSB position), and then taking the one's complement (inverse) of the result. For purposes of computing the checksum field *Header Checksum* is initialized to a value of zero. Because some header fields may change during transit (e.g. TTL, segmentation related fields), this is *verified* and *recomputed* at each router.

It is important to note that the checksum *only applies to values in the IP header and not to the data*. Separating the checksum for headers and data has advantages and disadvantages. Because the header usually occupies fewer octets than the data, having a separate checksum reduces processing time at routers which only need to compute header checksums. The separation also allows higher level protocols to choose their own checksum scheme for the data. The chief disadvantage is that higher level protocols are forced to add their own checksum or risk having corrupted data go undetected.

Fields *source IP address* and *destination IP address* contain the 32-bit IP addresses of the datagram's sender and intended receiver. Although the datagram may be routed through many intermediate routers, the source and destination fields never change; they specify the IP addresses of the original source and ultimate destination.



The value in the *protocol* field specifies which high-level protocol was used to create the message being carried in the *data* area of a datagram. In essence, the value of *protocol* specifies the format of the *data* area. The mapping between a high level protocol and the integer value used in the *protocol* field must be administered by a central authority to guarantee agreement across the entire Internet. A list of assigned protocol identifiers is maintained by the Internet Assigned Number Authority (IANA):

- 1 = ICMP = Internet Control Message Protocol
- 2 = IGMP = Internet Group Management Protocol
- 8 = EGP = Exterior Gateway Protocol (RP = Routing Protocol)
- 89 = OSPF = Open Shortest Path First (RP = Routing Protocol)
- 6 = TCP = Transmission Control Protocol
- 17 = UDP = User Datagram Protocol

Communication protocols use a technique called *multiplexing* and *demultiplexing* throughout the layered hierarchy. When sending a message, the source host includes extra bits that encode the message type, originating application program, and protocols used. At the receiving end, the destination host uses the extra information to guide processing.

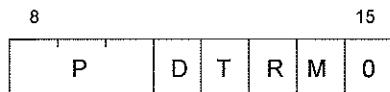
The IP layer in the source host sets the *protocol* field before transmission. The IP layer **multiplexes** different modules that send IP datagrams into IP datagrams by using the *protocol* field to specify the content of the IP datagram (data field).

The IP layer in the destination host uses the *protocol* field to choose a module that handles the incoming IP datagram. The IP layer **demultiplexes** the IP datagram based on its protocol type.

Multiplexing and demultiplexing occur at almost every protocol layer.

4.1.6 Type of Service (TOS)

*Specification by an upper-layer protocol
how the current IP datagram should be handled.*



P = Precedence → priority (0-7) = level of importance

Type of Transport (select one):

- *D = Delay* → normal / low
- *T = Throughput* → normal / high
- *R = Reliability* → normal / high
- *M = Monetary Cost* → normal / low

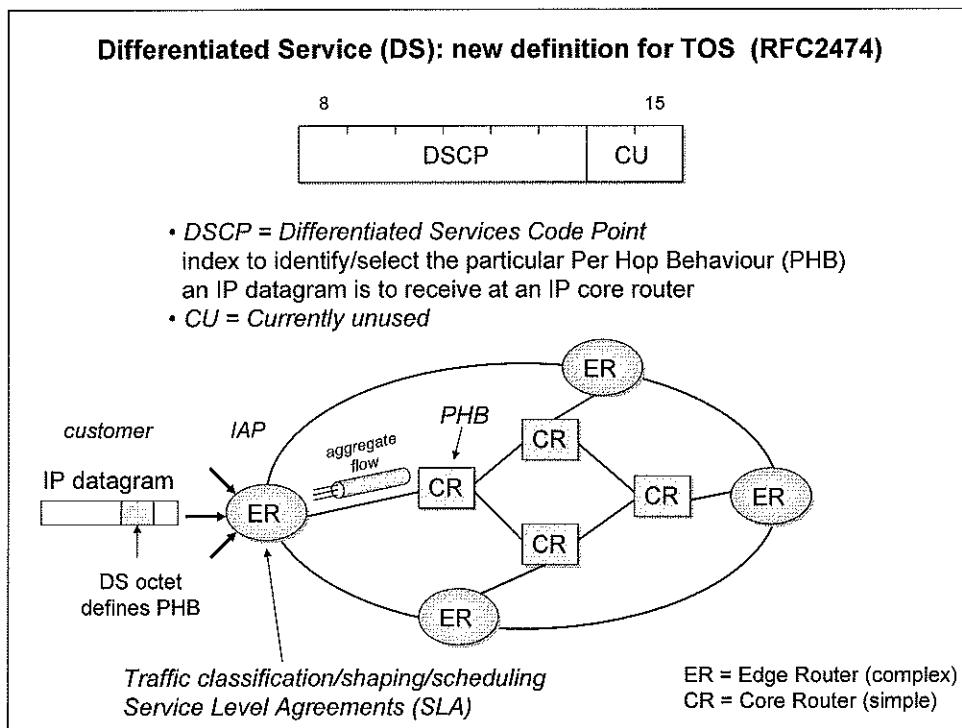
*Hint to the routing algorithm (select between paths with different performances)
Internet does not guarantee the requested Type of Service!*

Three *Precedence* bits specify datagram precedence, with values ranging from 0 (normal precedence) through 7 (network control), allowing senders to indicate the importance of each datagram. Although *most host and router software ignores Type Of Service (TOS)*, it is an important concept because it provides a mechanism that will eventually allow control information to have precedence over data.

Bits D, T, and R specify the *type of transport* the datagram desires. When set, the D bit requests *low delay*, the T bit requests *high throughput*, the R bit requests *high reliability* and the M bit requests *low monetary cost*. Of course, it may *not be possible for an internet to guarantee the type of transport requested* (i.e., it could be that no path to the destination has the requested property). The transport request is a *hint to the routing algorithms*, not as a demand. If a router does know more than one possible route to a given destination, it can use the type of transport field to select one with characteristics closest to those desired. Suppose the router can select between a low capacity leased line or a high bandwidth (high delay) satellite connection. Datagrams carrying keystrokes from a user to a remote computer have the D bit set requesting a quick delivery, while datagrams carrying a bulk file transfer could have the T bit set requesting that they travel across the high capacity satellite path.

The routing algorithms must choose from among underlying physical network technologies that each have characteristics of delay, throughput, and reliability. Often, a *given technology trades off one characteristic for another* (e.g., higher throughput rates at the expense of longer delay). The algorithm is given a hint about what is most important; *it seldom makes sense to specify all three types of service*.

Recommended values for TOS field: Telnet (D=1), FTP-control (D=1), FTP-data (T=1), SMTP-command phase (D=1), SMTP-data phase (T=1), DNS UDP query (D=1), SNMP (R=1), NNTP=Network News Transfer Protocol (M=1).



RSVP is intended to support quality-of-service (QoS) offering in the Internet and in private internets. Although RSVP is a useful tool in this regard, it is relatively complex to deploy. Further, it may not scale well to handle large volumes of traffic because of the amount of control signalling required to coordinate integrated QoS offerings and because of the maintenance of state information required at routers.

As the burden on the Internet grows, and as the variety of applications grow, there is an immediate need to provide differing levels of QoS to different traffic flows. The differentiated services (DS) architecture (RFC 2475) is designed to provide a *simple, easy-to-implement, low-overhead tool to support a range of network services that are differentiated on the basis of performance*.

Several key characteristics of DS contribute to its efficiency and ease of deployment:

- IP packets are *labelled* for differing QoS treatment using the existing IPv4 Type of Service octet. Thus, no change is required to IP.
- A *Service Level Agreement (SLA)* is established between the service provider (internet domain) and the customer *prior to the use of DS*. This avoids the need to incorporate DS mechanisms in applications. Thus, existing applications need not be modified to use DS.
- DS provides a built-in *aggregation mechanism*. All traffic with the same DS octet is treated the same by the network service. For example, multiple voice connections are not handled individually but in the aggregate. This provides for good scaling to larger networks and traffic loads.
- DS is implemented in individual routers by *queuing and forwarding packets based on the DS octet*. Routers deal with each packet *individually* and do not have to save state information on packet flows.

Although DS is intended to provide a simple service based on relatively simple mechanisms, the set of RFCs related to DS is relatively complex.

Differentiated Service (DS): Per Hop Behaviour

DSCP = label to classify packets for DS

- **DSCP = 000000 → PHB BE = Best Effort Forwarding**
Default PHB
- **DSCP = xxx000 → PHB PS = IPv4 Precedence Service**
Backward compatibility with the IPv4 Precedence service
- **DSCP = 101110 → PHB EF = Expedited Forwarding**
Virtual leased line service.
Low loss, low latency, low jitter and guaranteed bandwidth for an aggregate flow.
- **DSCP = 0xxxx0 → PHB AF = Assured Forwarding**
Four different service classes (AF1, AF2, AF3, AF4) defined by the ISP (Internet Service Provider) in terms of delay, throughput, loss.
Each class has 3 different levels of *drop priority* (low, medium, high).

PHB = preferential, differentiated queuing behaviour in the IP Core Routers for aggregate flows (the PHB can be implemented differently for each router)

Differentiated Services

The DS type of service is provided within a DS domain, which is defined as a contiguous portion of the Internet over which a consistent set of DS policies are administered. Typically, a DS domain would be under the control of one administrative entity. The services provided across a DS domain are defined in a *Service Level Agreement* (SLA), which is a service contract between a customer and the service provider that specifies the *forwarding service that the customer should receive for various classes of packets*. A customer may be a user organization or another DS domain. Once the SLA is established, the customer submits packets with the DS octet marked to indicate the packet class. The service provider must assure that the customer gets at least the agreed QoS for each packet class. To provide that QoS, the service provider must configure the appropriate forwarding policies at each router (based on DS octet value) and must measure the performance being provided each class on an ongoing basis. (Traffic Shaping = The process of delaying packets within a packet stream to cause it to conform to some defined traffic profile).

If a customer submits packets intended for destinations within the DS domain, then the DS domain is expected to provide the agreed service. If the destination is beyond the customer's DS domain, then the DS domain will attempt to forward the packets through other domains, requesting the most appropriate service to match the requested service.

PHB = the externally observable forwarding behaviour applied at a node (router) to a behaviour aggregate.

The default PHB is *Best Effort Forwarding*. Packets are forwarded in the order that they are received as soon as link capacity becomes available (higher priority packets are forwarded first).

4.1.7 Options

Options: primarily used for *network control, debugging and measurement enabled by the source*
 not required in every datagram, variable length
 padded out to a multiple of 4 bytes

- **Security**

Specifies how secret the IP datagram is (ignored by most routers).

- **Strict Source Routing**

Gives the *complete routing path* to be followed from source to destination.

- send emergency packets when routing tables are corrupted
- make timing, throughput measurements

- **Loose Source Routing**

The source host gives a *list of routers* not to be missed.

- **Record Route**

Each router appends its IP address \leq 9 IP addresses

- **Timestamp**

Each router appends its IP address and timestamp (in ms) \leq 4 (IP,time) pairs

control

monitor

Control: The idea behind *source routing* is that it provides a way for the sender to dictate a path through the internet. For example, to test the throughput over a particular physical network, N, system administrators can use source routing to force IP datagrams to traverse network N even if routers would normally choose a path that did not include it. The ability to make such tests is especially important in a production environment, because it gives the network manager freedom to route users' datagrams over networks that are known to operate correctly while simultaneously testing other networks. Of course, such routing is only useful to people who understand the network topology; the average user has no need to know or use it.

Monitor: The *routing and timestamp options* are the most interesting because they provide a way to monitor or control how internet routers route datagrams. The record route option allows the source to create an empty list of IP addresses and arrange for each router that handles the datagram to add its IP address to the list.

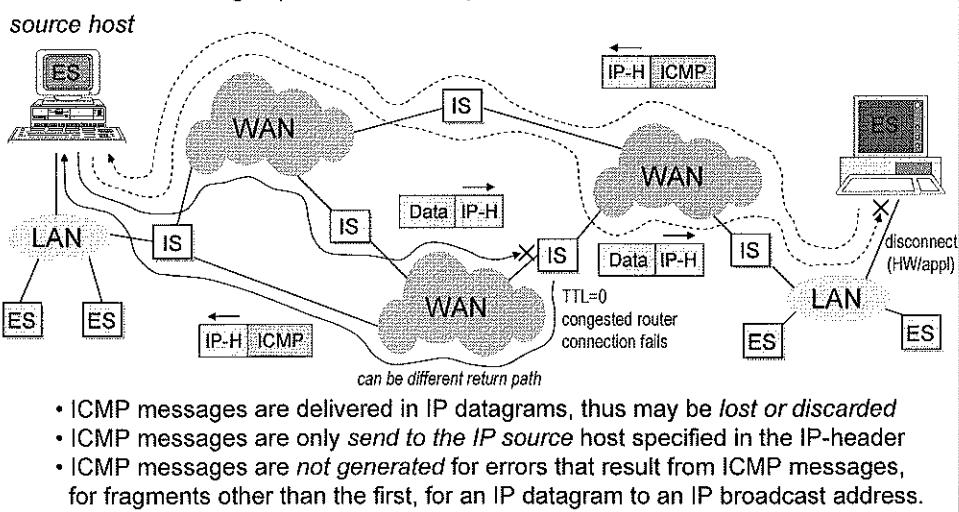
Whenever a *router* handles a datagram that has the record route option set, the router adds its address to the record route list (RFC 791: the router records its 'outgoing' IP-address. The option space is limited: the maximum IP-header length is $15 \times 4 = 60$ octets. The standard IP header takes 20 bytes. The options code needs 3 bytes so the space to hold IP addresses of routers is limited to 37 bytes. This results in a maximum of 9 IP addresses or 4 (IP address, timestamp) pairs that can be registered in the IP header option space.

When the datagram arrives, the *destination* host must extract and process the list of IP addresses. If the destination handles the datagram as usual, it will ignore the recorded route. Note that the source must agree to enable the record route option and the destination must agree to process the resultant list; a single machine will not receive information about recorded routes automatically just because it turns on the record route option.

5. ICMP = Internet Control Message Protocol (RFC792)

IP: connectionless, unreliable, best effort delivery of datagrams

ICMP: from hosts / routers to source host
control and error reporting (\neq error correct) + testing
integral part of each IP implementation



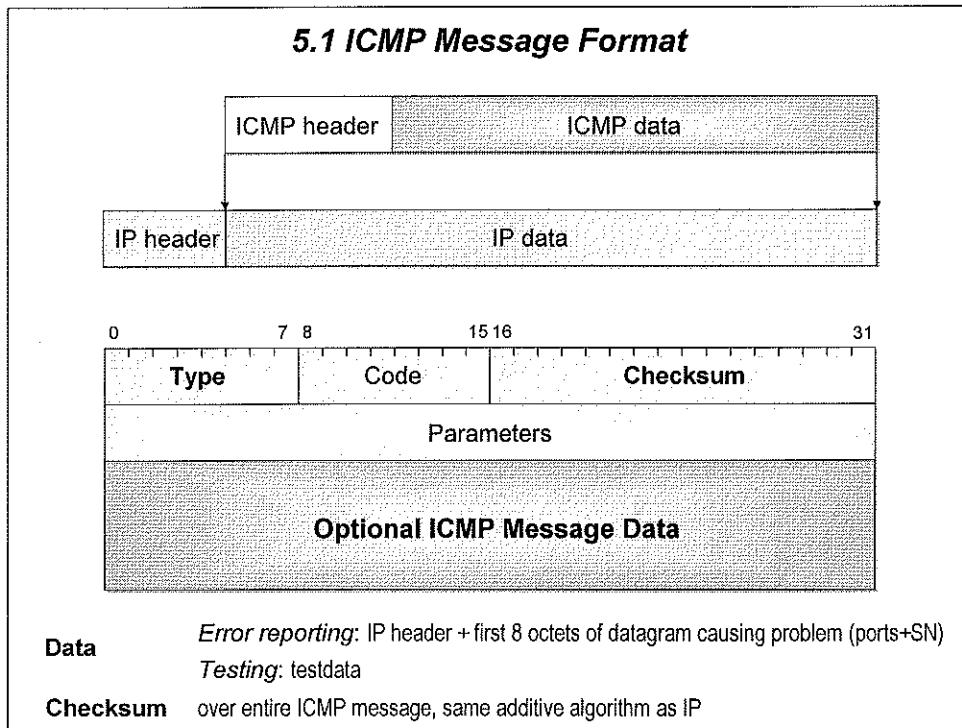
IP fails to deliver datagrams when a network fails, a destination host is disconnected, the TTL counter expires or a router becomes congested. The ICMP provides a special-purpose message mechanism to inform about failures and is an integral part of IP.

ICMP only *reports* error conditions (problem feedback) to the *original IP source* in response to a datagram. The source must relate errors to individual application programs and take action to *correct* the problem.

ICMP is restricted to communication with the *IP source*. The datagram only contains address fields that specify the IP source and the destination; it does not contain a complete record of its trip through the internet (except for unusual cases where the record route option is used). Because routers can establish and change their own routing tables, there is no global knowledge of routes. Thus, when a datagram reaches a given router, *it is impossible to know the route it has taken* to arrive there. If the router detects a problem, it cannot know the set of intermediate routers that processed the datagram, so it cannot inform them of the problem. Instead of silently discarding the datagram, the router uses ICMP to inform the IP source that a problem has occurred, and trusts that host administrators will cooperate with network administrators to locate and repair the problem.

Each ICMP message travels across the internet *in the data portion of an IP datagram*, and are routed exactly like datagrams carrying information for users; there is no additional reliability or priority. Thus, error messages themselves may be lost or discarded. Source and network must be robust to loss of all ICMP messages.

In an already congested network, the error message may cause additional congestion. To avoid the problem of having error messages about error messages (prevent infinite recursion of ICMP message generation) *ICMP messages are not generated for errors that result from datagrams carrying ICMP error messages, for datagrams destined to an IP broadcast (prevent broadcast storm) or for a fragment other than the first.*



Although each ICMP message has its own format, they all begin with the same three fields:

- an 8-bit integer message *Type* field that identifies the message (meaning and format)
- an 8-bit *Code* field that provides further information about the message type
- a 16-bit *Checksum* field (ICMP uses the same additive checksum algorithm as IP, but the ICMP checksum only covers the ICMP message).

- Optional ICMP *message data*

There are two types of ICMP messages: *error reporting* and *testing*

- ICMP messages that report errors always include the IP header (20 octets) and the first 8 octets (64 bits) of the datagram causing the problem. The reason for returning more than the datagram header alone is to allow the receiver to determine more precisely which protocol(s) and which application program were responsible for the datagram. Higher-level protocols in the TCP/IP suite are designed so that crucial information is encoded in the first 8 octets (64 bits) [TCP: source port, destination port, sequence number → they identify the TCP segment and the application].

- ICMP messages used for testing (request / reply) can include test data in the ICMP message data part.

5.2 ICMP Message Types

5.2.1 Error and Problem Reporting

Type	Name	Function
3	<i>Destination Unreachable</i>	Send by host or router when a segment is discarded: <ul style="list-style-type: none"> • Destination host/network unreachable or unknown • Protocol, port, SAP not present at destination host • Source routing used and route not available • Fragmentation needed and D=1 in IP header • Communication with destination network/host administratively prohibited
11	<i>Time Exceeded</i>	TTL expired and IP datagram discarded
12	<i>Parameter Problem</i>	Syntactic or semantic error in the IP header. ICMP message has a pointer to the octet in the datagram causing the error.
4	<i>Source Quench</i>	A congested router or host that discards datagrams requests the source to reduce the rate at which datagrams are sent.
5	<i>Redirect</i>	A router informs a host attached to one of its networks to use an alternative router 'on the same network' for forwarding datagrams to a specific destination.

The **IP header and the first 8 octets of the data field** are included in the ICMP message. This allows the *source host* to determine which protocol and application program were responsible for the IP datagram causing a problem

ICMP Reporting

ICMP messages are a response to an IP datagram. A reference to this IP datagram (the IP header and the first 8 octets) is included in the ICMP message. This information is interpreted by the source host.

Path MTU Discovery (PMTUD) mechanism based on Don't Fragment flag (D)

Fragmentation is mostly avoided due to extra processing needed in the routers and problems like Deterministic Fragment Loss and IP Identification wrap-around.

The PMTUD mechanism proposes a guaranteed way to avoid fragmentation: set the Don't Fragment (D) bit in the IP header on every datagram. If the datagram reaches a router that cannot forward it, because of its length, the router must drop the datagram and return an ICMP "Destination Unreachable/Fragmentation Needed and D set" message. The host receiving this message reduces its packet size until the router can forward the datagram. The MTU estimate value is communicated to TCP, which changes the MSS (Maximum Segment Size) for new connections to that destination.

5.2.2 Testing (Request / Reply)

Type	Name	Function
8/0	<i>Echo Request/Reply</i>	Reachability Testing The destination is instructed (request) to return (reply) the same message (optional data of variable length) to the source. It is a mechanism to check the reachability of a specified host or router.
13/14	<i>Timestamp Request/Reply</i>	Performance Testing It is a mechanism to sample the delay characteristic and the round-trip time (RTT) of the internet. 3 time fields (32-bit) in Universal Time (ms since midnight): <ul style="list-style-type: none"> • Originate Timestamp: source transmits request • Receive Timestamp: destination receives request • Transmit Timestamp: destination transmits reply
17/18	<i>Address Mask Request/Reply</i>	Subnet Mask Testing A host sends to a local router a request to learn from the reply the subnet mask (32-bit) used for the 'local' network.

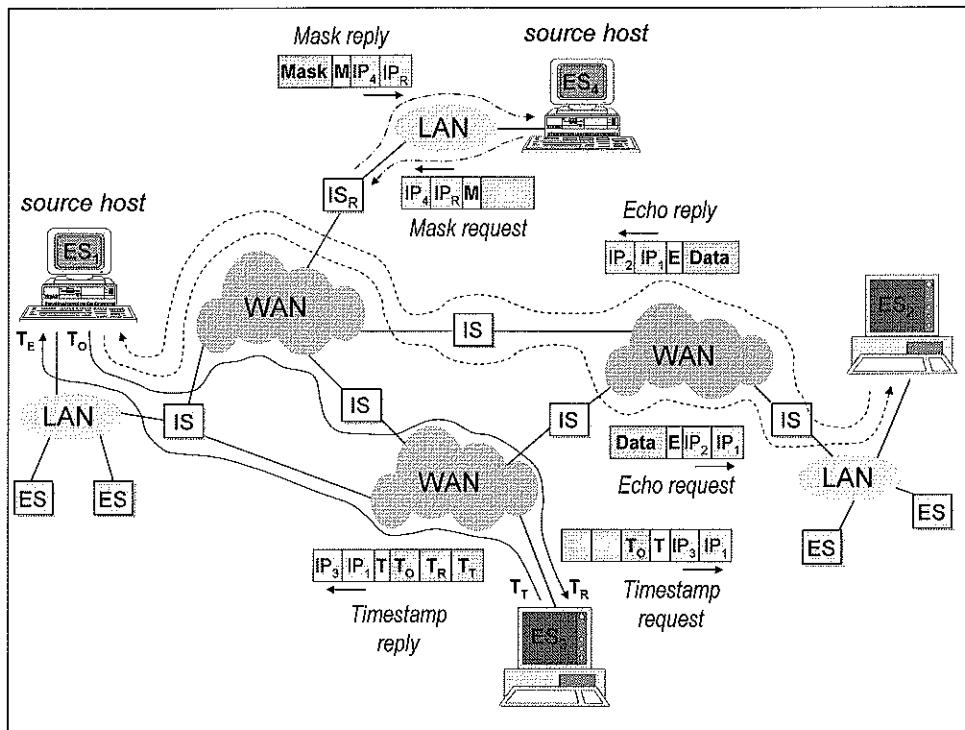
Parameter field = 16-bit Identifier + 16-bit Sequence Number
(Used by the IP-source to match replies to requests.)

The ICMP protocols provides facilities to help network managers or users identify network problems.

Echo Request / Echo Reply

One of the most frequently used debugging tools invokes the ICMP *echo request* and *echo reply* messages. A host or router sends an ICMP echo request message to a specified destination. Any machine that receives an echo request formulates an echo reply and returns it to the original sender. *The request contains an optional data area; the reply contains a copy of the data sent in the request.* The echo request and associated reply can be used to test whether a *destination is reachable* and responding. Because both the request and reply travel in IP datagrams, successful receipt of a reply verifies that major pieces of the transport system work. First, IP software on the source machine must route the datagram. Second, intermediate routers between the source and destination must be operating and must route the datagram correctly. Third, the destination machine must be running (at least it must respond to interrupts), and both ICMP and IP software must be working. Finally, routes in routers along the return path must be correct.

On many systems, the command users invoke to send ICMP echo requests is named *ping* (Packet InterNet Gropher). Ping does not use TCP or UDP. Sophisticated versions of ping send a series of ICMP echo requests, capture responses, and provide statistics about datagram loss. They allow the user to specify the length of the data being sent and the interval between requests. Less sophisticated versions merely send one ICMP echo request and await a reply.



Timestamp Request / Timestamp Reply

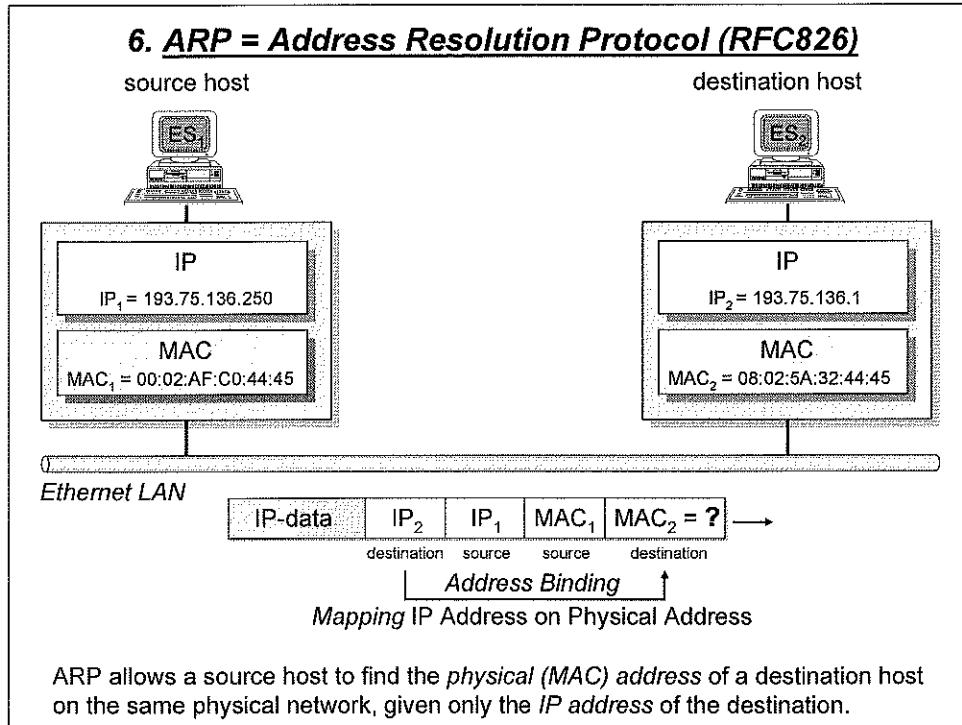
Hosts use the three timestamp fields to compute estimates of the delay time between them and to synchronize their clocks. Because the reply includes the Originate Timestamp (T_O) field, the time at which the request entered the remote machine (Receive Timestamp = T_R), as well as the time at which the reply left (Transmit Timestamp = T_T), the host can compute the network transit time, and from that, estimate the differences in remote and local clocks.

In practice, accurate estimation of round-trip delay can be difficult and substantially restricts the utility of ICMP timestamp messages. To obtain an accurate estimate of *round trip time* ($RTT = T_O - T_E$) one must take *many measurements and average them*. However, the round-trip time between a pair of machines that connect to a large internet *can vary dramatically*, even over short periods of time. Because IP is a best-effort technology, datagrams can be dropped, delayed, or delivered out of order. Thus, merely taking many measurements may not guarantee consistency; sophisticated statistical analysis may be needed to produce precise estimates.

Address Mask Request / Address Mask Reply

When hosts use subnet addressing, some bits in the hostid portion of their IP address identify a physical network. To participate in subnet addressing, hosts need to know which bits of the 32-bit internet address correspond to the physical network and which correspond to host identifiers. The information needed to interpret the address is represented in a 32-bit quantity called the subnet mask.

To learn the subnet mask used for the local network, a host can send an address mask request message to a router and receive an address mask reply. The host making the request can either send the message *directly*, if it knows the router's address, or *broadcast* the message if it does not.



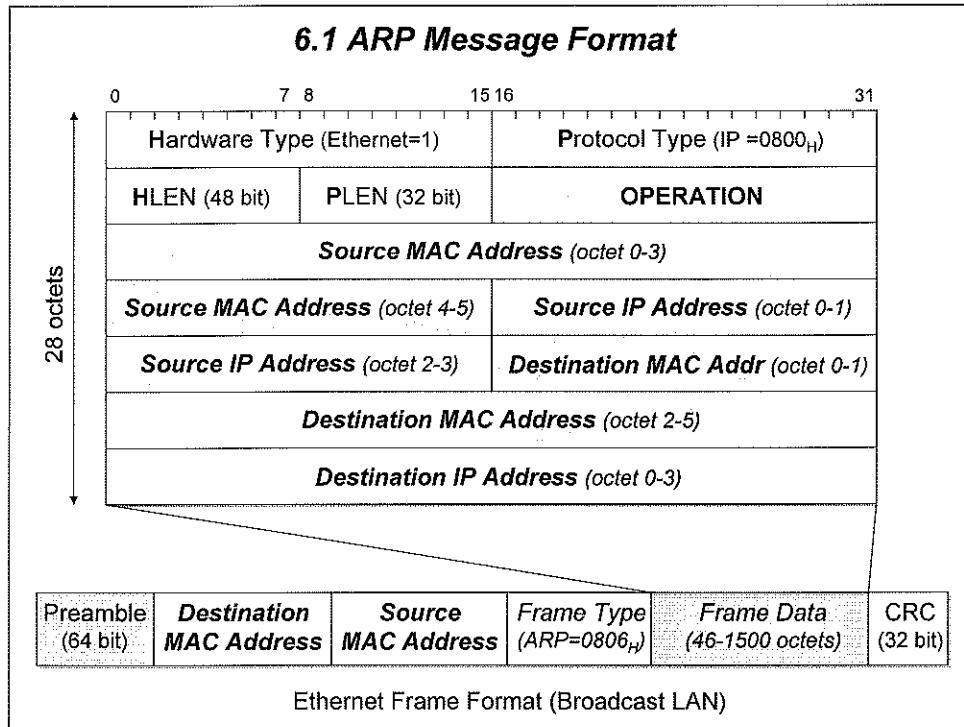
Two hosts on the same physical network can communicate *only if they know each other's physical network address*. A host or a router needs a technique to *map an IP address to the correct physical address* when it needs to send a packet across a physical net.

Consider two machines ES_1 and ES_2 that share a physical network. Each has an assigned IP address IP_1 and IP_2 and a physical address MAC_1 and MAC_2 . The goal is to devise low-level software that *hides physical addresses* and *allows higher-level programs to work only with internet addresses*. Ultimately, however, communication must be carried out by physical networks using whatever physical address scheme the hardware supplies. Suppose machine ES_1 wants to send a packet to machine ES_2 across a physical network to which they both attach, but ES_1 has only ES_2 's internet address IP_2 . The question arises: how does ES_1 map that address to ES_2 's physical address, MAC_2 ?

The problem of *mapping high-level addresses to physical addresses* is known as the *address resolution problem* and has been solved in several ways. Some protocol suites keep tables in each host that contain pairs of high-level and physical addresses. Others solve the problem by encoding hardware addresses in high-level addresses. Using either approach exclusively makes high-level addressing awkward at best.

ARP is a low-level protocol that hides the underlying network physical addressing, permitting to *assign independent (from MAC addresses) IP addresses to every host*.

Remark: ARP works well only on *broadcast LANs*. In point-to-point LANs (like ATM), ARP requests are replied by an ARP server. When a host boots up, it registers its MAC address with the ARP server.



Encapsulation and Identification

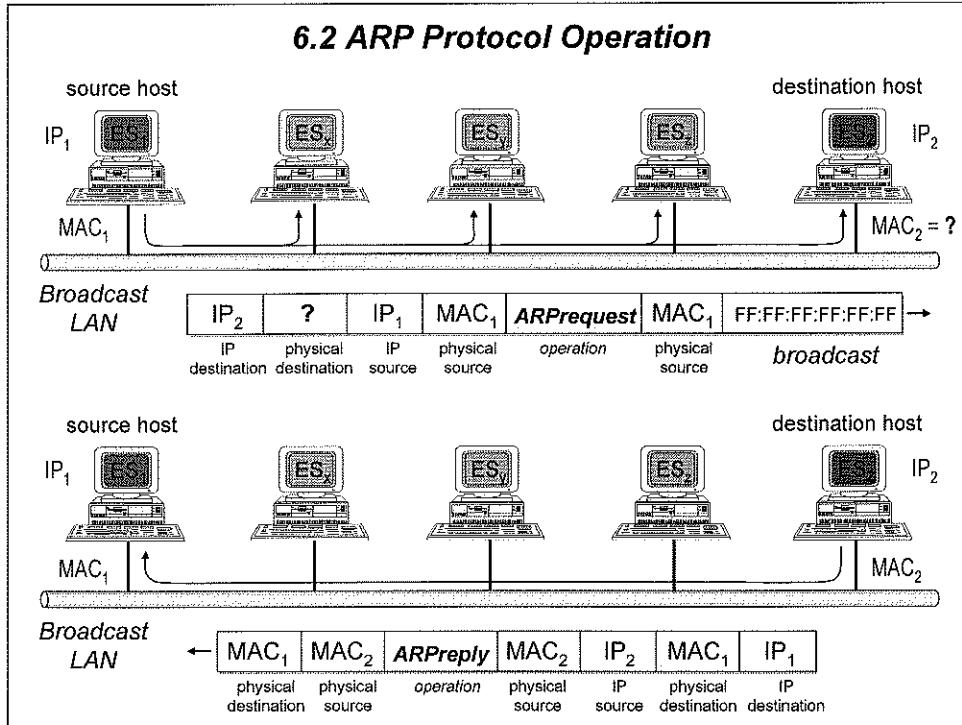
An ARP message is carried in the data portion of a physical frame (*encapsulation*).

Ethernet Frame: To *identify* the frame as carrying an ARP request or ARP reply, the sender assigns a special value to the type field in the frame header and places the ARP message in the frame's data field. Ethernet frames carrying ARP messages have a Frame Type field of a standard value 0806_H . (Frame Type field for IP = 0800_H , for RARP = 8035_H) The 28 octets of the ARP messages are padded until 46 octets.

IEEE 802.3 frame: Frame Length: $2E_H$ (46 octets) - $5DC_H$ (1500 octets).

ARP Message Format (for IP-to-Ethernet resolution)

- Unlike most protocols, the data in ARP packets does not have a fixed-format header. Instead, the message is designed to be useful with a variety of network technologies. ARP can be used with arbitrary *physical (hardware) addresses* and arbitrary *protocol addresses*. The example shows the 28-octet ARP message format used on Ethernet hardware (where physical addresses are 48-bits or 6 octets long), when resolving IP protocol addresses (which are 32-bits or 4 octets long).
- Fields *HLEN* and *PLEN* allow ARP to be used with arbitrary networks because they specify the length of the MAC (hardware) address and the length of the high-level protocol address. The source supplies its hardware address and IP address, if known, in fields *Source MAC Address* and *Source IP Address*.
- Field *Hardware Type* specifies a hardware interface type for which the source seeks an answer; it contains the value 1 for Ethernet. Similarly, field *Protocol Type* specifies the type of high-level protocol address the sender has supplied; it contains 0800 (HEX) for IP addresses.
- Field *Operation* specifies:
 - ARP request (1), ARP response (2)
 - RARP request (3), RARP response (4)



Resolution Through Direct Mapping

A numbering scheme can be selected that makes address resolution efficient. This means selecting a function f that maps IP addresses to physical addresses. Resolving IP address IP_1 means computing: $MAC_1 = f(IP_1)$. But MAC_1 is mostly fixed!

IP_1 and MAC_1 pairs can also be stored in a table and to resolve an IP address a search in the table is done.

Resolution Through Dynamic Binding

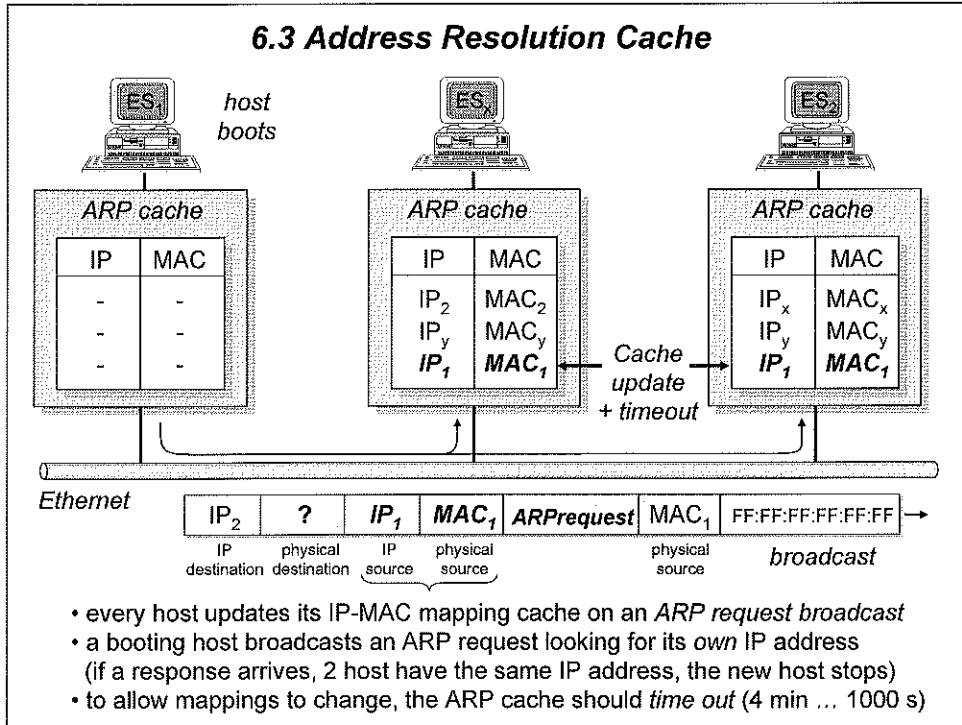
The Address Resolution Protocol (ARP) is an efficient and dynamic solution to the address resolution problem for networks like the Ethernet that have *broadcast* capability. New hosts can be added to the network without recompiling code and maintenance of a centralized database is not required. To *avoid maintaining* a table of mappings, a *low-level protocol is used to bind addresses dynamically*.

When host ES_1 wants to resolve the IP address IP_2 , it *broadcasts an ARP request* that asks the host with IP address IP_2 to respond with its physical address, MAC_2 . All hosts, including ES_2 , receive the request, but only host ES_2 recognizes its IP address and sends an *ARP reply* that contains its physical address. When ES_1 receives the reply, it uses the physical address to send the internet packet directly to ES_2 .

When making a request, the source also supplies the destination IP address (ARP), or destination MAC address (RARP). Before the destination host responds, it fills in the missing addresses, swaps the destination and source pairs and changes the operation to a reply. Thus, a reply carries the IP and physical addresses of the requestor, as well as the IP and MAC addresses of the machine for which a binding was sought.

The Address Resolution Protocol, ARP, allows a host to find the physical address of a target host on the same physical network, given only the destination's IP address.

ARP frame ≠ IP datagram



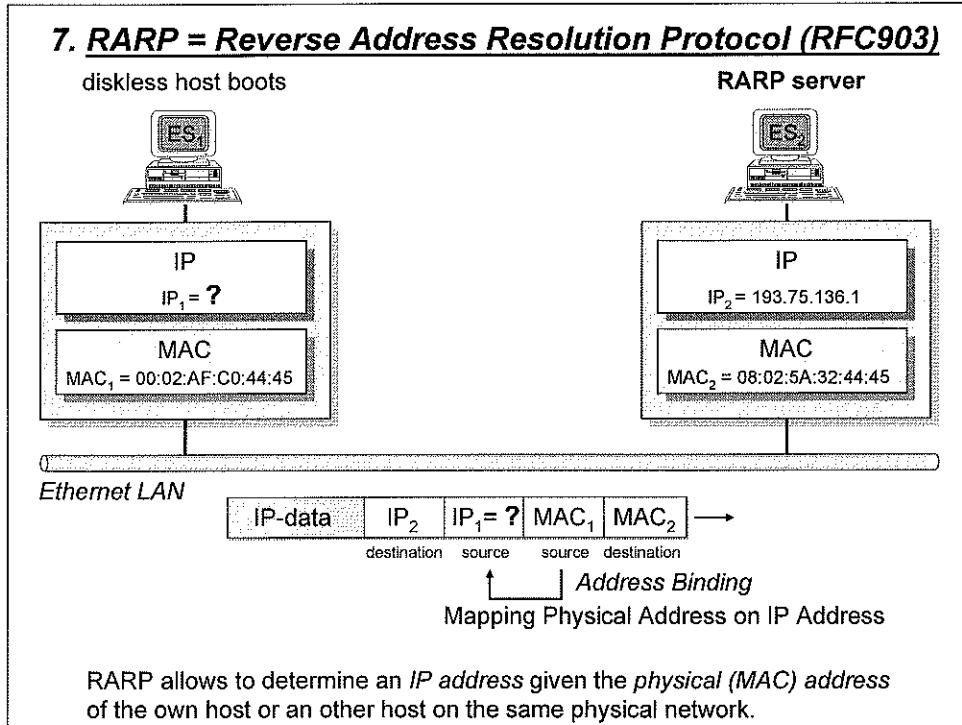
It may seem silly that for ES_1 to send a packet to ES_2 it first sends a broadcast that reaches ES_2 . Or it may seem even sillier that ES_1 broadcasts the question, "How can I reach you?" instead of just broadcasting the packet it wants to deliver. But there is an important reason for the exchange. *Broadcasting is far too expensive* to be used every time one host needs to transmit a packet to another because it requires every host on the network to process the broadcast packet. To reduce communication costs, hosts that use ARP maintain a *cache of 'recently' acquired IP-to-MAC address bindings* so they do not have to use ARP repeatedly. Whenever a host receives an ARP reply, it saves the host's IP address and corresponding MAC (hardware) address in its cache for successive lookups. When transmitting a packet, the host always looks in its cache for a binding before sending an ARP request. If the host finds the desired binding in its cache, it need not broadcast on the network.

- If host ES_1 is about to use ARP because it needs to send to ES_2 , there is a high probability that host ES_2 will need to send to ES_1 in the near future. Extra network traffic is avoided by arranging for ES_1 to include its IP-to-physical address binding when sending a request to ES_2 .

Because ES_1 broadcasts its initial ARP request, *all* hosts on the network receive it and can extract and store in their cache ES_1 's IP-to-MAC address binding.

- Every host broadcasts its mapping when it boots. This broadcast is generally done in the form of an *ARPrequest looking for its own IP address*. There should be *no response*, but a side effect of the broadcast is to make an entry in everyone's ARP cache. If a response does arrive, two machines have been assigned the same IP address. The new one should inform the system manager and not boot.

- To allow mappings to change (e. g replacement of an Ethernet board gives a new Ethernet address), entries in the ARP cache should *time out* after a few minutes (recommended = 4 min, practice = 1000sec)

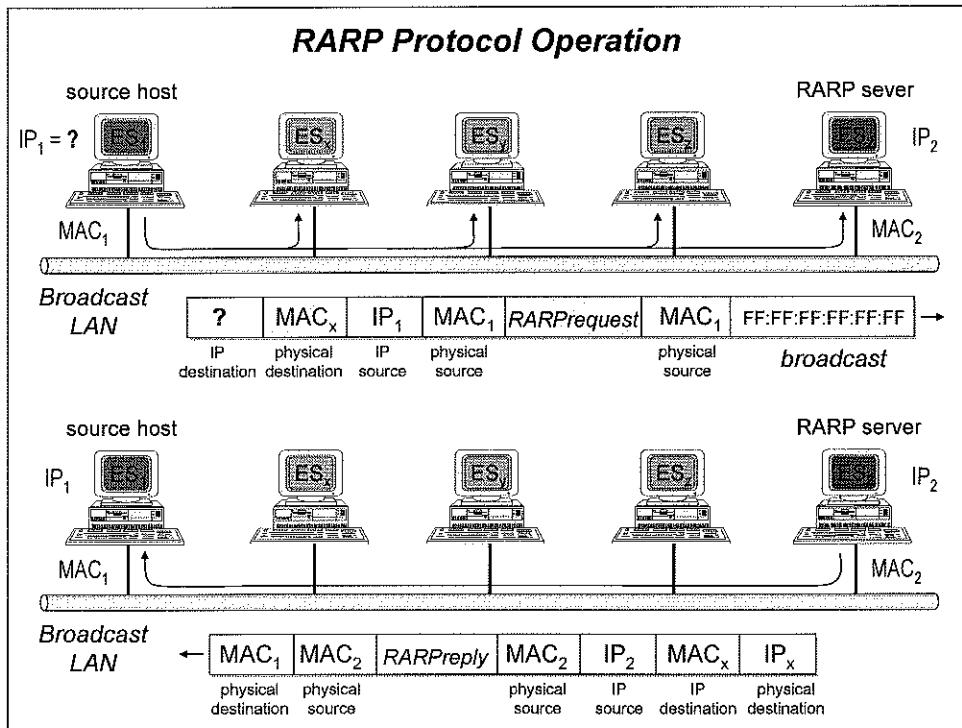


Physical network addresses are both low-level and *hardware dependent*, and each host using TCP/IP is assigned a 32-bit IP address that is *independent* of the host's hardware address. Application programs always use the IP address when specifying a destination. Hosts and routers must use physical addresses to transmit datagrams across underlying networks; they rely on address resolution schemes like ARP to perform the binding.

The question arises, "How does a diskless machine determine its IP address at start-up (boot)?" Such hosts will normally get their binary image of the *operating system from a remote server*. Furthermore, because many diskless hosts use standard TCP/IP file transfer protocols to obtain their initial boot image, they must obtain and use an IP address before the operating system runs.

To allow a single software image to be used on a set of hosts, it must be built without having the machine's IP address bound into the image. In particular, designers try to keep *both bootstrap code and initial operating system images free from specific IP addresses*, so the same image can be run on many hosts. When such code starts execution on a diskless host, it must use the network to contact a server to obtain the host's IP address.

The idea behind finding an IP address is simple: the diskless host sends a request to a RARP server, and waits until the server sends a response. The server has a disk where it keeps a database of internet addresses. In the request, the host needing to know its internet address must uniquely identify itself, so the server can look up the correct internet address and send a reply. Both the host that issues the request and the server that responds use physical network addresses during their brief communication.



Like an ARP message, a RARP message is sent from one host to another encapsulated in the data portion of an Ethernet frame. An Ethernet frame carrying a RARP request has the usual preamble, Ethernet source and destination addresses, and packet type fields in front of the frame. The Frame Type contains the value 8035_H to identify the contents of the frame as a RARP message. The data portion of the frame contains the 28-octet RARP message.

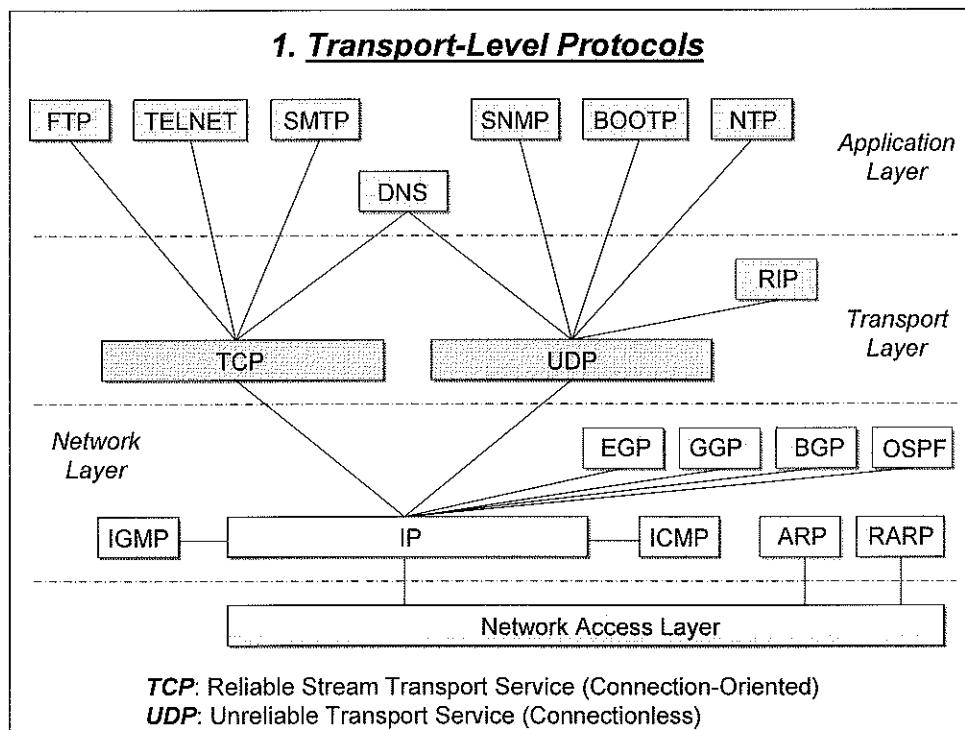
The sender *broadcasts* a RARP request that specifies itself as both the sender and destination host ($MAC_x = MAC_1$), and supplies its physical (MAC) network address in the destination physical address field. All hosts on the network receive the request, but *only those authorized to supply the RARP service process the request* and send a reply; such hosts are known as RARP servers. For RARP to succeed, the network must contain at least one RARP server.

Servers answer requests by filling in the destination protocol address field, changing the message type from request to reply, and sending the reply back directly to the host making the request. The original host receives replies from 'all' RARP servers, even though only the first is needed.

All communication between the host seeking its IP address and the server supplying it must be carried out using only the physical network. The protocol allows a host to ask about an arbitrary destination ($MAC_x \neq MAC_1$). Thus, the sender supplies its hardware address separate from the destination hardware address, and the server is careful to send the reply to the sender's hardware address. On an Ethernet, having a field for the sender's hardware address may seem redundant because the information is also contained in the Ethernet frame header. However, not all Ethernet hardware provides the operating system with access to the physical frame header.

RARP frame ≠ IP datagram



Application Layer (DARPA Services)

FTP = File Transfer Protocol

TELNET = Virtual Terminal

SMTP = Simple Mail Transfer Protocol

DNS = Domain Name Server

SNMP = Simple Network Management Protocol

BOOTP = Bootstrap Protocol

NTP = Network Time Protocol

Transport Layer

RIP = Routing Information Protocol

UDP = User Datagram Protocol

TCP = Transmission Control Protocol

Internet Layer

EGP = Exterior Gateway Protocol

GGP = Gateway to Gateway Protocol

BGP = Border Gateway Protocol

OSPF = Open Shortest Path First

IP = Internet Protocol

IGMP = Internet Group Management Protocol

ICMP = Internet Control Message Protocol

ARP = Address Resolution Protocol

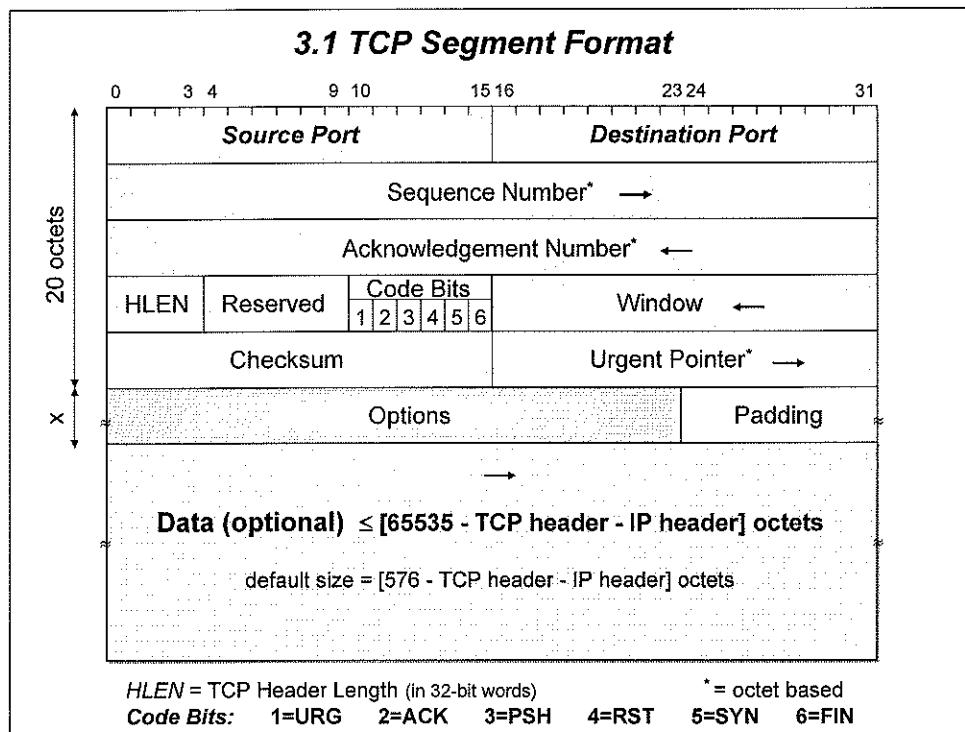
RARP = Reverse Address Resolution Protocol

2. TCP = Transmission Control Protocol

- ***connection oriented (point-to-point virtual circuit) - “end-to-end”***
TCP does connection (two endpoints) establishment before data transfer, to establish *initial sequence numbers* for data transfer in each direction.
- ***unstructured byte-stream / buffered transfer***
TCP provides the abstraction of carrying a stream of bytes (octets) in sequence from source to destination, *ignoring message boundaries*. To make transfer efficient data from a stream is *collected* to fill a reasonable large datagram or a large block of data is *divided* into smaller pieces before transmitting.
- ***full-duplex***
Both ends of a TCP connection can simultaneously read and write segments.
- ***multiplexed***
Many applications can share access to a single TCP layer (*multiple conversations*).
- ***reliable***
TCP uses sequence numbers, checksums, timeouts and retransmissions to ensure that the destination receives transmitted segments.
- ***flow-controlled***
A TCP source uses feedback flow control from the network/TCP receiver to *adjust its transmission rate* to the rate supportable in the network/TCP receiver.

Observations concerning the TCP mechanisms:

1. TCP operates above IP, and IP provides only a best-effort datagram transmission service.
2. TCP is an “end-to-end” protocol. The TCP segment is not interpreted by intermediate networks or routers but by the hosts (end stations).
3. TCP is a *byte stream* service. There are no record markers automatically inserted by TCP.
4. End-to-end recovery from unreliability by retransmission leads to the need for *sequence numbering*, both for reordering and for duplicate detection.
5. *Flow control* requires that both ends uniquely agree on which information may now be sent at any time, and which information has been received and acknowledged.
6. In a concatenation of many packet networks, it is possible for a *packet to circulate* in one or more networks for an indeterminate amount of time, only to emerge at a moment when it may cause the maximum *confusion* for the receiving party.
7. *Termination of a full-duplex exchange* of information should be graceful in the sense that both sides should be aware when one side has completed all its transmissions and has received all that it has been sent.
8. Every process should be able to engage in *multiple “conversations”* with the same or other processes without confusion.
9. Because different networks have different maximum packet sizes they can carry, the arrival of information in a particular segment should not have any semantic significance. Rather, the *stream* should be reassembled, and any semantic or syntactic markings should be embedded in the data stream and interpreted by the application receiving it.



The unit of transfer between the TCP software on two hosts is called a *segment*. Segments are exchanged to establish connections, to transfer data, to send acknowledgements, to advertise window sizes, and to close connections.

Each segment is divided into two parts, a header followed by data. The header, known as the TCP header, carries the expected identification and control information.

Fields **Source Port** and **Destination Port** contain the TCP port numbers that identify the application programs at the ends of the connection.

The **Sequence Number** field identifies the position of the *first octet* in the data in the segment in the sender's byte stream (Wrap-around = $2^{32} - 1 = 4.3$ Gbyte).

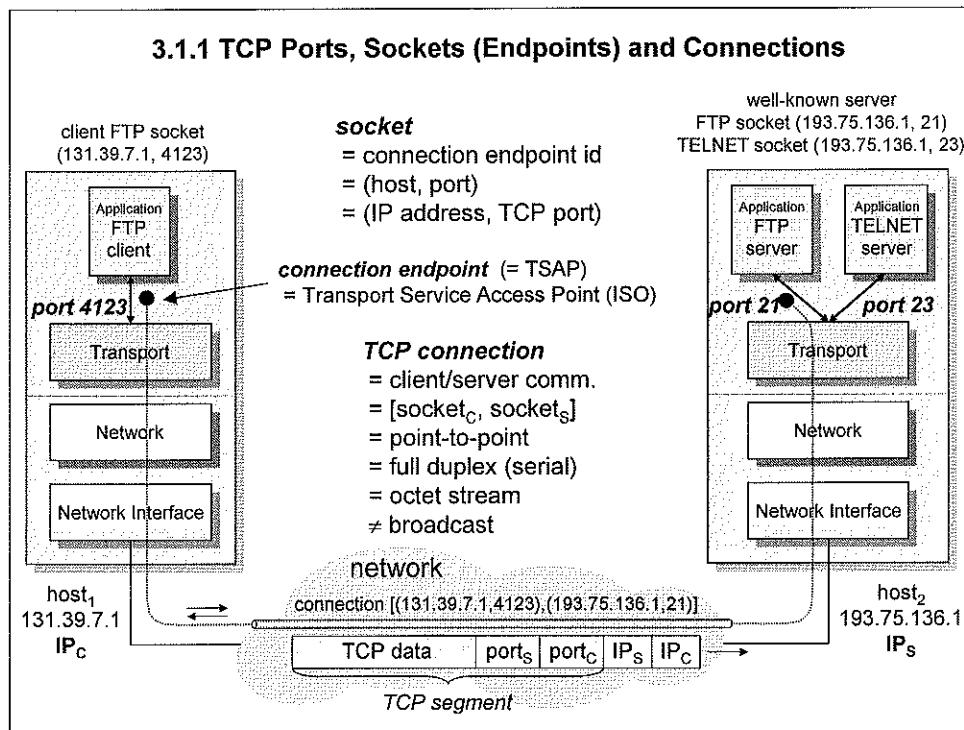
The **Acknowledgement Number** field identifies the number of the *octet* that the source expects to receive *next* (only positive ACK). Note that the sequence number refers to the stream flowing in the same direction as the segment, while the acknowledgement number refers to the stream flowing in the opposite direction as the segment. Because TCP uses *piggybacking*, an acknowledgement travelling from host A to host B may travel in the same segment as data travelling from host A to host B, even though the acknowledgement refers to data sent from B to A.

The **HLEN** field contains an integer (1 ... 15=2⁴-1) that specifies the length of the segment header measured in 32-bit words. It is needed because the *Options* field varies in length, depending on which options have been included. Thus, the size of the TCP header varies depending on the options selected (max. length = 15x4 = 60 bytes)

The 6-bit field marked **Reserved** is reserved for future use.

The 6-bit field **Code Bits** determines the purpose and contents of the segment.

The **Window** field advertises how much data (number of bytes < 65 kbytes) the source of the TCP segment is willing to *receive* every time it sends a segment by specifying the available buffer size of the receiver (example of piggybacking). It contains a 16-bit unsigned integer in network-standard byte order (Big Endian). The maximum window of $2^{16}-1 = 65535$ octets can be made larger by the *window scale option*.



TCP provides a reliable, serial communication path, or *virtual circuit* (connection oriented) between processes exchanging a *full-duplex stream of octets*.

Each process is assumed to reside in an internet host that is identified by an *Internet Protocol address*. Each process has a number of logical full-duplex *ports* through which it can set up and use TCP connections. A host is identified by its IP address and a port of a process is identified by a 16-bit value. Because multiple processes can use the TCP service at the same time, the port numbers included in the TCP header are the key used for *multiplexing* and *demultiplexing* the processes in the TCP segments. Ports are not interpreted across the entire Internet, but only on a single host.

An *endpoint* of a TCP connection is identified by a port-IP address combination and is called a *socket*:

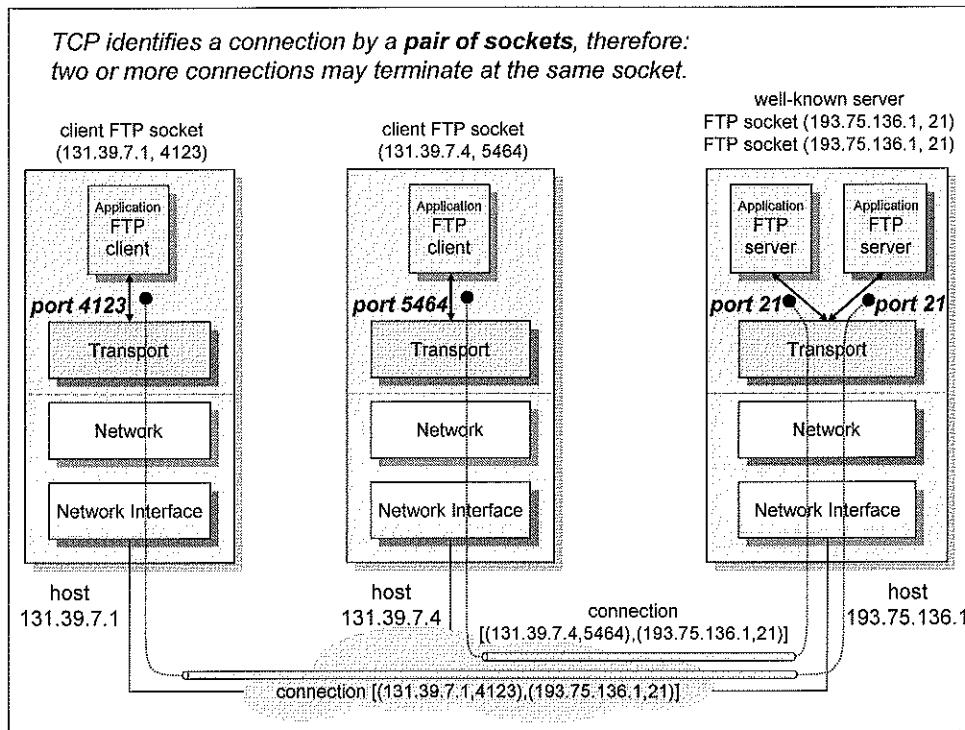
socket = *TCP connection endpoint* = (host IP address, process port number)

The TCP service provides port identifiers to processes on request (dynamic, local) or a process is given *well-known port* at system initialization (static, universal) in the case of common processes and services.

To obtain a TCP service, a *connection* must be explicitly established between a socket on the client host C and the server host S. Connections are identified by the socket identifiers at both ends:

*TCP connection*_{C,S}
 = client/server communication
 = $(socket_C, socket_S)$
 = $(IP_C, TCP port_C, IP_S, TCP port_S)$

A pair of sockets forms a connection that links the processes by a reliable, serial, two-way data stream. All TCP connections are *full-duplex* (traffic in both directions at the same time) and *point-to-point* (each connection has exactly two endpoints). TCP does *not* support multicasting or broadcasting.



IP includes the source and destination IP address in the header of the IP datagram. TCP puts a header at the front of each segment. This includes the source and destination port number. The four numbers (source and destination sockets) identifying a TCP connection are thus included in the IP datagram.

The *port numbers* are used to *keep track of different client/server conversations*. Before a client can use a TCP service, *TCP allocates a unique port number to the client process conversation*. When the client is sending a TCP segment, this port number becomes the "source" port number in the TCP header. Of course the TCP at the server end has assigned a port number of its own for the conversation (see well-known port numbers). The sending TCP entity has to know the port number used by the other end as well. It finds out when the connection starts. It puts this in the "destination" port field. If the server end sends a TCP segment back to the client, the source and destination port numbers will be reversed.

It is important to realize that a *socket may be involved in more than one connection*. In the shown example, socket (192.5.48.1, 21) is used for two connections at the same time. It is *the 'connection' that TCP uses to distinguish among pairs of communicating processes*, not just the local socket identifiers. Two or more TCP connections may terminate at the same socket, because *connections are identified by socket identifiers at 'both' ends: (socket_C, socket_S)*.

Well-known TCP Port Numbers and the Application Layer

Decimal	Keyword	UNIX keyword	Description
7	ECHO	echo	Echo
11	USERS	systat	Active Users
13	DAYTIME	daytime	Daytime
20	FTP-DATA	ftp-data	File Transfer Protocol (data)
21	FTP	ftp	File Transfer Protocol
23	TELNET	telnet	Terminal Connection
25	SMTP	smtp	Simple Mail Transfer Protocol
37	TIME	time	Time
42	NAMESERVER	name	Host Name Server
43	NICNAME	whois	Who Is
49	LOGIN	login	Login Host Protocol
53	DOMAIN	nameserver	Domain Name Server
79	FINGER	finger	Finger
80	HTTP	http	Hypertext Transfer Protocol
161	SNMP	snmp	Simple Network Management Protocol
247-255			Reserved
256-1024			UNIX ports (trusted applications)

- *well-known ports*:(< 1024) reserved for standard services (static) ← server
- *not well-known ports*:TCP service provides port number (dynamic) ← client

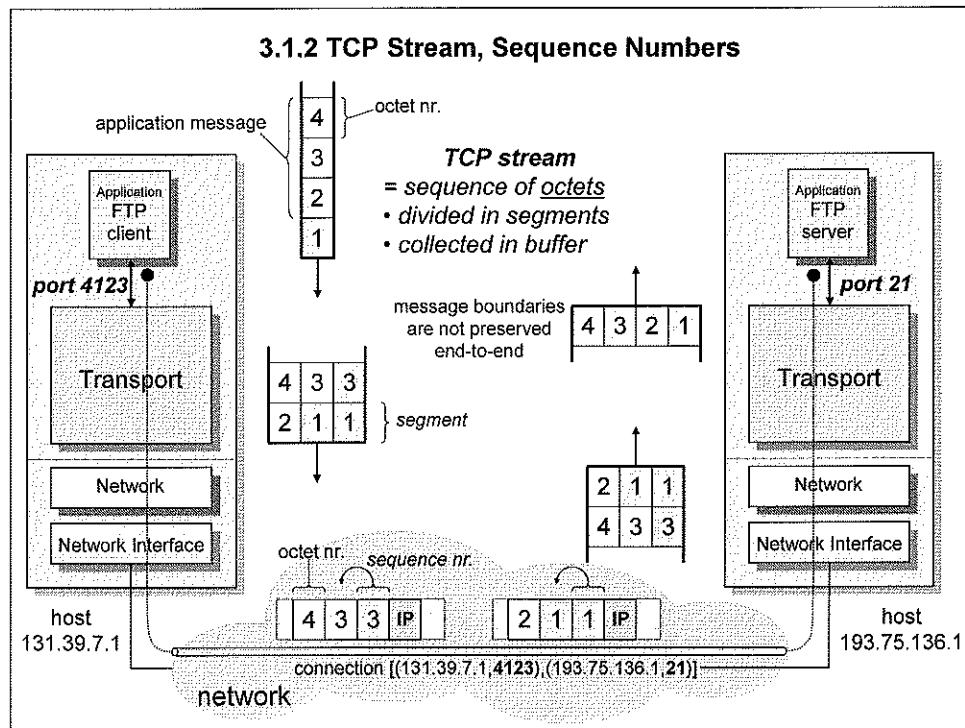
Applications are implemented following the *client / server model*.

How can a client find a *server*? Suppose a client wants to send a file to a host whose IP address is 193.75.136.1. To start the file transfer process, the client needs more than just the IP address. The client has to connect to the FTP server at the other end. How does the client learns the server's port? In general, *application programs* are 'specialized' for a specific set of tasks. Most systems have separate programs to handle file transfers, remote terminal logins, mail, etc. When the client connects to 193.75.136.1, the client must specify that a conversation with the FTP server is wanted. This is done by having "well-known ports" for each server. TCP uses port numbers to keep track of individual conversations.

A *client* that initiates a connection setup normally is assigned a more or less *random port number* (local use), while a *server* that sits waiting for requests is assigned a *specific port number* (global use - many users, specific task).

For example, if a user wants to send a file, he will start a client process called "ftp" on the local host. It will open a connection using some random number, say 1234, for the port number on its end. However it will specify port number 21 for the other end. This is the official (fixed, widely published) port number for the FTP server. Note that there are two different programs involved. The user runs the ftp client on the local host. This is a program designed to accept commands from the local terminal and pass them on to the other end. The program on the remote host is the FTP server. It is designed to accept commands from the network connection, rather than an interactive terminal. There is no need for the client program to use a well-known port number for itself. Nobody is trying to find it. However the servers have to have well-known numbers, so that clients can open connections to them and start sending them commands. *Once a client has contacted a server, the server knows the client's port (it was contained in the TCP segment header) and can reply to it.*

An official set of low-numbered ports are given in "Assigned Numbers" as well-known ports. All systems on an internet, if they provide the associated well-known services, do so using the well-known ports.



TCP is a *stream-oriented* protocol: a continuous flow of octets is transmitted along a connection.

TCP is free to divide the stream of data into segments for transmission without regard of the size of transfer (message) used by application programs. The chief advantage of allowing TCP to choose a division is *efficiency*. It can accumulate enough octets in a *buffer* to make segments reasonable long, reducing the high overhead that occurs when segments contain only a few data octets (IP header + TCP header \geq 40 bytes).

- A TCP connection is an *unstructured byte stream*, not a *message stream*. The structure of a data stream is not considered in the transmission of the TCP stream. *Message boundaries are not preserved end to end*. There is no way for the receiving TCP unit to detect the unit(s) in which the data were written. If the application on one end writes 10 bytes (octets), followed by a write of 20 bytes, followed by a write of 50 bytes, the application on the other end of the connection cannot tell what the size of the individual writes were. The other end may read the 80 bytes in four reads of 20 bytes at a time. One end puts a stream of bytes into TCP and the same, identical stream of bytes appears at the other end.
- Each segment has a *sequence number*. This is used so that the receiving end can make sure that it gets the segments in the right order, and that it hasn't missed any. *TCP does number the octets, not the segments*. So if there are 500 octets of data in each segment, the first segment might be numbered 0, the second 500, the next 1000, the next 1500, etc. The sequence number is an implicit way to number the segments and to indicate their length.
- TCP does not interpret the contents of the octets, this is up to the applications on each end of the connection.

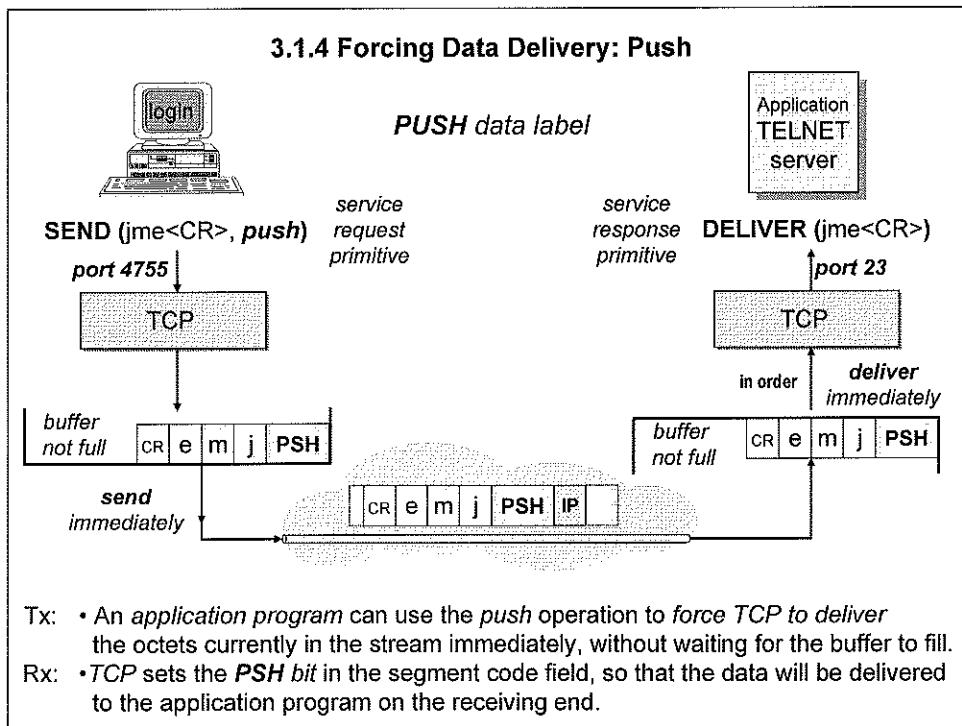
3.1.3 Code Bits

- determine the purpose and contents of the segment
- indicate how to interpret other fields in the header

Position	Name	Function (if bit set)	
10	URG	<i>Urgent</i> Immediate delivery of urgent data (out of order). Urgent Pointer field is valid.	<i>data label</i>
12	PSH	<i>Push</i> Immediate delivery of data (in order), without waiting for other data (no buffering).	
11	ACK	<i>Acknowledge</i> Acknowledgement Number field is valid.	<i>error/flow control</i>
13	RST	<i>Reset</i> Reset the connection.	<i>connection management</i>
14	SYN	<i>Synchronize</i> Establish a connection (request + reply). Synchronize the sequence numbers.	
15	FIN	<i>Final</i> Release a connection. End of octet stream from sender.	

The 6-bit field Code Bits included in the TCP header determines the purpose and contents of the TCP segment. They tell how to interpret other fields in the TCP header.

- URG is set to 1 if the *Urgent Pointer* is in use. The *order of data delivery is not preserved*. The Urgent Pointer is used to indicate a byte offset from the current sequence number at which urgent data are to be found. This facility is in lieu of interrupt messages. This facility is a bare bones way of allowing the sender to signal the receiver without getting TCP itself involved in the reason for the interrupt.
- The ACK bit is set to 1 to indicate that the *Acknowledgement Number* is valid. If ACK is 0, the segment does not contain an acknowledgement so the Acknowledgement Number field is ignored.
- The PSH bit indicates *PUSHed data*. The *order of data delivery is preserved*. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received (which it might otherwise do for efficiency reasons).
- The RST bit is used to *reset a connection* that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection. In general, if a segment with the RST bit on is received, the receiver should break and reset the transport connection.
- The SYN bit is used to *establish a connection*. The *connection request* has SYN = 1 and ACK = 0 to indicate that the piggyback acknowledgement field is not in use. The *connection reply* does bear an acknowledgement, so it has SYN = 1 and ACK = 1. In essence the SYN bit is used to denote *connection request* and *connection accepted*, with the ACK bit used to distinguish between those two possibilities.
- The FIN bit is used to *release a connection*. It specifies that the sender has no more data to transmit. However, after closing a connection, a process may continue to receive data indefinitely. Both SYN and FIN segments have sequence numbers and are thus guaranteed to be processed in the correct order.



TCP implements a *byte-stream service to increase transport efficiency*:

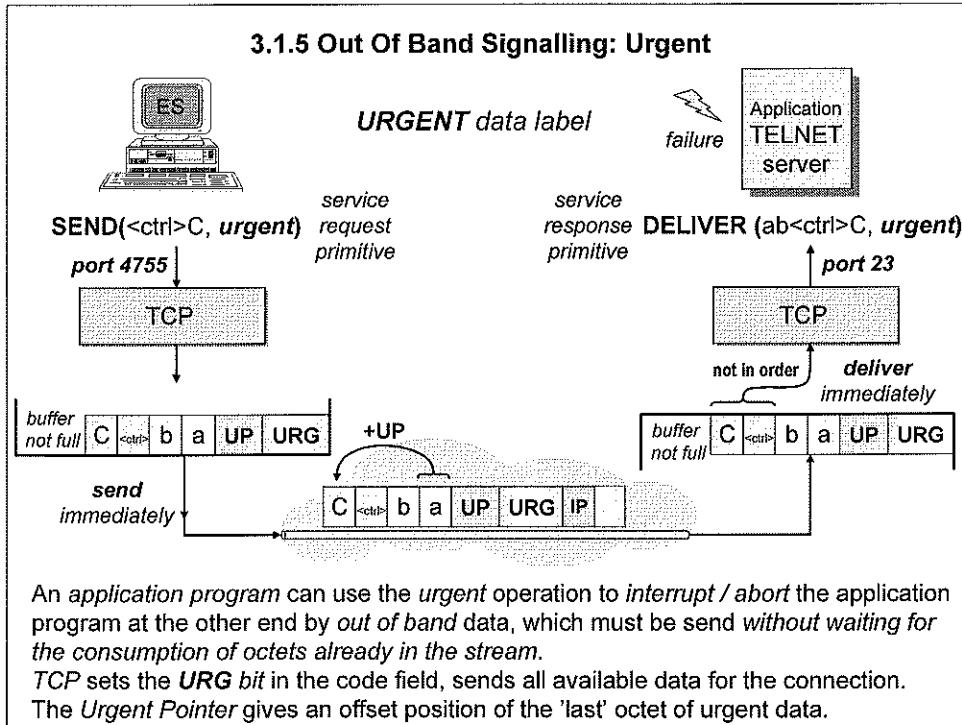
- TCP is free to divide the stream of data into segments for transmission without regard to the size of the messages application programs use. The chief advantage of allowing TCP to choose a division is efficiency (prevent fragmentation or a high retransmission rate due to long packets).
- It can accumulate enough octets in a buffer to make segments reasonably long, reducing the high overhead that occurs when segments contain only a few data octets.

This stream service can be broken by the application by introducing a data label: PUSH or URGENT.

Although buffering improves network throughput, it can interfere with some applications. On an *interactive terminal* to a remote machine, the user expects instant response to each keystroke. If the sending TCP buffers the data, response may be delayed, perhaps for hundreds of keystrokes. Similarly, because the receiving TCP may buffer data before making it available to the application program on its end, forcing the sender to transmit data may not be sufficient to guarantee delivery.

To accommodate interactive users, TCP provides a *push operation* that an application program can use to force delivery of octets currently in the stream without waiting for the buffer to fill. The push operation does more than force TCP to send all segments in the Tx buffer. It also requests TCP to *set the PSH bit in the segment code field (label)*, so the data will be delivered to the application program on the receiving end. The application from an interactive terminal uses the *push function after each keystroke*. Similarly, application programs can force output to be sent and displayed on the terminal promptly by calling the push function after writing a character or line.

	#packets	#bytes
Interactive data flow (telnet):	50%	10% → small packets (even bytes)
Bulk data flow (ftp, e-mail):	50%	90% → large packets (> 512 bytes)



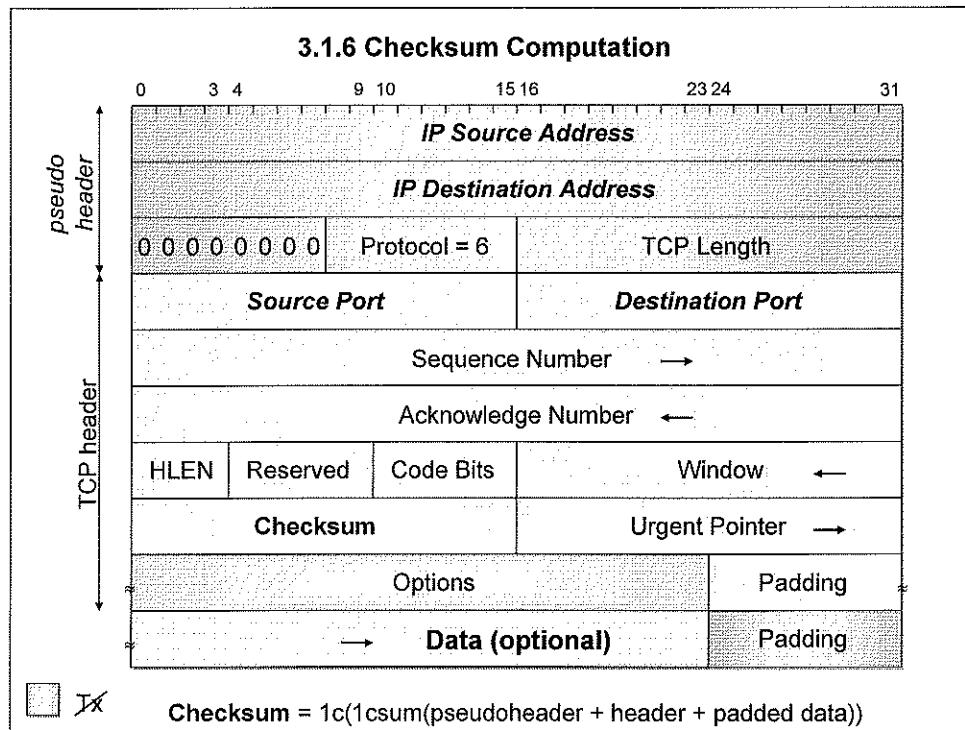
Although TCP is a stream-oriented protocol, it is sometimes important for the program at one end of a connection to send data *out of band*, without waiting for the program at the other end of the connection to consume octets already in the stream.

For example, when TCP is used for a remote login session, the user may decide to send a keyboard sequence that *interrupts* or *aborts* the program at the other end. Such signals are most often needed when a program on the remote machine fails to operate correctly. The signals must be sent without waiting for the program to read octets already in the TCP stream (or one would not be able to abort programs that stop reading input).

Urgent accommodates interrupt transmission for the application, but preserves the TCP connection.

To accommodate out of band signalling, TCP allows the sender to specify data as *urgent*, meaning that the receiving program should be notified of its arrival as quickly as possible, *regardless of its position in the stream* (the order of delivery is not preserved).

The *Urgent Pointer* is a 16-bit value representing an offset from the sequence number of the segment carrying the Urgent Pointer. The sum of the sequence number and the Urgent Pointer is the sequence number of the *last* octet of urgent data.

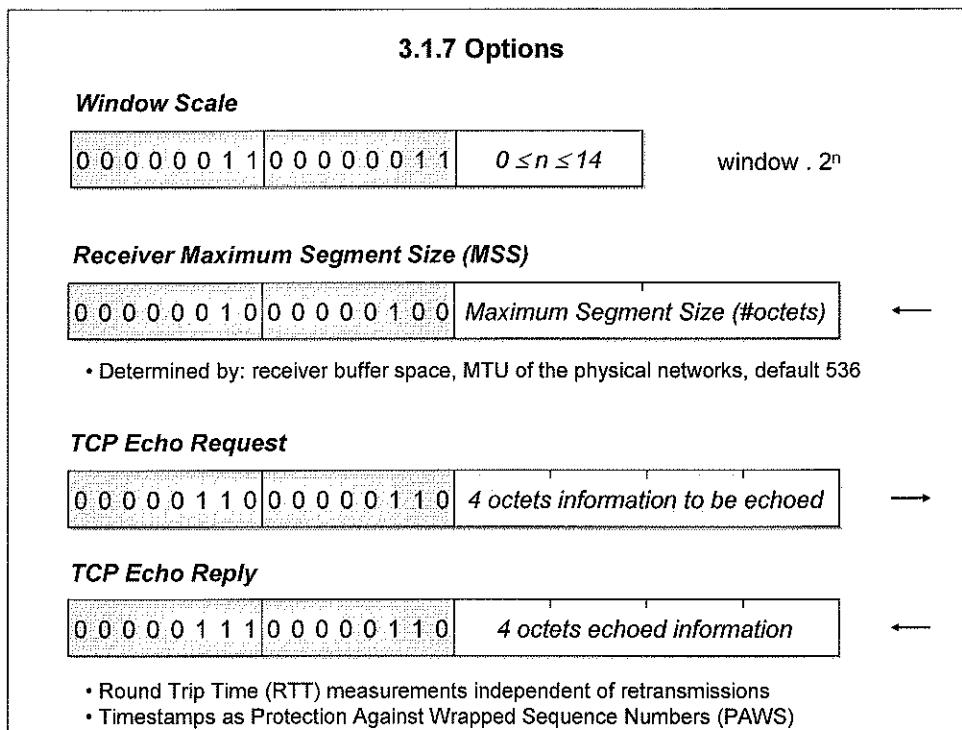


- The *Checksum* field in the TCP header contains a 16-bit integer checksum (end-to-end) used to verify the integrity of the data as well as the TCP header. To compute the checksum, TCP prepends a *pseudo header* to the segment, appends an octet containing zero (at the data) if necessary to pad the segment to a multiple of 16 bits, and computes the 16-bit checksum over the entire result (data included). TCP assumes the checksum field itself is zero for purposes of the checksum computation. As with other checksums, TCP uses 16-bit arithmetic and takes the one's complement of the one's complement sum. At the receiving site, TCP software performs the same computation to verify that the segment arrived intact.
- The sending TCP assigns field *Protocol* the value that the underlying delivery system will use in its protocol type field. For IP datagrams carrying TCP, the value is 6.
- The *TCP Length* field specifies the total length of the TCP segment including the TCP header. At the receiving end, information used in the pseudo header is extracted from the IP datagram that carried the segment and included in the checksum to verify that the segment arrived at the correct destination intact.

The pseudo header and the padding octet are *not transmitted* with the TCP segment, nor are they included (counted) in the TCP Length.

- The purpose of using a *pseudo header* is exactly the same as in UDP. It allows the receiver to verify that the segment has reached its correct destination, which includes both a *host IP address as well as a protocol port number (the socket)*. Both the source and destination IP addresses are important to TCP because it must use them to identify a connection to which the segment belongs. Therefore, whenever a datagram arrives carrying a TCP segment, *IP must pass to TCP the source and destination IP addresses* from the datagram as well as the segment itself.

This calculation is a *violation of the layering concept*, because the TCP and IP layers are not independent of each other. Therefore the TCP/IP protocol is called a *structured (hierarchical) protocol*, not a layered protocol (like OSI). It is included because *TCP uses the socket to identify an application program*.

**Window Scale**

n is the number of bits by which the receiver right-shifts the true receiver window size, to scale it into a 16-bit value to be sent in the TCP header. The maximum value of n is 14 (prevent confusion between sequence numbers), yielding a maximum window of $2^{16+14} = 2^{30} \approx 10^9 = 1$ Gbyte.

Receiver Maximum Segment Size (MSS)

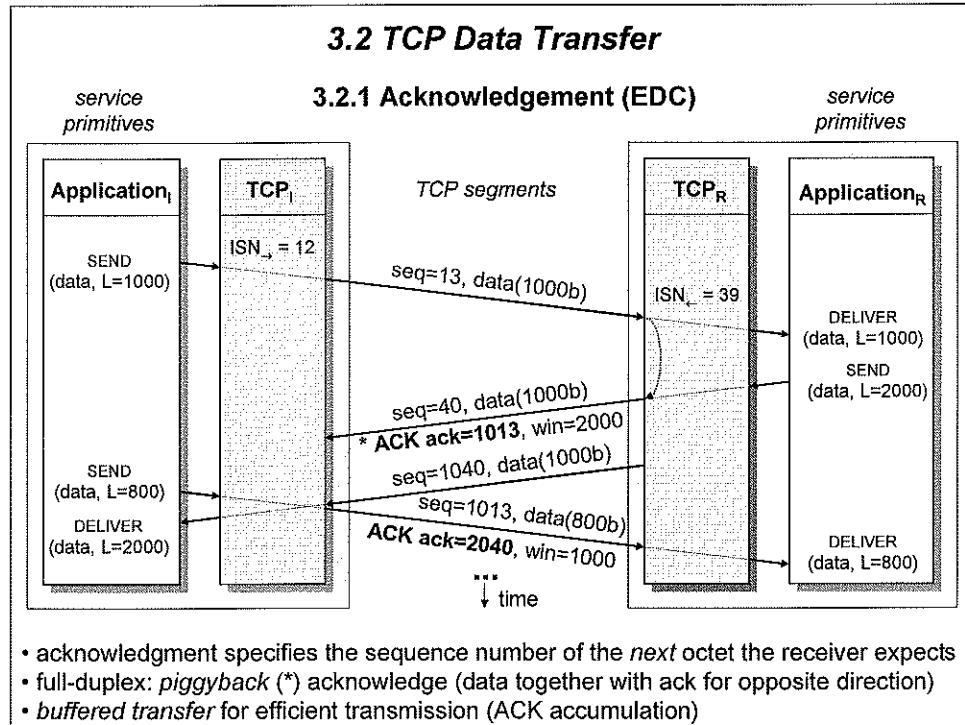
If this option is set in the initial segment (SYN = 1) of a TCP connection, it gives the maximum segment size (MSS) in octets (not including the TCP header), that the sender of the option is willing to receive on the associated TCP connection. If this option is not used, any segment size may be sent to the receiver. The *maximum* value of the MSS coincides with the minimum possible size of the window allowed for flow control purposes ($\text{max MSS} \leq \text{Rx window}$ = available Rx buffer space).

The MSS can be used to adapt TCP connections to network parameters present at connection set up time. If the two TCP endpoints *lie on the same physical network*, TCP uses a MSS resulting in IP datagrams matching the *network MTU*. If the endpoints *do not lie on the same physical network*, MSS of 536 is used (the default size of an IP datagram, 576, minus the standard size of IP and TCP headers). In a general internet environment, choosing a good MSS can be difficult. For a small MSS the overhead of the TCP/IP headers becomes large. Large segments must be fragmented and have a higher error probability. The optimum segment size, S, occurs when the IP datagrams carrying the segments are as large as possible without requiring fragmentation anywhere along the path from source to destination. In practice, finding S is difficult. MSS is used in the congestion window algorithm.

TCP Echo Request/Reply

To use this option, TCP must send a TCP request option in its connection-initiating SYN segment and receive one in the SYN segment of the other TCP entity.

When used for RTT measurements or PAWS, the four bytes define the time at which the segment was transmitted (*timestamp*).



Acknowledgements always specify the sequence number of the next octet, that the receiver expects to receive.

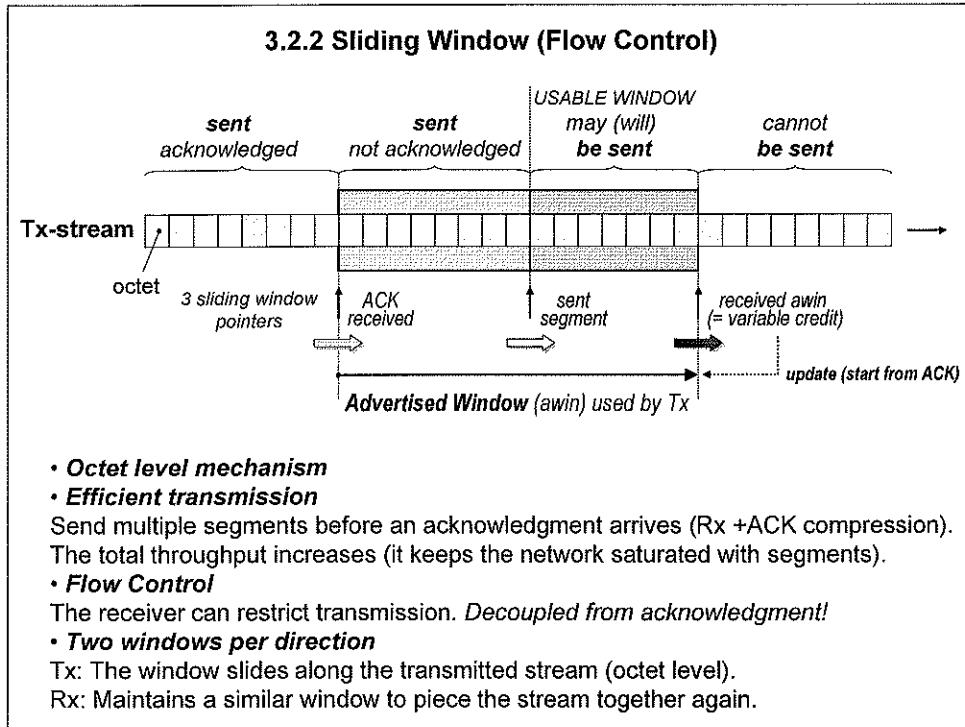
Acknowledge Policy

- The TCP acknowledgement scheme is typically *cumulative*. It reports how much of the stream has accumulated. When data is accepted, the need for acknowledgment is recorded. *TCP waits until a data segment in the reverse direction on which to piggyback the acknowledgment.*
- To avoid a long delay, a *window timer* is set. If the timer expires before an acknowledgment is sent, TCP sends an *empty segment* containing the appropriate acknowledgment number.
- There are *only positive acknowledges*. When an error is detected, no negative acknowledge is send. A *retransmission timeout (RT)* is build in at the transmitter side to retransmit, starting from the wrong segment.

Cumulative acknowledgements have both advantages and disadvantages:

- + acknowledgements are easy to generate
- + unambiguous, produce a minimum overhead
- + lost acknowledgements do not necessarily force retransmission
- the sender does not receive information about all successful transmissions, but only about a single position in the stream that has been received.

To understand why lack of information about all successful transmissions makes the protocol less efficient, think of a window that spans 5000 octets starting at position 101 in the stream, and suppose the sender has transmitted all data in the window by sending five segments. Suppose further that the first segment is lost, but all others arrive intact. The receiver continues to send acknowledgements, but they all specify octet 101, the next highest contiguous octet it expects to receive. There is no way for the receiver to tell the sender that most of the data for the current window has arrived. (Selective Acknowledge is an advisory option).



TCP views the data stream as a *sequence of octets* or bytes that it divides into segments for transmission. Each segment travels across an internet in a single IP datagram.

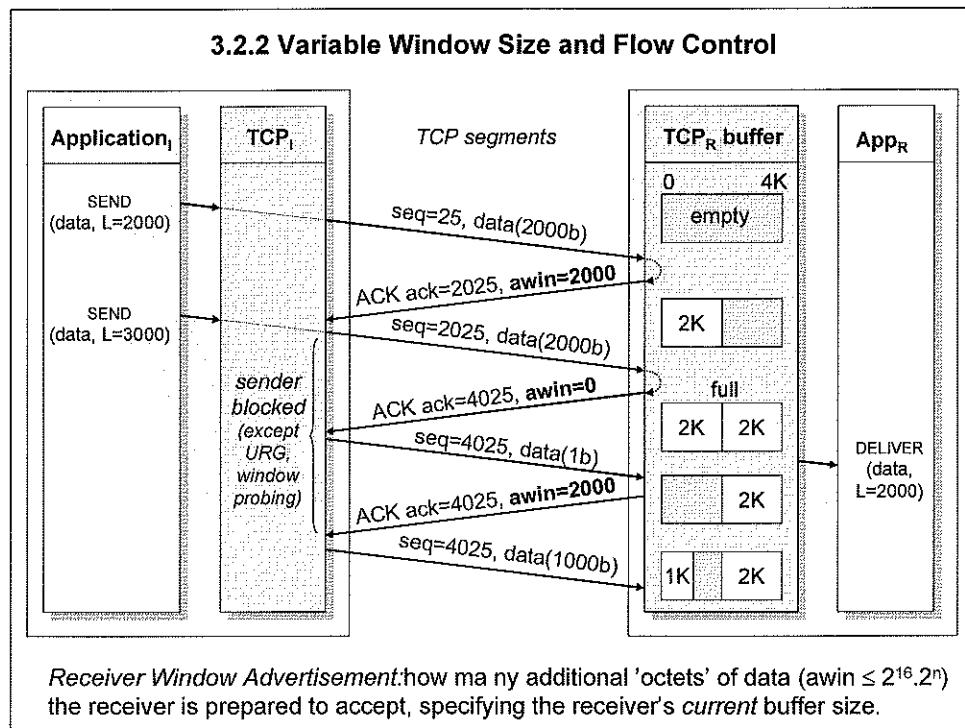
TCP uses a specialized sliding window mechanism to solve two important problems: *efficient transmission* and *flow control*.

- The TCP window mechanism makes it possible to *send multiple segments before an acknowledgement* arrives. Doing so increases total throughput because it keeps the network busy (TCP is not a stop&wait protocol). The receiver is not required to immediately acknowledge incoming segments, but may wait and issue a cumulative acknowledgment for a number of segments (ACK compression).
- The TCP form of a sliding window protocol also solves the end-to-end flow control problem, by allowing the receiver to *restrict transmission* until it has sufficient buffer space to accommodate more data.

The TCP *sliding window mechanism* operates at the octet level, not at the segment or packet level. Octets of the data stream are numbered sequentially, and a sender keeps *three pointers* associated with every connection. The pointers define a sliding window. The *first* pointer marks the left of the sliding window, separating octets that have been sent and acknowledged from octets to be sent and/or acknowledged. A *second* pointer marks the right of the sliding window and defines the highest octet in the sequence that can be sent before more acknowledgements are received. The *third* pointer marks the boundary inside the window that separates those octets that have already been sent from those octets that have not been sent. The protocol software sends all octets in the window limited by the congestion mechanism. The boundary inside the window moves from left to right.

The *usable window* are the octets that are allowed to be sent.

- Tx advances the *trailing edge* of its usable window each time it transmits a segment.
- Tx advances the *leading edge* of its usable window only when it is granted *credits* by *window advertisement* of the receiver. In this credit scheme acknowledgement is decoupled from flow control. A segment may be acknowledged without granting new credit.

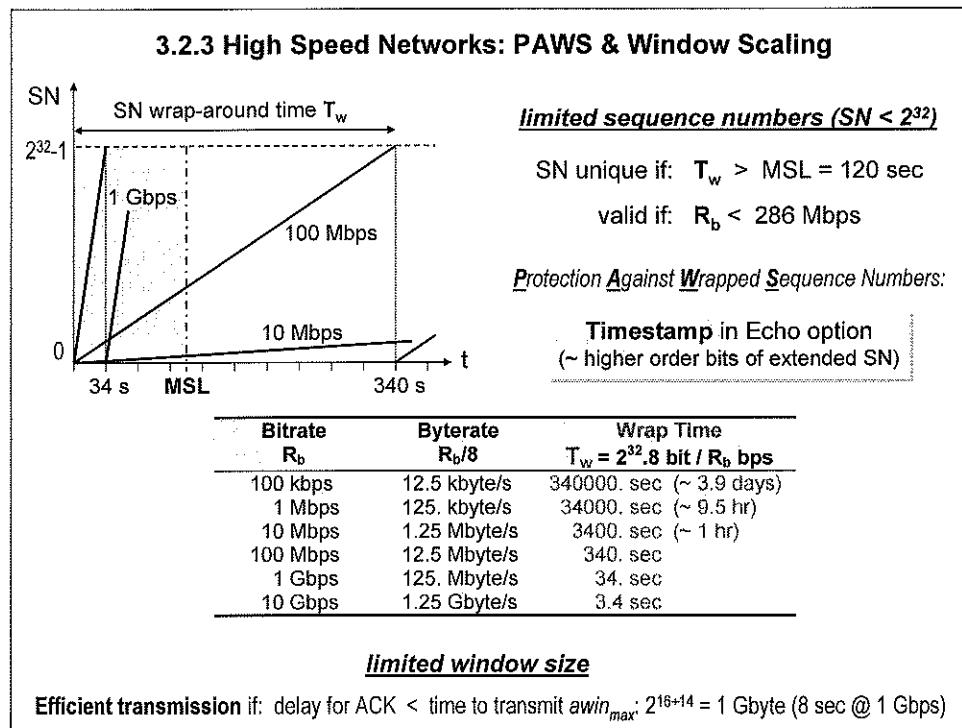


TCP allows the *window size* (octet based) to *vary* over time. Window management in TCP is not directly tied to acknowledgments (as it is in HDLC, frame based, *fixed window size*). Each acknowledgement, which specifies how many octets have been received, contains a *window advertisement* that specifies how many additional octets of data the receiver is prepared to accept, indicating the receiver's current buffer size.

- In response to an *increased window advertisement*, the sender increases the size of its sliding window and proceeds to send octets that have not been acknowledged.
- In response to a *decreased window advertisement*, the sender decreases the size of its window and stops sending octets beyond the boundary. TCP should *not contradict* previous advertisements by shrinking the window past previously acceptable positions in the octet stream. Instead, smaller advertisements accompany acknowledgements, so the window size changes at the time it slides forward.

The advantage of using a variable size window is that it provides *end-to-end flow control*. If the receiver's buffer begins to become full, it cannot tolerate more packets, so it sends a smaller window advertisement. In the extreme case, the receiver advertises a *window size of zero* to *stop all transmissions*, with two exceptions:

- First, *urgent data* may be sent to allow the user to kill the process running on the remote host.
- Second, the sender may send a 1-byte segment periodically (not shown on the figure) to trigger the receiver to announce the next byte expected and the *window size*. This mechanism is called *window probing*. Eventually, one of these 1-byte window probes triggers a response that reports a nonzero advertised window. This advertises that buffer space becomes available and triggers the flow of data again. The TCP standard explicitly provides this option to *prevent deadlock if a window announcement ever gets lost*. The window probing mechanism makes the receiver side as simple as possible: it simply responds to segments from the sender, and it never initiates any activity on its own. It is an example of the protocol design rule: *smart sender/dumb receiver*.



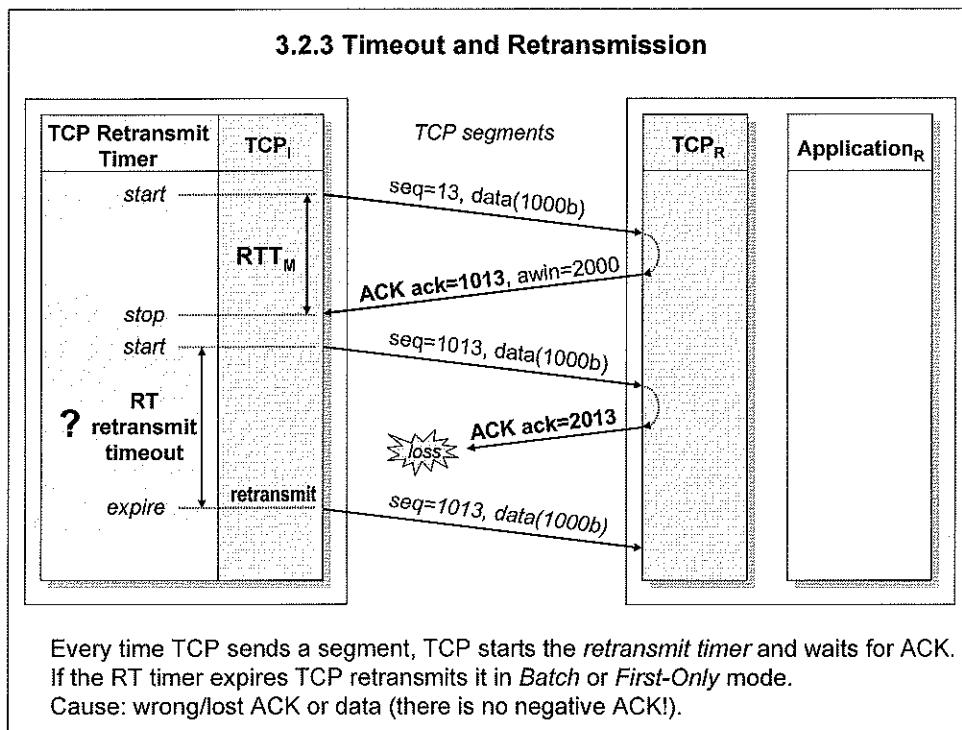
Networks with High-Speed and long ACK-delay (high delay x bandwidth pipe)

The bitrate of the available networks is increasing into Gigabits.

- TCP, has a **limited window size** (max. 65.535 octets), and does not effectively utilize the potential transmitting capacity of a high-speed network if *the delay for ack exceeds the time it takes to transmit a full, maximum-sized window*. The window scale option increases the size (max. $65.535 \times 2^{14} = 2^{30} \approx 10^9$ octets = 8.10^9 bits). A 1 Gbps network needs 8 sec to transmit this window.
- TCP also has **limited sequence numbers**. High-speed networks may *consume the sequence number space to fast*. Confusion between segments is possible, violating the primary function of TCP, which is to deliver data in sequence and with 100% integrity. *Duplicate use of sequence numbers* may occur either because:
 - they are reused during the *same connection* before all previously sent packets using that sequence number have been delivered,
 - a previous instance of the connection has left packets in the network and a *new* instance is started before the *old* packets have disappeared (see problem of initial sequence numbers at connection establishment).

The TCP specification includes a Maximum Segment Lifetime (MSL) of 120 sec, which limits the persistence in the network of any segment and thus of its associated sequence numbers. The TCP sequence number calculation (32-bit) will be unambiguous, only if it takes longer than an MSL for the current sequence number to wrap around: $2^{32} \cdot 8 \text{ bits} / 120 \text{ seconds} = 286$ Mbps.

The TCP Timestamps Option (REC 1323) can be used to support the Protection Against Wrapped Sequence Numbers (PAWS) mechanism. Since the timestamp values are monotonically increasing (in modular arithmetic), the receiver can reject any segment that arrives with an out-of-order timestamp. This might occasionally result in the loss of a good packet (that followed a long-delay path), but it will prevent any state duplicates from creeping in. In effect, the timestamp values act as the high-order bits of an extended sequence number (but not used for ordering or ack).



Retransmit Policy

The TCP transmitter maintains a queue of segments that have been sent but not yet acknowledged. If a retransmit timer (RT) expires before the ACK of a segment, retransmission of the segment is done. A TCP implementation can select between two retransmit strategies:

- *Batch*: One retransmit timer for the entire queue. If the timer expires, retransmit ‘all’ segments in the queue and reset the timer.
- *First-only*: One retransmit timer for the entire queue. If the timer expires, retransmit the segment at the ‘front’ of the queue and reset the timer.

Example

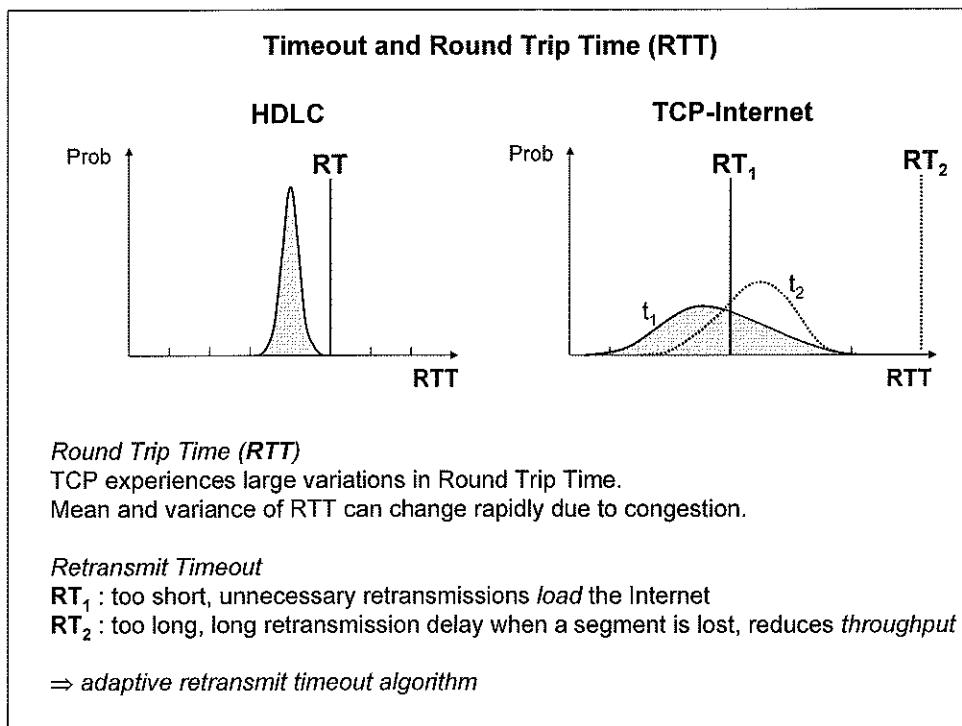
Think of a window that spans 5000 octets starting at position 101 in the stream, and suppose the sender has transmitted all data in the window by sending five segments. Suppose further that the first segment is lost, but all others arrive intact. The receiver continues to send acknowledgements, but they all specify octet 101, the next highest contiguous octet it expects to receive.

When a timeout occurs (mostly a network problem: congested router) at the sender’s side, the sender must choose between two potentially inefficient schemes.

- *Batch*: It may choose to retransmit all five segments instead of the one missing segment. Of course, when the retransmitted segment arrives, the receiver will have correctly received all data from the window, and will acknowledge that it expects octet 5101 next. However, that acknowledgement may not reach the sender quickly enough to prevent the *unnecessary retransmission* of other segments from the window.

- *First-only*: If the sender retransmits only the first unacknowledged segment, it must wait for the acknowledgement before it can decide what and how much to send. Thus, it reverts to a simple positive acknowledgement protocol and may lose the advantages of having a large window.

→ by changing the size of the receiver window, the size of transmitted data can be changed: *congestion window*



It is much more difficult in the internet transport layer to select a good *retransmit timeout RT*, than in a data link protocol.

- In the HDLC case, *the expected delay is highly predictable* (i.e., has a low variance), so the timer (RT) can be set to go off just slightly after the acknowledgement is expected, as shown in the figure. Since acknowledgements are rarely delayed in the data link layer, the absence of an acknowledgement at the expected time generally means the frame or the acknowledgement has been lost.
- TCP is faced with a radically different environment. The probability density function for the time it takes for a TCP acknowledgement to come back has a *large variance*. Determining the Round-Trip Time (RTT) to the destination is tricky. Even when it is known, deciding on the timeout interval is also difficult. If the timeout is set too short, say RT₁, unnecessary retransmissions will occur, clogging the internet with useless packets. If it is set too long (RT₂), performance will suffer due to the long retransmission delay whenever a packet is lost. Furthermore, the mean and variance of the acknowledgement arrival distribution can change rapidly within a few seconds as congestion builds up or is resolved.

The solution is to use a *highly dynamic algorithm* that constantly *adjusts the retransmit timeout (RT) interval*, based on continuous measurements of network performance.

Adaptive Retransmit Timeout (RT) Algorithm

Jacobson

Estimated RTT: $RTT_E(i) = \alpha \cdot RTT_E(i-1) + (1-\alpha) \cdot RTT_M(i)$ M=measure

Mean Deviation: $D(i) = \alpha \cdot D(i-1) + (1-\alpha) \cdot |RTT_E - RTT_M|$

Retransmit Timeout: $RT = RTT_E + 4 \cdot D$

smoothing factor $\alpha \rightarrow 1$ (7/8 typical) makes the weighted average RTT_E immune to changes that last a short time (e.g. 1 segment with long delay)

Karn

Do not update RTT on *retransmitted* segments (prevent wrong estimate).

Now use *transmit timer back off*: until no retransmission.

$$RT(i) = 2 \cdot RT(i-1)$$

- The algorithm generally used by TCP to *adjusts the retransmit timeout* interval, is due to Jacobson (1988) and works as follows. For each connection, TCP maintains a variable, RTT_E , that is the best current estimate of the round-trip time to the destination in question. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long. If the acknowledgement gets back before the timer expires, TCP measures how long the acknowledgement took, say, RTT_M . It then updates RTT_E according to the formula:

$$RTT_E(i) = \alpha \cdot RTT_E(i-1) + (1 - \alpha) \cdot RTT_M$$

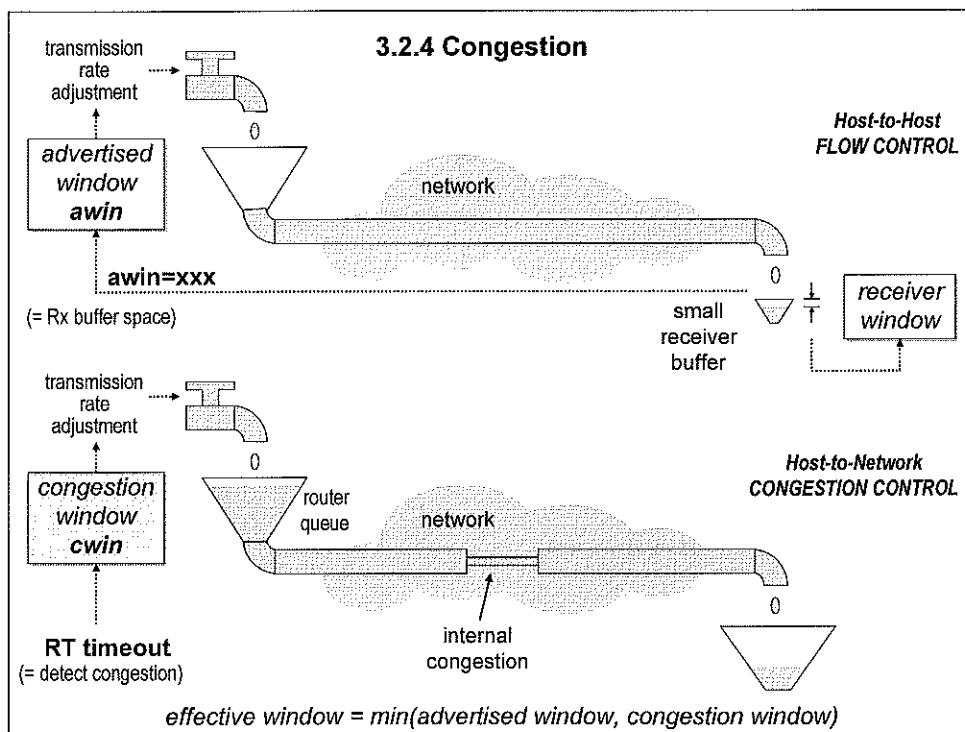
where α is a smoothing factor that determines how much weight is given to the old value. Typically $\alpha = 7/8$. The older the measurement, the less weight it is given (when the expression is expanded). This is called exponential average.

Jacobson suggested using the *mean deviation* as a cheap estimator of the *standard deviation*. This algorithm requires keeping track of another smoothed variable, D , the deviation. Whenever an acknowledgement comes in, the difference between the expected and observed values, $|RTT_E - RTT_M|$ is computed. A smoothed value of this is maintained in D by:

$$D(i) = \alpha \cdot D(i-1) + (1 - \alpha) \cdot |RTT_E - RTT_M|$$

where α may or may not be the same value used to smooth RTT_E . While D is not exactly the same as the standard deviation, it is good enough and Jacobson showed how it could be computed using only integer adds, subtracts, and shifts, a big plus. Most TCP implementations use this algorithm.

- One problem that occurs with the dynamic estimation of RTT_E is what to do when a segment times out and is sent again. When the acknowledgement comes in, it is unclear whether the acknowledgement refers to the first transmission or a later one. Guessing wrong can seriously contaminate the estimate of RTT_E . Karn, an amateur radio enthusiast interested in transmitting TCP/IP packets by ham radio, gives a simple algorithm: do not update RTT_E on any segments that have been retransmitted. Instead, the *retransmit timeout* is '*doubled*' on each failure (*timeout*) until the segments get through the first time (or a set max. is reached). This is *binary exponential backoff* (see CSMA/CD in Ethernet). Most TCP implementations use it.



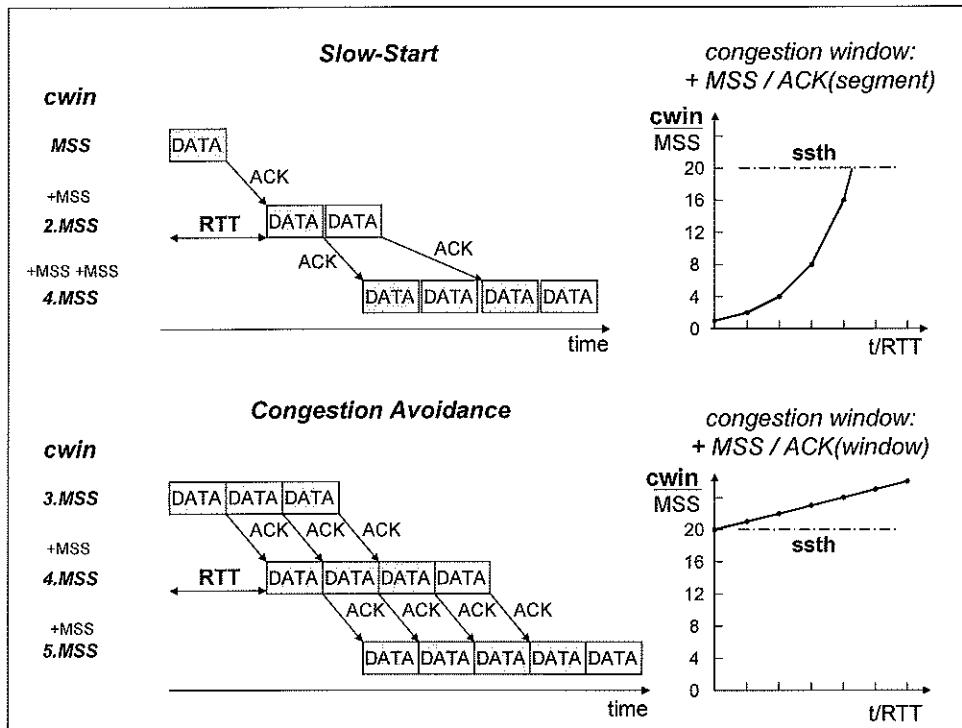
When the load offered to a network is more than it can handle, congestion builds up. Although the network layer also tries to manage congestion, most of the work is done by TCP because *the real solution to congestion is to slow down the data rate*. The idea is not to inject a new packet into the network until an old one leaves (i.e. is delivered). TCP attempts to achieve this goal by *dynamically manipulating the window size* (the number of bytes the sender may transmit).

The first step in managing congestion is *detecting* it. A timeout by a lost packet could be caused by (1) *noise* on a transmission line or (2) packet discard at a *congested* router. Nowadays, packet loss due to transmission errors is relatively rare because most long-haul trunks are fiber (although wireless networks are a different story). Consequently, most transmission timeouts on the internet are due to congestion. TCP assumes that *timeouts are caused by congestion* and monitors timeouts.

When a connection is established, a suitable window size has to be chosen:

- **Receiver window:** The receiver specifies a window based on its buffer size. If the sender sticks to this window size, a buffer overflow at the receiver is prevented. The first figure shows a thick pipe leading to a small-capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost.
- **Congestion window:** In the second figure the limiting factor is not the bucket capacity, but the internal carrying capacity of the network. If too much water comes in too fast, it will back up and some will be lost (in this case by overflowing the funnel).

The internet solution is to realize that two potential problems exist - network capacity and receiver capacity - and to deal with each of them separately. To do so, each sender maintains two windows: the *receiver window* granted by the receiver and a second window, the *congestion window*. Each reflects *the number of bytes the sender may transmit*. The number of bytes that may be sent is the minimum of the two windows. Thus the effective window is the minimum of what the sender thinks is all right and what the receiver thinks is all right.



In normal operation TCP may send several segments at once to transmit a large usable window. **Congestion detection:** Once packet loss and *retransmission* occurs, TCP must react. Since packet loss is often caused by congestion, it is a good indication that the network is overloaded. **Reaction:** The sender should *reduce its load* to the network.

Slow-Start algorithm (Jacobson and Karel, 1988)

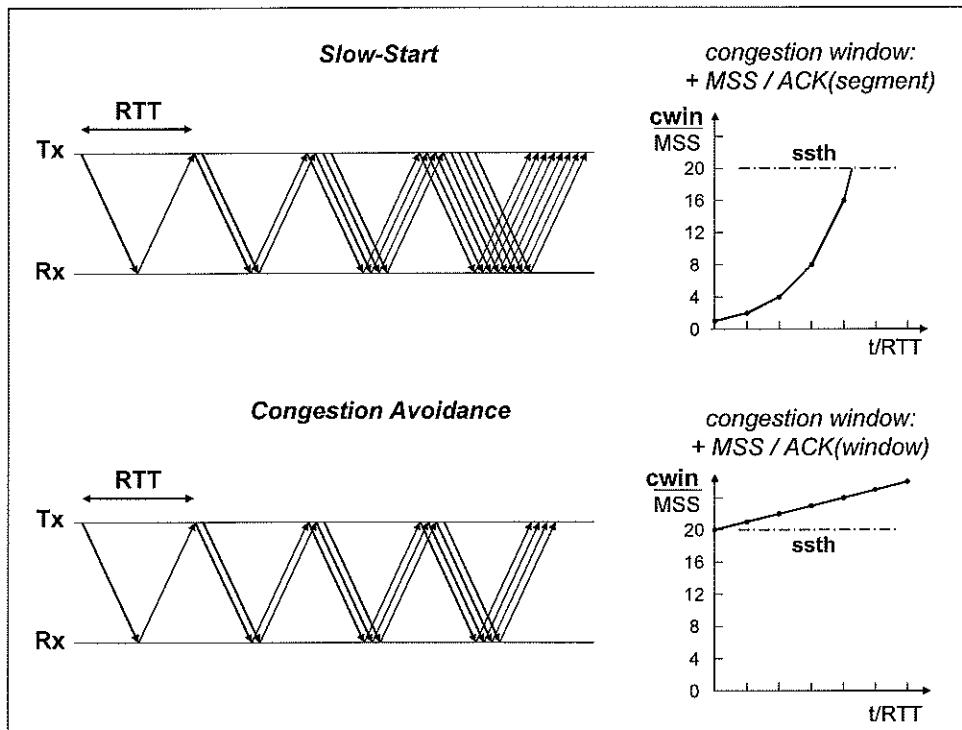
A TCP sender initiates a slow-start phase at the beginning of a connection, and whenever packet loss implies that the network may be congested. A TCP sender is limited to send no more data than is allowed by the congestion window (and is also still limited by the usable receiver window). The size of the congestion window is an estimate of how much data can be sent without congesting the network.

At the beginning of a slow-start phase, the congestion window 'cwin' is set equal to the maximum segment size (MSS). Thus, only one packet can be sent. Then, each time an ACK is received, the congestion window is increased by the MSS value. This can cause an exponential growth in the size of the congestion window, and thus in the number of packets sent. Each packet sent results in the return of an ACK. So, if the congestion window was $N \times \text{MSS}$, then one RTT later N ACKs will be received, causing the MSS to be added to the congestion window N times. The new congestion window will be $2N \times \text{MSS}$, and so after each RTT the window size will double.

This exponential growth phase is stopped (to prevent overload of the network again) at the *slow-start threshold* 'ssth'.

Connection Avoidance

To avoid increasing the congestion window to quickly and causing additional congestion, the congestion avoidance phase is entered when the congestion window increases above the slow-start threshold 'ssth'. The congestion window increases by 1 MSS only if *all* segments in the window have been acknowledged.



The congestion window 'cwin' is the congestion control's counterpart to flow control's advertised window. TCP can have no more than the minimum of 'cwin' and the advertised window 'awin' number of bytes of unacknowledged data.

The 'awin' is send by the receiving side of the connection, but there is no one to send a suitable 'cwin'. The transmitter sets the 'cwin' based on the level of congestion it perceives to exist in the network. This involves decreasing 'cwin' when the level of congestion goes up and increasing 'cwin' when the level of congestion goes down. This mechanism is called: *additive increase / multiplicative decrease*.

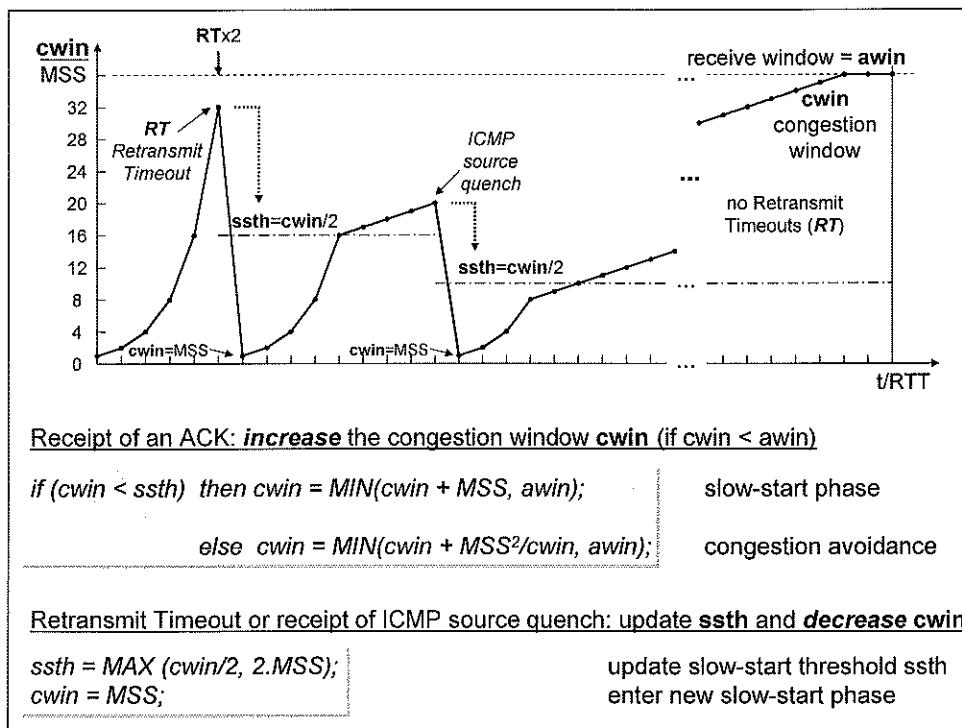
A congestion control strategy is able to increase the congestion window, to take advantage of newly available capacity in the network. This is done by the *slow-start* and the *congestion avoidance* procedures.

The key question is: *how does the source determine that the network is congested and that it should decrease the congestion window?*

Congestion is a condition of severe delay, caused by an overload of datagrams at one or more switching points (routers). When congestion occurs, delays increase and the router begins to enqueue datagrams until it can route them. Each router has finite storage capacity and datagrams compete for that storage (in a datagram based internet, there is no preallocation of resources to individual TCP connections). In the worst case, the router reaches capacity and starts to *drop datagrams*.

Endpoints do not usually know the details of where congestion has occurred or why. To them, congestion simply means increased delay. Unfortunately, most transport protocols use *timeout and retransmission* so they respond to increased delay by retransmitting datagrams. Retransmissions aggravate congestion instead of alleviating it. *The increased traffic will produce increased delay*, leading to increased traffic, and so on, until the network becomes useless. This condition is *congestion collapse*.

The main reason packets are not delivered, and a *RT timeout* results, is that a packet was dropped due to congestion.



Routers watch queue lengths and use techniques like ICMP source quench to inform hosts that congestion has occurred. TCP can help avoid congestion by reducing transmission rates automatically whenever this ICMP message is received or large delays (congestion) occur.

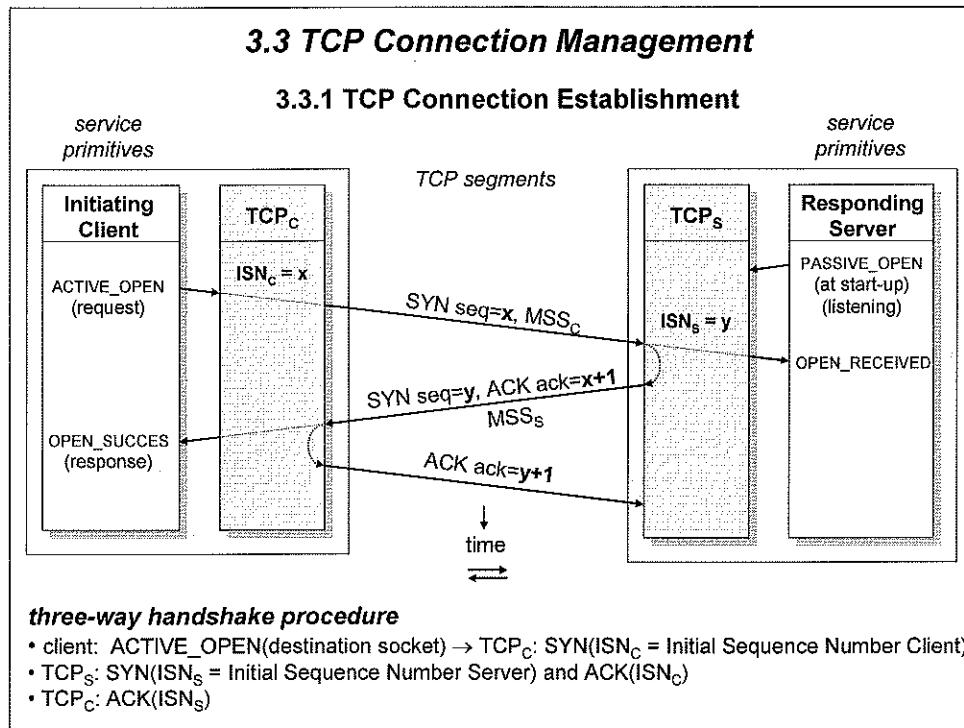
Decrease Congestion Window

When a *RT timeout* occurs or an *ICMP Source Quench* packet comes in, the slow-start threshold 'ssth' is set to half of the 'actual' congestion window 'cwin', and the congestion window is reset to one maximum segment size (MSS). This strong reduction is needed because it is easy to drive a network into saturation, but hard for the network to recover. The 'ssth' remembers the actual congestion window being used by the slow-start increase procedure.

Increase Congestion Window

Slow-start is then used to determine the network capacity, except that *exponential* grow stops when the slow-start threshold 'ssth' is hit. If the congestion window would be left there, each time a timeout is detected the window is cut in half. Soon the window size would be stuck at one packet, TCP would become a stop-and-wait protocol. Why the name 'slow'? This start-up is slower than the case where the source sends as many packets as the advertised window allows. (Routers may not be able to consume this burst of packets.)

Congestion avoidance is used when crossing the threshold 'ssth'. Successful transmissions grow the congestion window *linearly* (by one maximum segment for each burst) instead of one per segment. In stead of waiting for an entire 'cwin' of ACKs to add one MSS, TCP adds a little for each ACK that arrives. In effect, this algorithm is guessing that it is probably acceptable to cut the congestion window in half, and then it gradually works its way up from there.



TCP is a *connection-oriented* protocol.

Service primitives used in *client-server environment*.

PASSIVE_OPEN: The *server* indicates its readiness to accept a connection request. The term 'passive' is used to indicate that it is ready to receive a connection request, rather than actively initiate the setting up of a connection.

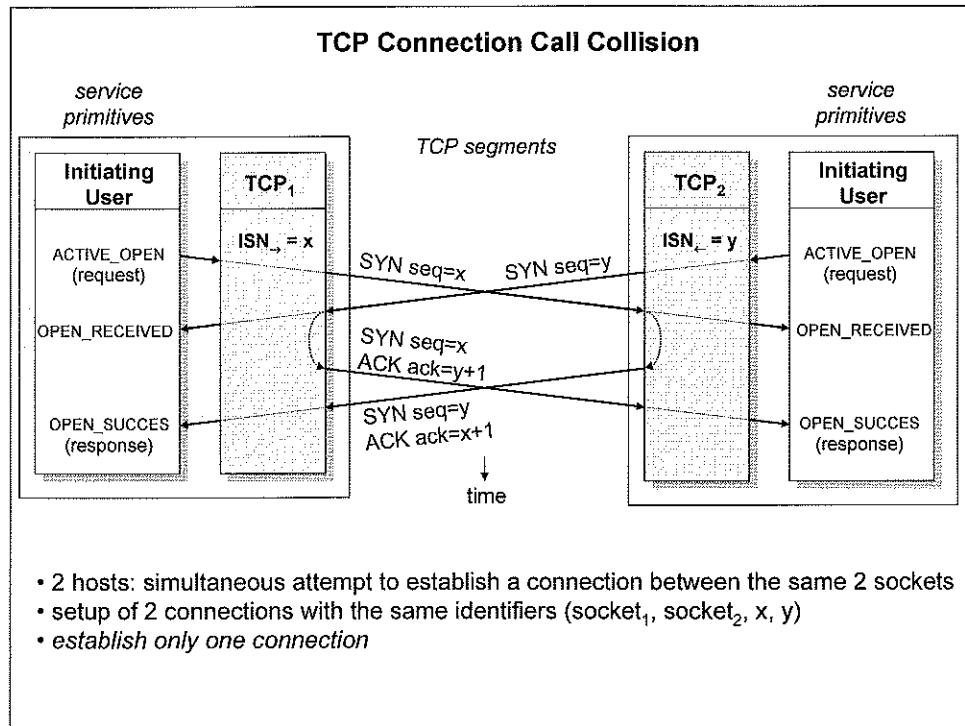
ACTIVE_OPEN: In a client-server interaction the *client* always initiates the setting up of a connection with this request, specifying the socket of the destination and the maximum segment size **MSS_c** for the client receiver.

TCP uses a **three-way handshake** to establish a connection.

- The first segment of a handshake can be identified because it has only the SYN bit set in the code field (ACK = 0).
- The second message has both the SYN bit and ACK bit set, indicating that it acknowledges the first SYN segment as well as continuing the handshake.
- The final handshake message is only an acknowledgement and is merely used to inform the destination that both sides agree that a connection has been established.

Note that a SYN segment consumes 1 byte of sequence space ($x+1, y+1$) so it can be acknowledged unambiguously.

The three-way handshake is both *necessary and sufficient for correct synchronization* between the two ends of the connection. TCP builds on an *unreliable packet delivery service*, so messages can be lost, delayed, duplicated, or delivered out of order. Thus, the protocol must use a timeout mechanism and *retransmit lost requests*. Trouble arises if retransmitted and original requests arrive while the connection is being established, or if retransmitted requests are delayed until after a connection has been established, used, and terminated. A three-way handshake (plus the rule that TCP ignores additional requests for connection after a connection has been established) solves these problems.



- 2 hosts: simultaneous attempt to establish a connection between the same 2 sockets
- setup of 2 connections with the same identifiers (socket₁, socket₂, x, y)
- establish only one connection

Simultaneous Open - Not client-server model

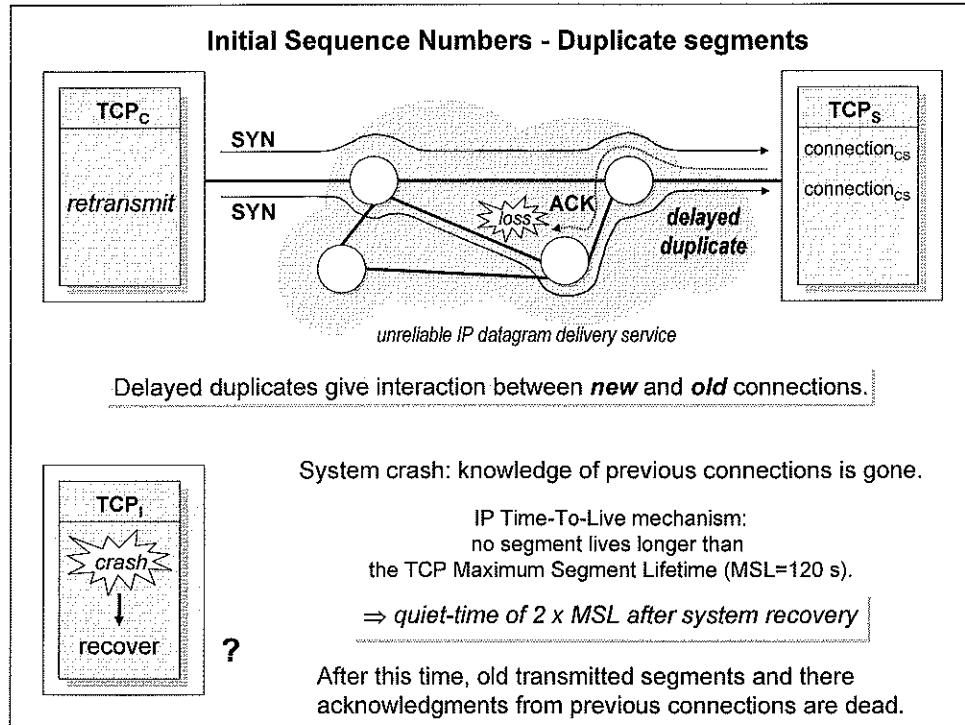
It is possible, although improbable, for two applications (not based on the client-server model) to both perform an 'active open' to each other at the same time (same hosts, same port numbers). Each end must transmit a SYN, and the SYNs must pass each other in the network. It also requires each end to have a local port number that is well known to the other end.

Also in this situation the three-way handshake works.

- Both TCP entities send a SYN segment at the same time. Both hosts simultaneously attempt to establish a connection between the same two sockets.
- When each end receives the SYN, it "resends the SYN" and acknowledges the received SYN. Each side returns an ACK segment acknowledging the appropriate sequence number.
- When each end receives the SYN plus the ACK, both sides of the connection are set up. Each end can start to send data independently. The result of these events is that *just one connection is established*, not two because connections are identified by their endpoints (sockets). If the first setup results in a connection identified by (x,y) and the second one does too, only one table entry is made, namely for (x,y).

Thus, a connection can be established from either end or from both ends simultaneously. Once the connection has been established, data can flow in both directions equally well (full-duplex). There is no master or slave.

Remark: A simultaneous open requires the exchange of four segments, one more than the normal three-way handshake. Both ends act as client and server.



Establishing a connection sounds easy, but it is actually surprisingly tricky. At first glance, it would seem sufficient for one transport entity to just send a SYN segment to the destination and wait for an ACK reply. The problem occurs when *the network can lose, store, and duplicate packets*.

- Delayed duplicates give interaction between new and old connections

Imagine a subnet that is so congested that acknowledgements hardly ever get back in time, and *each packet times out and is retransmitted two or three times*. Suppose that the subnet uses datagrams inside, and every packet follows a different route. Some of the packets might get stuck in a traffic jam inside the subnet and take a long time to arrive, that is, they are stored in the subnet and pop out much later.

The worst possible nightmare is as follows. A user establishes a connection with a bank, sends messages telling the bank to transfer a large amount of money to the account of a not-entirely-trustworthy person, and then releases the connection. Unfortunately, each packet in the scenario is duplicated and stored in the subnet. After the connection has been released, all the packets pop out of the subnet and arrive at the destination in order, asking the bank to establish a new connection, transfer money (again), and release the connection. The bank has no way of telling that these are duplicates. It must assume that this is a second, independent transaction (with same TCP connection), and transfers the money again.

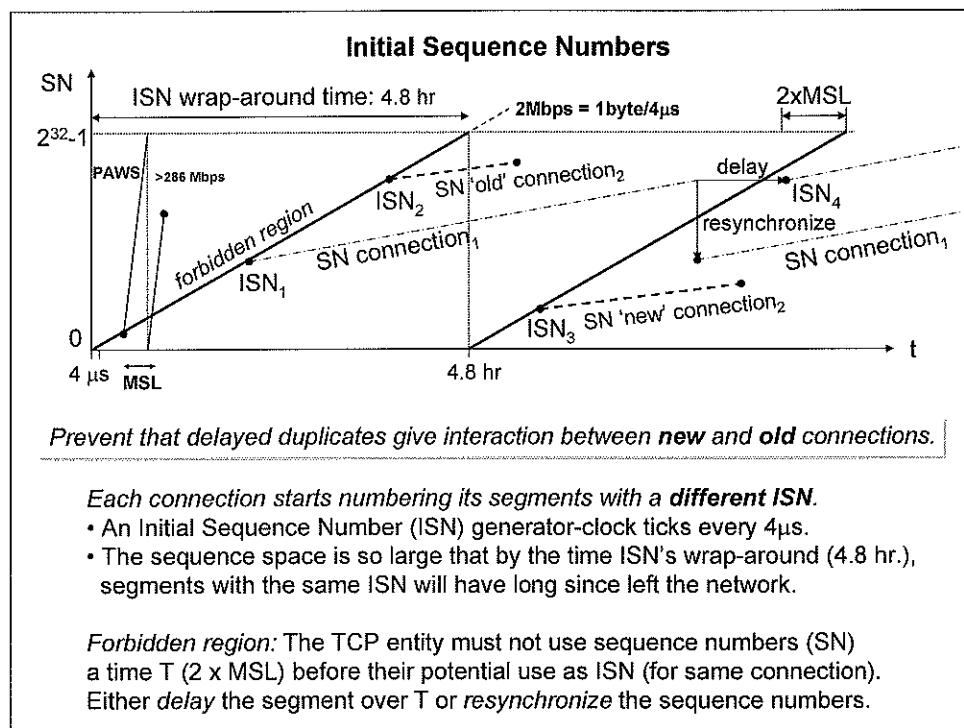
The crux of the problem is the existence of *delayed duplicates*.

- System crash

If a *host crashes* and loses its memory, it will no longer know which connections were active. Sequence number may be inappropriately reused. The standard TCP specification requires that a quiet-time be enforced after system recovery. This is specified twice the Maximum Segment Lifetime ($2 \times MSL$).

- Connection Closed

A similar pause of $2 \times MSL$ is recommended after a connection is closed to ensure that both sides can be assured that the close has been successful.

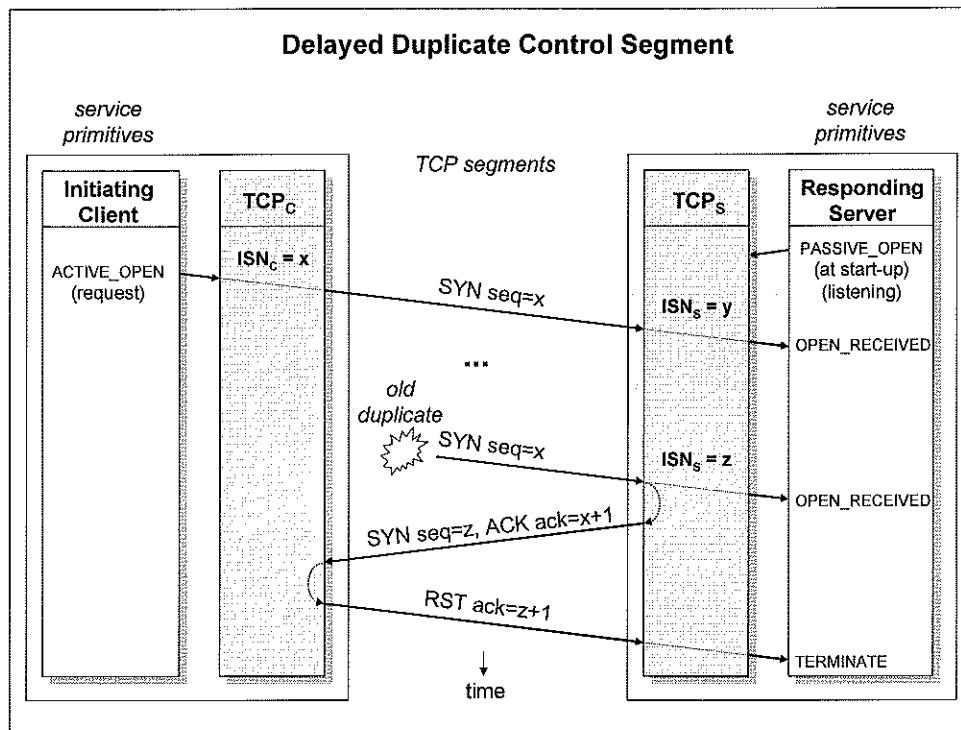


There is no restriction on the reuse of a pair of sockets (= connection). New instances (incarnations) of old connections may encounter problems if the new incarnation uses a sequence number recently used by an earlier incarnation of a connection. Connections must be reusable, especially after a crash of the host, without introducing problems with reuse of sequence numbers (which are finite and have a wrap-around problem) and subsequent confusion over which segments contain the data associated with the most recent incarnation.

The **Maximum Segment Lifetime (MSL = 120 sec)** is the maximum amount of time any segment can exist in the network before being discarded (due to the TTL mechanism of the IP datagrams that carry the TCP segments).

An *initial sequence number (ISN)* generator is ticking at the rate of one tick per 4 μ s. The ISN space cycles approximately every 4.8 hours. It is assumed that segments do not stay in the network for this long so that, by the time the ISN has recycled, any previous use of the segment sequence number will have long since left the network.

- For any data rate less than the clock rate of the ISN generator (1 byte/ $4\mu s$ = 2Mbps), the curve of actual sequence numbers used versus time will eventually run into the *forbidden region* from the left. The greater the slope of the *actual* sequence number curve (of a connection), the longer this event is delayed. Before sending a segment, the transport entity must check if it is about to enter the forbidden region, and if so, either *delay* the segment for $2 \times MSL$ sec or *resynchronize* the sequence numbers.
 - After closing a connection or a system crash, a connection cannot be reused for a time $2 \times MSL$.
 - Very *high speed networks* (> 286 Mbps = $2^{32} \times 8\text{bit}/120\text{sec}$) can consume the SN space at high speeds and enter the forbidden region from underneath (problems *within the same connection*: PAWS). In this case TCP timestamps in the TCP Echo and TCP Echo Reply options are used: any arriving segment whose timestamp is earlier than the timestamp of the most recently accepted segment should be discarded as an old duplicate.



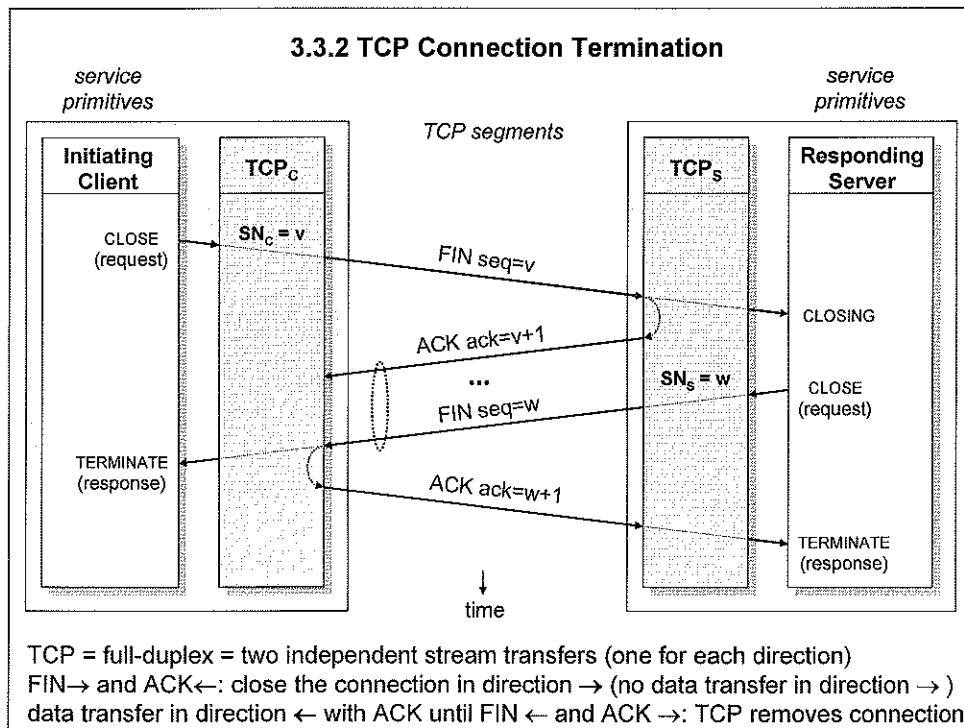
- *Normal three-way handshake operation*

The *three-way handshake* accomplishes two important functions. It guarantees that *both sides are ready to transfer data (and that they know they are both ready)*, and it allows both sides to *agree on initial sequence numbers*. Sequence numbers are sent and acknowledged during the handshake.

The two hosts can agree on sequence numbers for two streams (full-duplex) after only three messages. The client host C that initiates a handshake, passes its initial sequence number, x, in the sequence field of the first SYN segment in the three-way handshake. The responding server host S, receives the SYN, records the sequence number, and replies by sending its initial sequence number in the sequence field as well as an acknowledgement that specifies that it expects octet x+1 (acknowledgement of the ISN in the C → S direction). In the final SYN message of the handshake, C acknowledges the ISN in the S → C direction.

- *The three-way handshake in the presence of delayed duplicate control segments*

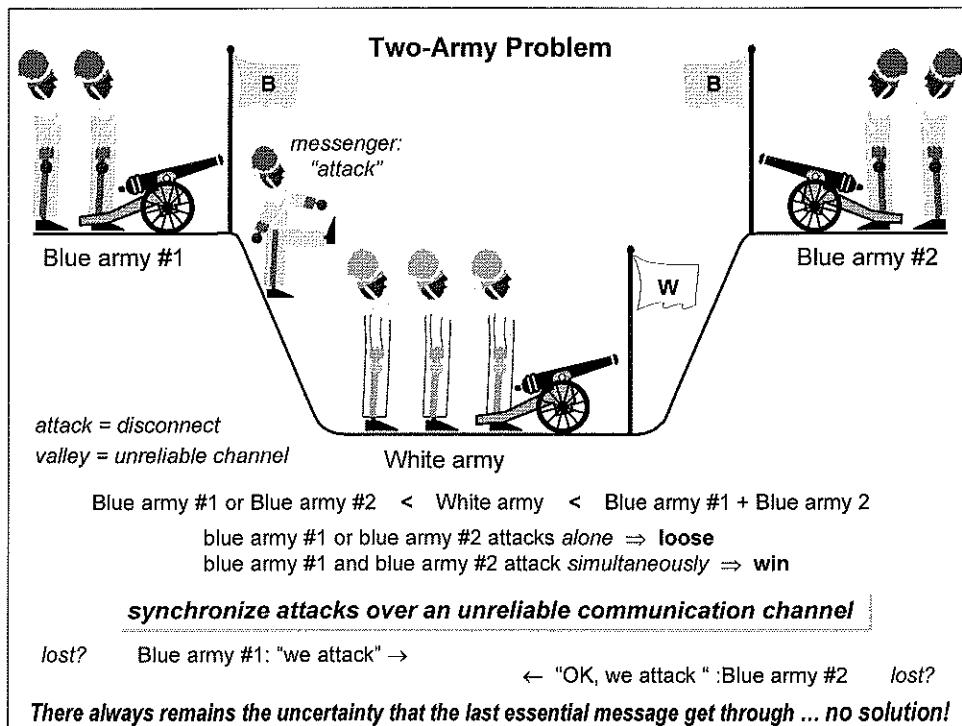
The first segment is a delayed duplicate ACTIVE_OPEN request from an old connection. This segment arrives at host S without host C's knowledge. TCP_S reacts to this segment by sending host C an acknowledgment, in effect asking for verification that host C was indeed trying to set up a new connection. When host C rejects (RST segment) host S's attempt to establish, host S realizes that it was tricked by a delayed duplicate and abandons the connection. In this way, a delayed duplicate does no damage.



Two programs that use TCP to communicate, can terminate the conversation gracefully using the *close* operation. Internally, TCP uses a *modified three-way handshake* to close connections. TCP connections are full duplex and they can be viewed as containing two independent stream transfers, one going in each direction. When an application program tells TCP that it has no more data to send, TCP will close the connection *in one direction*. To close its half of a connection, the sending TCP finishes transmitting the remaining data, waits for the receiver to acknowledge it, and then sends a segment with the FIN bit set. The receiving TCP acknowledges the FIN segment and informs the application program on its end that no more data is available (CLOSING service primitive).

Once a connection has been closed in a given direction, *TCP refuses to accept more data for that direction*. Meanwhile, data can continue to flow in the opposite direction until the sender closes it. Of course, acknowledgements continue to flow back to the sender even after a connection has been closed. When both directions have been closed, the TCP entity at each endpoint deletes its record of the connection.

The difference between three-way handshakes used to establish and terminate connections occurs after a machine receives the initial FIN segment. *Instead of generating a second FIN segment immediately, TCP sends an acknowledgement and then informs the application of the request to shut down. Informing the application program of the request and obtaining a response may take considerable time (e.g., it may involve human interaction). The acknowledgement prevents retransmission of the initial FIN segment during the wait. Finally, when the application program instructs TCP to shut down the connection completely, TCP sends the second FIN segment and the original site replies with the third message, an ACK. It is possible for the first ACK and the second FIN to be taken together in the same segment, reducing the total count to three.*

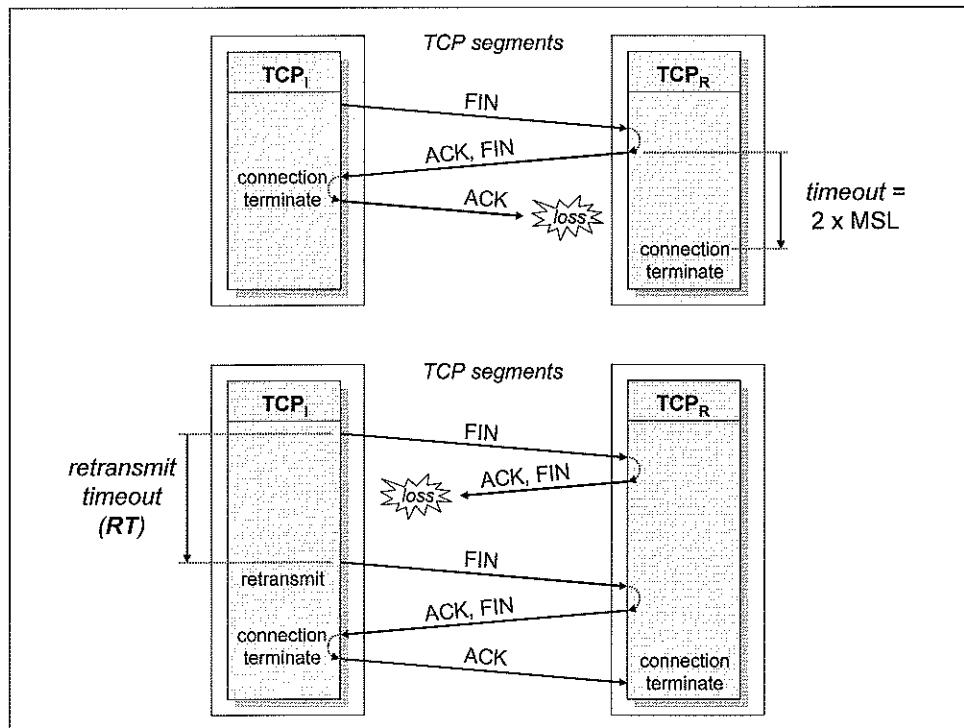


The **two-army problem**, describes the problem of terminating a connection over a network with packet loss. Imagine that a white army is encamped in a valley. On both of the surrounding hillsides are blue armies. The white army is larger than either of the blue armies alone, but together they are larger than the white army. If either blue army attacks by itself, it will be defeated, but if the *two blue armies attack 'simultaneously'*, they will be victorious.

The blue armies want to '*synchronize*' their attacks. However, their only communication medium is to send messengers on foot down into the valley, where they might be captured and the message lost (i.e., they have to use an unreliable communication channel). The question is: Does a protocol exist that allows the blue armies to win?

Suppose that the commander of blue army #1 sends a message reading: "I propose we attack at dawn on March 29. How about it?" Now suppose that the message arrives, and the commander of blue army #2 agrees, and that his reply gets safely back to blue army #1. Will the attack happen? Probably not, because commander #2 does not know if his reply got through. If it did not, blue army #1 will not attack, so it would be foolish for him to charge into battle.

Now let us improve the protocol by making it a *three-way handshake*. The initiator of the original proposal must acknowledge the response. Assuming no messages are lost, blue army #2 will get the acknowledgement, but the commander of blue army #1 will now hesitate. After all, he does not know if his acknowledgement got through, and if it did not, he knows that blue army #2 will not attack. We could now make a four-way handshake protocol, but that does not help either.



In fact, it can be proven that no protocol exists that works for the two-army problem. Suppose that some protocol did exist. Either the last message of the protocol is essential or it is not. If it is not, remove it (and any other unessential messages) until we are left with a protocol in which every message is essential. What happens if the final message does not get through? We just said that it was essential, so if it is lost, the attack does not take place. Since the sender of the final message can never be sure of its arrival, he will not risk attacking. Worse yet, the other blue army knows this, so it will not attack either.

To see the relevance of the two-army problem to releasing connections, just substitute "disconnect" for "attack." If neither side is prepared to disconnect until it is convinced that the other side is prepared to disconnect too, the disconnection will never happen.

In practice, one is usually prepared to take more risks when releasing connections than when attacking white armies, so the situation is not entirely hopeless.

To avoid the two-army problem, *timers* are used. If a response to a *FIN* is not forthcoming within two *maximum segment lifetimes* (2×120 sec), the sender of the *FIN* releases the connection. The other side will eventually notice that nobody seems to be listening to it any more, and time out as well. While this solution is not perfect, given the fact that a perfect solution is theoretically impossible, it will have to do. In practice, problems rarely arise.

3. UDP = User Datagram Protocol

- ***connectionless datagram delivery***

UDP has 'minimal overhead' to transport client/server application messages

- establish/release a connection is not needed, data transfer can start directly
- no sequence numbers (no ordering of incoming datagrams)
- no acknowledgments, no retransmissions
- no flow control (datagrams can arrive faster than the receiver can process)
- no congestion control (no feedback on the transmission rate of the source)

- ***unstructured byte-stream / buffered transfer***

UDP uses buffered ports, preventing data loss if the process is not ready.

- ***multiplexed***

UDP extends the host-to-host delivery service of the underlying IP network to an application-to-application communication service. Many applications can share access to a single UDP layer by the included multiplexing/demultiplexing.

- ***checksum***

Error detection on the data.

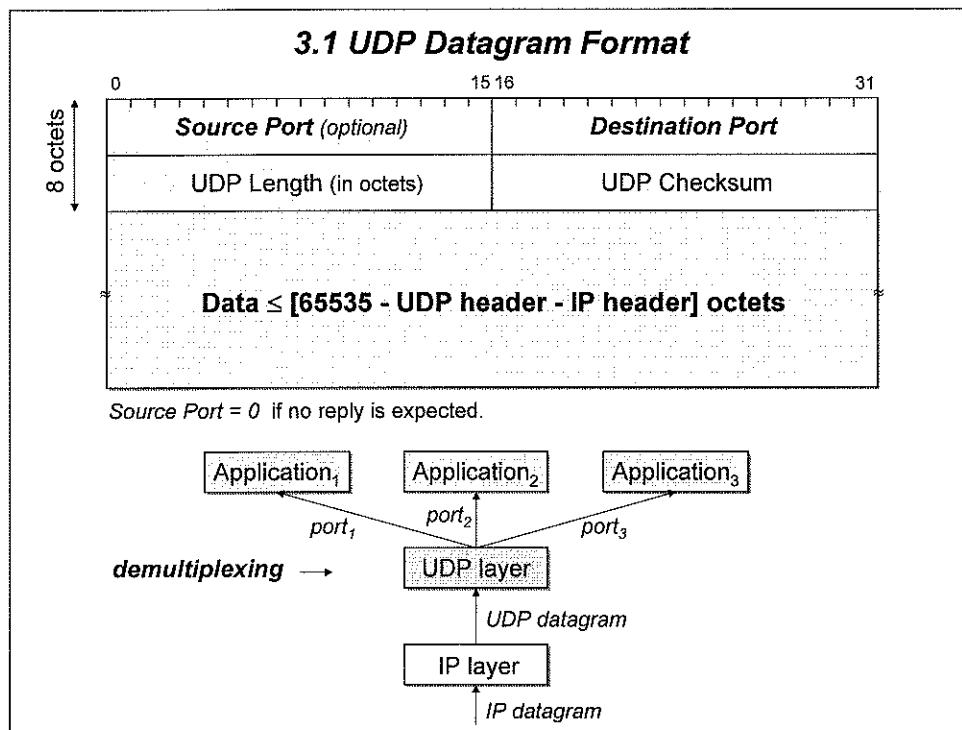
- ***unreliable***

UDP uses encapsulated raw IP datagrams to transport application messages. The application accepts full responsibility for handling the problem of reliability (message loss, duplication, delay, out-of order delivery, loss of connectivity).

UDP uses the underlying Internet Protocol to transport a message from one host to another, and provides the same *unreliable, connectionless datagram delivery* semantics as IP. It does not use acknowledgements to make sure messages arrive, it does not order incoming messages, and it does not provide feedback to control the rate at which information flows between the hosts. Thus, UDP messages can be lost, duplicated, or arrive out of order. Furthermore, packets can arrive faster than the receiver can process them.

UDP is almost a null protocol. It offers only *multiplexing/demultiplexing* (port number based, many applications can share access to a single UDP layer by the included multiplexing/demultiplexing) and *checksumming*, nothing else. Higher level protocols using UDP must provide their own retransmission, packetization, reassembly, flow control, congestion avoidance, and so on, if any are required. By itself, UDP is no more reliable than the underlying IP. UDP should *pass up* to the next layer any IP option that it receives, and should accept option specifications and other important *IP parameters* from higher level protocols that it should *pass downward* to IP. Similarly, all *ICMP error messages* received should be *passed upward* to the application lying above UDP.

In general, *ports are buffered*, so data that arrives before a process (protocol port) is ready to accept it, will not be lost. To achieve buffering, the protocol software located inside the operating system places packets that arrive for a particular protocol port in a (finite) queue until a process extracts them.



Each UDP message is called a *user datagram*. Conceptually, the user datagram consists of two parts, a UDP header and UDP data area. The header is divided into four 16-bit fields (8 octets) that specify the port from which the message was sent, the port to which the message is destined, the message length, and a UDP checksum.

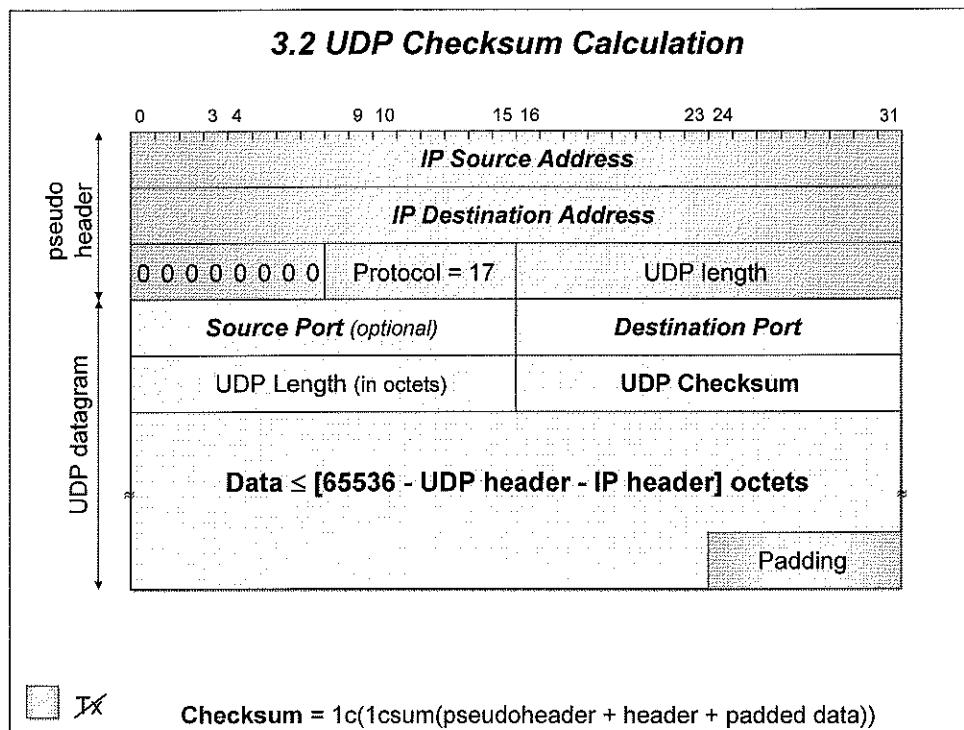
- The *Source Port* and *Destination Port* fields contain the 16-bit UDP protocol port numbers used to demultiplex datagrams among the processes waiting to receive them. The *Source Port* is optional. (UDP is not connection oriented.) When used, it specifies the port to which replies should be sent; if not used, it should be zero.

UDP provides protocol ports used to distinguish among multiple programs executing on a single machine. That is, in addition to the data sent, each UDP message contains both a *destination port number* and a *source port number*, making it possible for the UDP software on the destination to deliver the message to the correct receiving process and for the receiving process to send a reply.

The *IP layer* is responsible only for transferring data between a pair of hosts on an internet, while the *UDP layer* is responsible only for differentiating among multiple sources or destinations within one host.

Typically, a receiving process must obtain or be assigned a port number, and must be *listening* on that port for an arriving User Datagram to be delivered on that port. Arrival of a UDP datagram destined for a port on which no 'listen' is pending, should give rise to an *ICMP port unreachable* message and the UDP datagram is discarded.

- The *Length* field contains a count of octets in the UDP datagram, including the UDP header and the user data. Thus, the minimum value for *Length* is eight, the length of the header alone.



The UDP *Checksum* is *optional* and need not be used at all; a value of zero in the Checksum field means that the checksum has not been computed. This allows implementations to operate with little computational overhead when using UDP across a highly reliable local area network. IP does not compute a checksum on the data portion of an IP datagram. Thus, the UDP checksum provides the only way to guarantee that *data* has arrived intact.

- The *Checksum* field in the UDP header contains a 16-bit integer checksum used to verify the integrity of the data as well as the UDP header. To compute the checksum, UDP prepends a *pseudo header* to the segment, appends enough bytes containing zero (at the data) to pad the segment to a multiple of 16 bits, and computes the 16-bit checksum over the entire result (data included). UDP assumes the checksum field itself is zero for purposes of the checksum computation. As with other checksums, UDP uses 16-bit arithmetic and takes the one's complement of the one's complement sum. At the receiving site, UDP software performs the same computation to verify that the segment arrived intact.

The *pseudo header* allows the receiver to verify that the segment has reached its correct destination, which includes both a host IP address as well as a protocol port number (the socket).

- The sending UDP assigns field *Protocol* the value that the underlying delivery system will use in its protocol type field. For IP datagrams carrying UDP, the value 17.
- The *UDP Length* field specifies the total length of the UDP segment including the UDP header (not the pseudo header or the octet for data padding).
- The pseudo header and the padding octet are not transmitted with the UDP segment, nor are they included (counted) in the UDP Length.
- Using a *pseudo header* allows the receiver to verify that the segment has reached its correct destination, which includes both a *host IP address as well as a protocol port number (the socket)*.

HOOGESCHOOL VOOR WETENSCHAP & KUNST

DE NAYER INSTITUUT

SINT-KATELIJNE-WAVER

**Datacommunication
en
Digitale
Formulierarchitectuur**

Formulierarchium

EmSD
Embedded System Design



ir. J. Meel

nov 2007

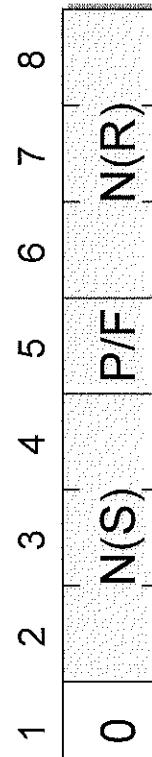
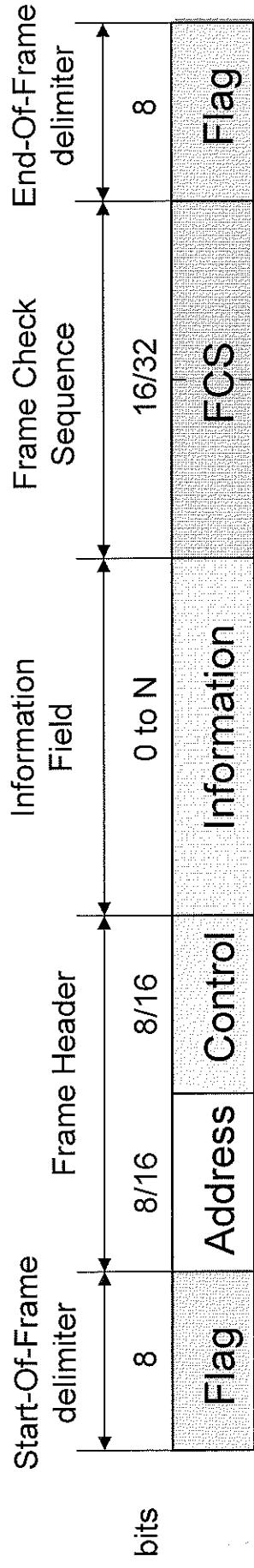
Basic Mode Format

	SYN	SYN	SOH	m	STX	text	ETX	BCC
--	------------	------------	------------	----------	------------	-------------	------------	------------

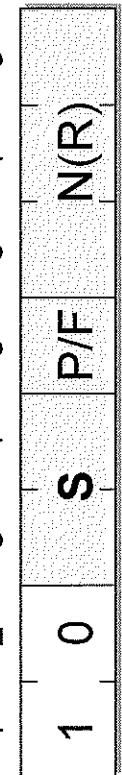
TC	Name	Ascii Code (7bit)	MSB first even parity	Hex Code (8 bit)	LSB first even parity	Hex Code (8 bit)
TC1	SOH	(01)_H	000 0001 1	(03)_H	1000 0001	(81)_H
TC2	STX	(02)_H	000 0010 1	(05)_H	0100 0001	(41)_H
TC3	ETX	(03)_H	000 0011 0	(06)_H	1100 0000	0(C0)_H
TC4	EOT	(04)_H	000 0100 1	(09)_H	0010 0001	(21)_H
TC5	ENQ	(05)_H	000 0101 0	(0A)_H	1010 0000	0(A0)_H
TC6	ACK	(06)_H	000 0110 0	(0C)_H	0110 0000	0(60)_H
TC7	DLE	(10)_H	001 0000 1	(21)_H	0000 1001	1(09)_H
TC8	NAK	(15)_H	001 0101 1	(2B)_H	1010 1001	(A9)_H
TC9	SYN	(16)_H	001 0110 1	(2D)_H	0110 1001	(69)_H
TC10	ETB	(17)_H	001 0111 0	(2E)_H	1110 1000	0(E8)_H

HDLC Frame Format

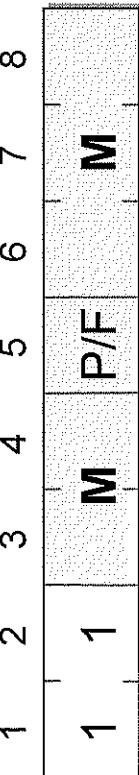
→ Direction of transmission



I-frame



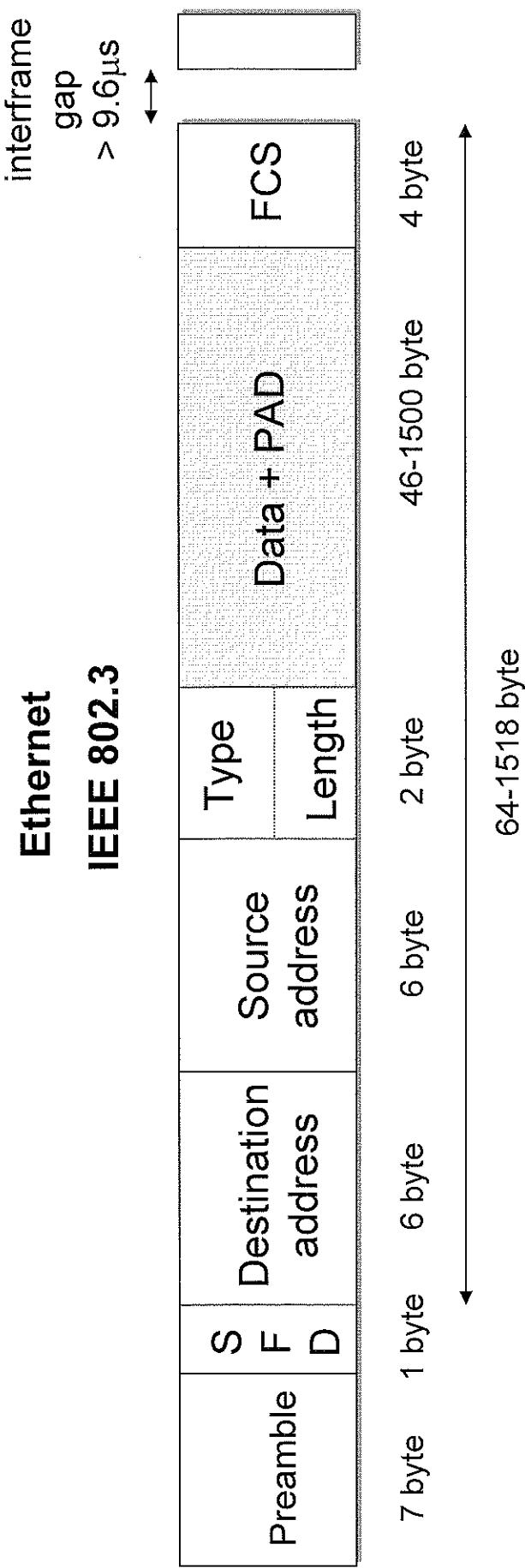
S-frame



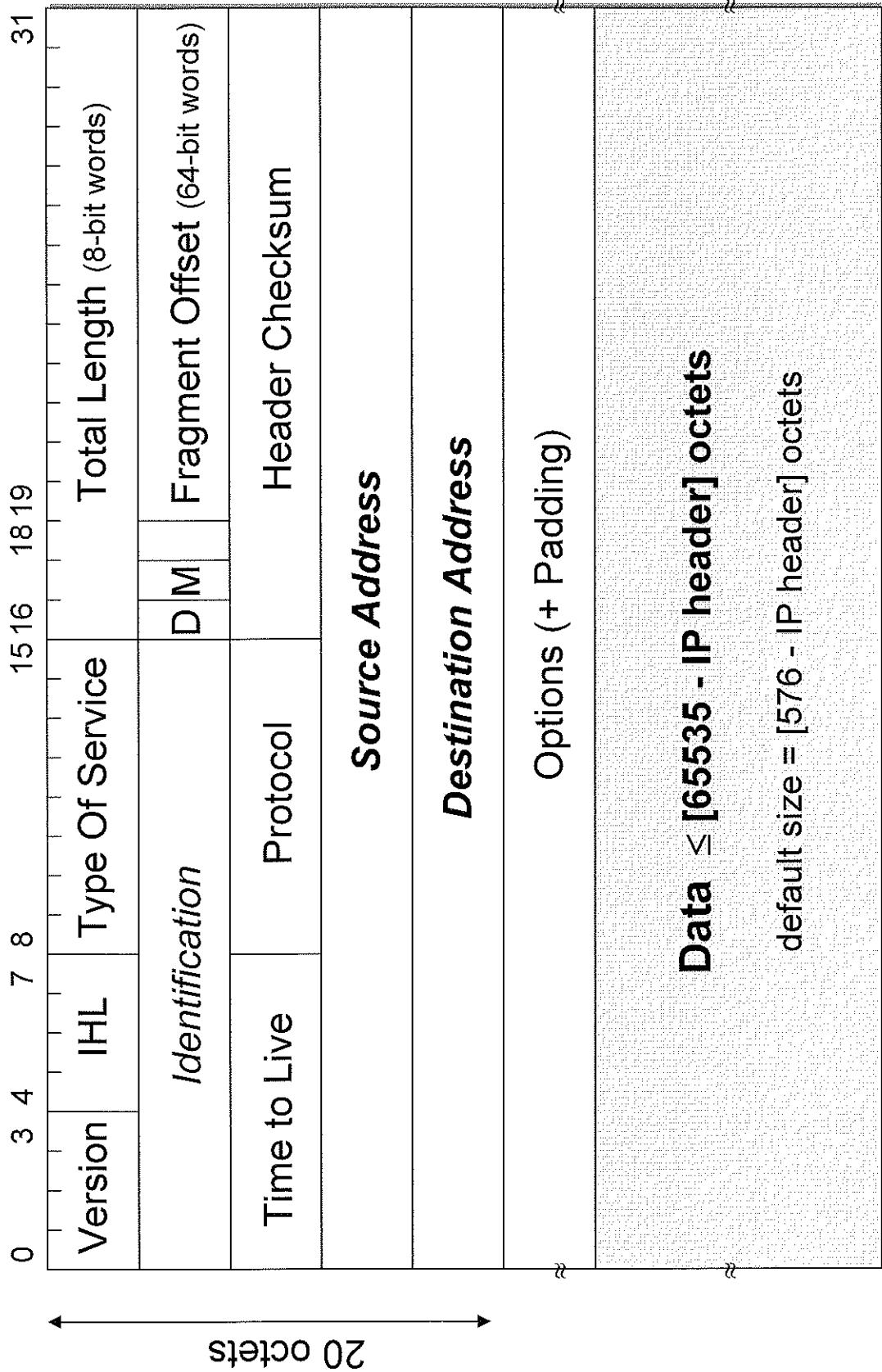
standard (8-bit) (N = 3 bit)

Type		Name	C/R	Description	
I		Information	C/R	exchange CO ‘user info’ (L3)	
S	RR RNR REJ SREJ	Receive Ready Receive Not Ready Reject Selective Reject	C/R C/R C/R C/R	positive ack (start Tx) positive ack (stop Tx) negative ack, go-back-N negative ack, selective reject	
				Link Management (Modes)	
		SNRM(E) SARM(E) SABM(E) DISC	Set Normal Mode (Extended) Set Asyn Response Mode (E) Set Asyn Balanced Mode (E) Disconnect	C	set NRM (N=3-bit; E:N=7-bit) set ARM (N=3-bit; E:N=7-bit) set ABM (N=3-bit; E:N=7-bit) terminate logic link connection
U	UA DM	Unnumbered Acknowledge Disconnect Mode	R	mode command accepted (CL) slave stays disconnected	
				Recovery	
	RSET FRMR	Reset Frame Reject	C R	reset V(R) and V(S) reports receipt of wrong frame	
UI		Unnumbered Information	C/R	Data Transfer exchange CL ‘control info’ (L3)	

Ethernet LAN Frame Format



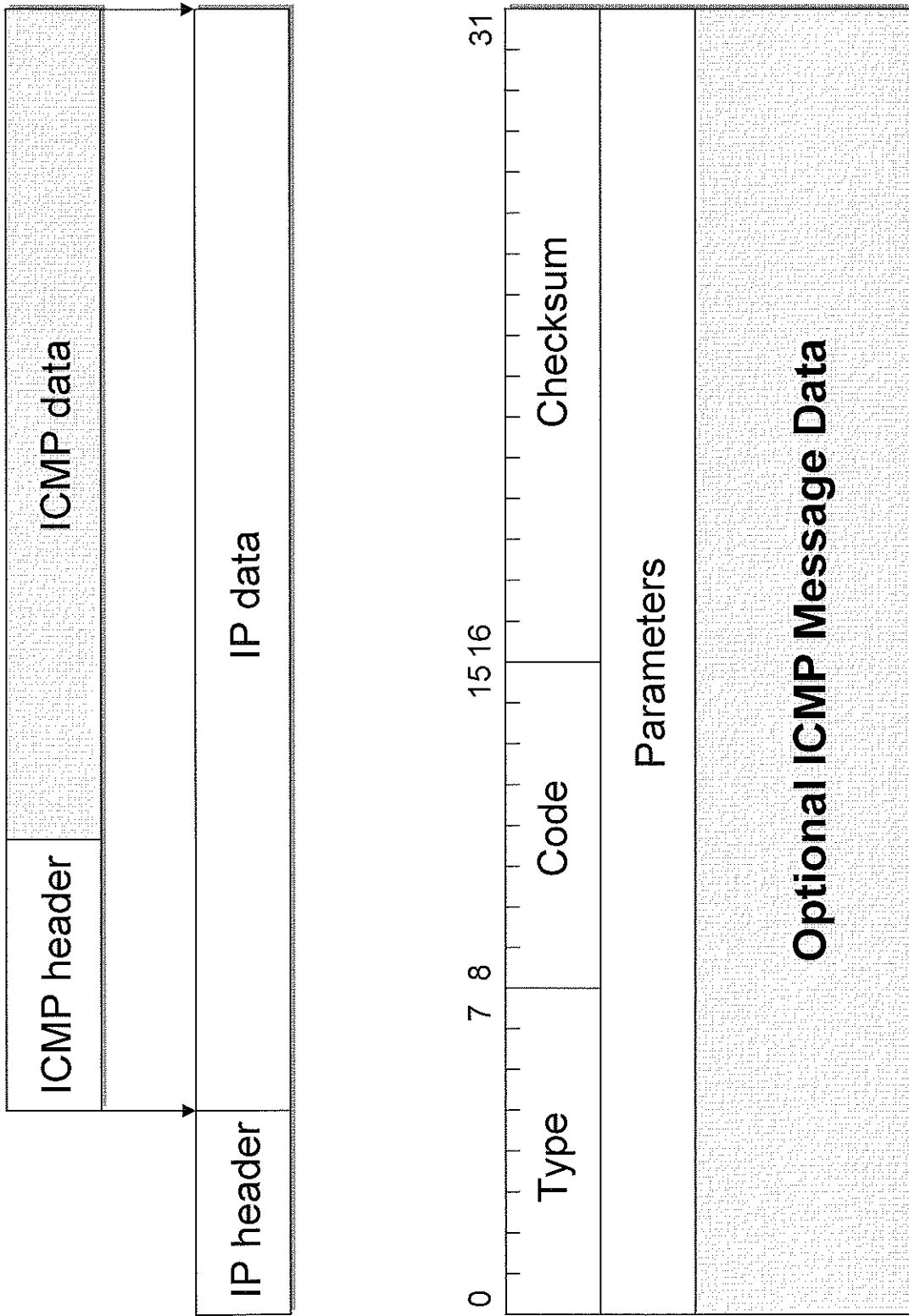
IPv4 Datagram Format



IHL = IP Header Length (in 32-bit words)

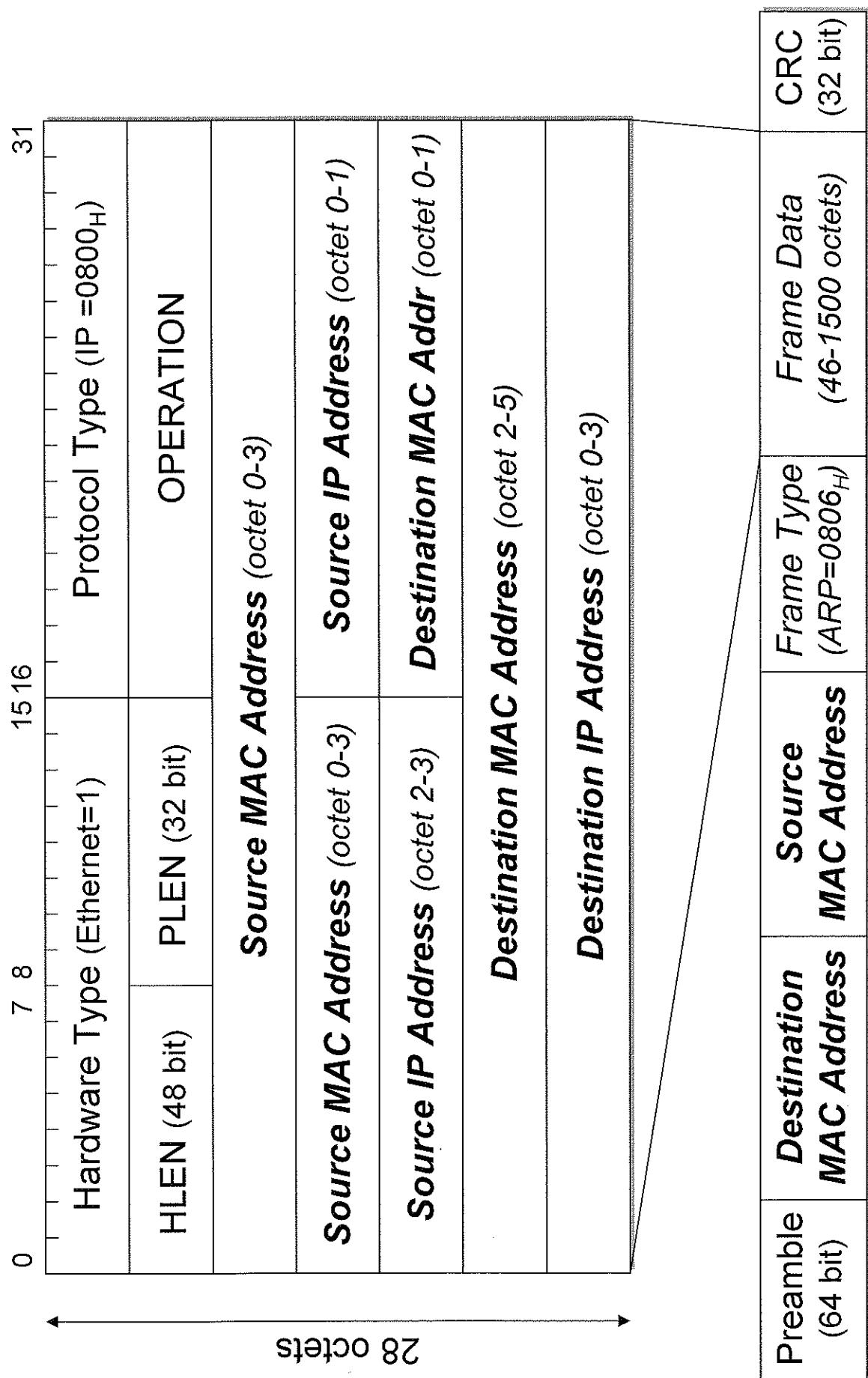
Flags:(D = Don't Fragment), (M = More)

ICMP Message Format

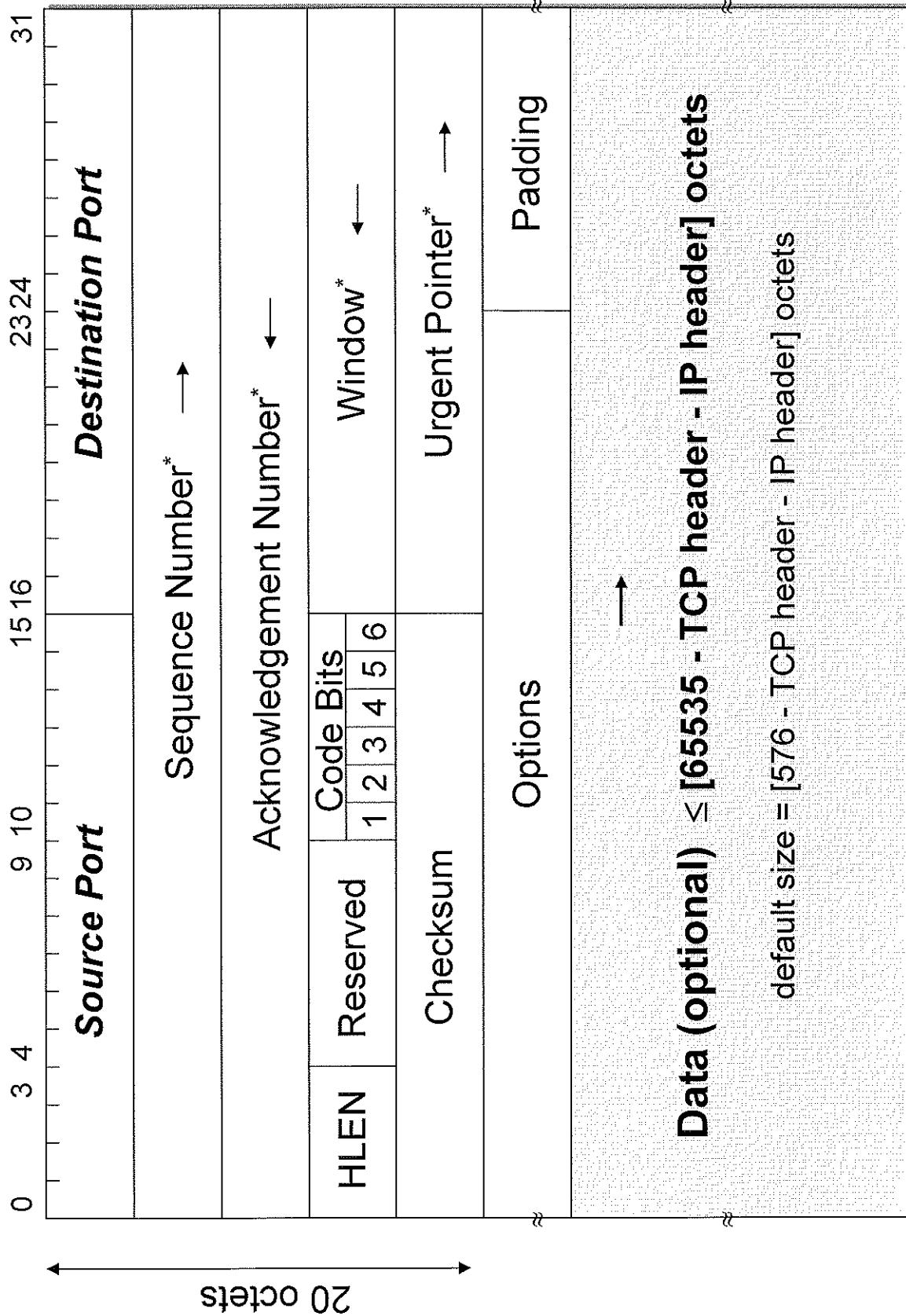


Checksum: over entire ICMP message, same additive algorithm as IP

ARP Message Format



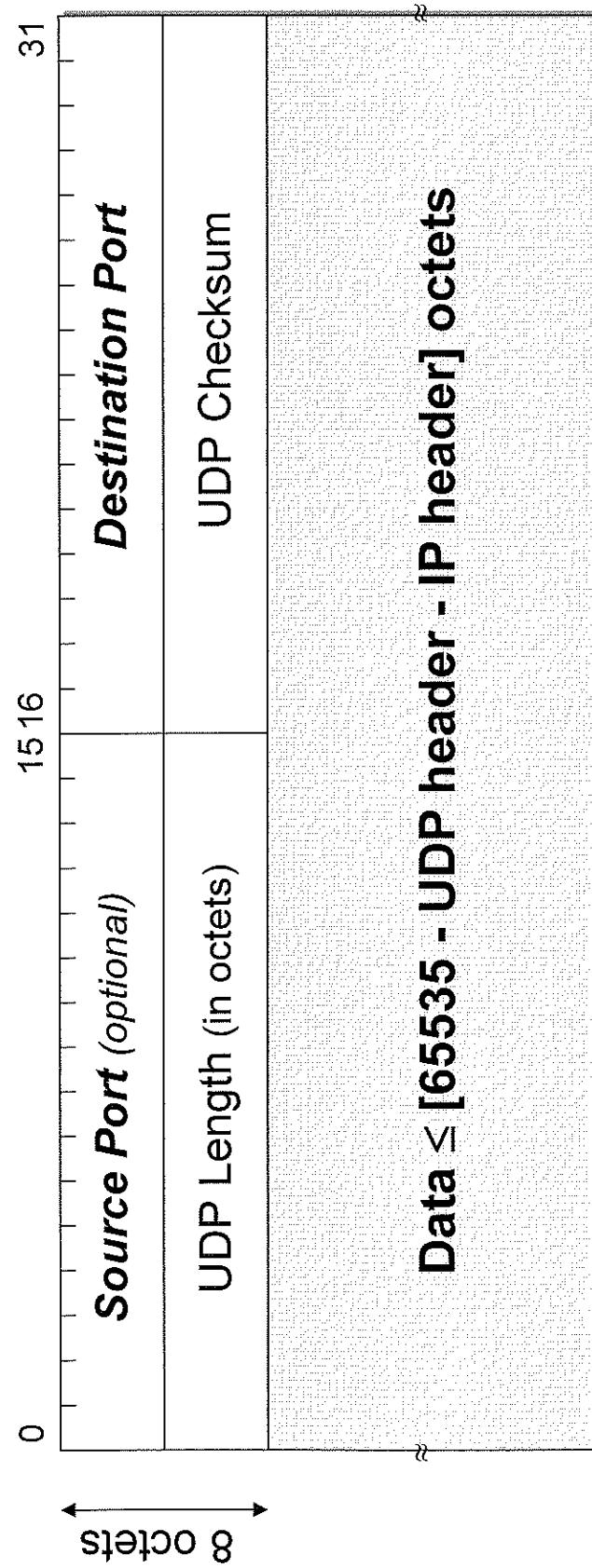
TCP Segment Format



$HLEN = \text{TCP Header Length (in 32-bit words)}$
1=URG 2=ACK 3=PSH 4=RST 5=SYN 6=FIN

$* = \text{octet based}$

UDP Datagram Format



Source Port = 0 if no reply is expected.