



CONET MODULE 1 **ETHERNET BASED** **I/O SYSTEMS**

LECTURE NOTES

Onur Akbatı
Levent Uçun
Galip Cansever



Contents

1. ETHERNET	4
1.1 FOUR BASICS OF ETHERNET	5
1.1.1. NETWORK CORE: CIRCUIT SWITCHING & PACKET SWITCHING CIRCUIT SWITCHING.....	13
1.1.2. OVERVIEW OF DELAY IN PACKET SWITCHED NETWORKS.....	17
2. LAYERS	18
2.1. OSI REFERENCE MODEL.....	18
2.1.1. SEVEN LAYER ISO-OSI REFERENCE MODEL	20
2.1.1.1. APPLICATION LAYER	20
2.1.1.2. PRESENTATION LAYER	20
2.1.1.3. SESSION LAYER.....	21
2.1.1.4. TRANSPORT LAYER.....	21
2.1.1.5. NETWORK LAYER.....	21
2.1.1.6. DATA LINK LAYER	21
2.1.1.7. PHYSICAL LAYER	22
2.2. FIVE LAYER INTERNET PROTOCOL	22
2.2.1. APPLICATION LAYER	22
2.2.2. TRANSPORT LAYER.....	22
2.2.3. NETWORK LAYER.....	23
2.2.4. DATA LINK LAYER	23
2.2.5. PHYSICAL LAYER	23
2.3. APPLICATION LAYER	24
2.3.1. HTTP (HYPER TEXT TRANSPORT PROTOCOL)	24
2.3.1.1. NON-PERSISTENT AND PERSISTENT CONNECTIONS	25
2.3.2. FTP (FILE TRANSFER PROTOCOL)	28
2.3.3. SMTP (SIMPLE MAIL TRANSFER PROTOCOL).....	30
2.3.4. DNS (DOMAIN NAME SERVICE).....	30
2.4. TRANSPORT LAYER	32
2.4.1. MULTIPLEXING AND DEMULTIPLEXING	34
2.4.2. CONNECTIONLESS TRANSPORT: UDP (USER DATAGRAM PROTOCOL)	34
2.4.3. CONNECTION-ORIENTED TRANSPORT (TRANSMISSION CONTROL PROTOCOL)	37
2.4.3.1. TCP SEGMENT STRUCTURE	38
2.4.3.2. SEQUENCE NUMBERS AND ACKNOWLEDGE NUMBERS.....	41
2.4.3.3. PRINCIPLES OF RELIABLE DATA TRANSFER.....	42
2.4.3.4. ARQ ALGORITHMS.....	42
2.5. NETWORK LAYER.....	48
2.5.1. INTERNET PROTOCOL (IPV4).....	48
2.5.1.1. IP ADDRESSING	48
2.5.1.2. ASSIGNING ADDRESSES.....	51
2.5.1.3. TRANSPORTING A DATAGRAM FROM SOURCE TO DESTINATION	52
2.5.1.4. DATAGRAM FORMAT	54
2.5.1.5. IP FRAGMENTATION AND REASSEMBLY.....	57
2.5.1.6. INTERNET CONTROL MESSAGE PROTOCOL (ICMP).....	59
2.6. LINK LAYER AND LOCAL AREA NETWORKS (LAN).....	60
2.6.1. ERROR-DETECTION AND CORRECTION TECHNIQUES.....	64
2.6.1.1. PARITY CHECKS.....	65
2.6.1.2. CHECKSUM METHODS.....	66
2.6.1.3. CYCLIC REDUNDANCY CHECK (CRC).....	66
2.6.2. LINK LAYER ADDRESSING.....	66
2.6.2.1. ADDRESS RESOLUTION PROTOCOL (ARP)	68
2.6.2.2. SENDING A DATAGRAM TO A NODE OFF THE LAN	70
3. BRIDGES AND SWITCHES	72
3.1. HUBS	72
3.2. BRIDGES.....	74
3.2.1. FORWARDING AND FILTERING	74
3.3. BRIDGES VERSUS ROUTERS.....	76
3.4. SWITCHES.....	78
4. INDUSTRIAL ETHERNET TECHNOLOGIES.....	80
4.1. NON-REAL TIME APPLICATIONS	81
4.1.1. MODBUS	81
4.1.1.1. PROTOCOL VERSIONS	82
4.1.1.2. COMMUNICATION AND DEVICES.....	82
4.1.1.3. FRAME FORMAT	83
4.1.1.4. SUPPORTED FUNCTION CODES	84
4.1.1.5. IMPLEMENTATIONS	85
4.1.1.6. LIMITATIONS.....	85
4.1.2. ETHERNET IP.....	86
4.1.2.1. COMMON INDUSTRIAL PROTOCOL (CIP).....	87
4.1.2.2. CONFIGURATION WITH EDS-FILES	88
4.1.2.3. TECHNICAL DETAILS.....	88
4.2. REAL TIME APPLICATIONS	89
4.2.1. ETHERNET POWERLINK.....	89



4.2.1.1.	OVERVIEW	89
4.2.1.2.	STANDARDIZATION	89
4.2.1.3.	PHYSICAL LAYER	89
4.2.1.4.	DATA LINK LAYER	90
4.2.1.5.	BASIC CYCLE	90
4.2.1.6.	MULTIPLEX FOR BANDWIDTH OPTIMIZATION	92
4.2.1.7.	ETHERNET POWERLINK SAFETY	92
4.2.2.	PROFINET	92
4.2.2.1.	TECHNOLOGY	93
4.2.2.2.	PROFINET COMPONENT MODEL (PROFINET CBA)	94
4.2.2.3.	PROFINET AND THE PERIPHERALS (PROFINET IO)	94
4.2.2.4.	CONFIGURATION WITH GSD-FILES	95
4.2.2.5.	PROFINET AND REAL TIME	96
4.2.2.6.	PROFINET AND ISOCHRONOUS COMMUNICATION	96
4.2.2.7.	PROFILES	96
4.2.2.8.	ADDITIONAL HIGHLIGHTS OF THE PROFINET CONCEPT	97
4.2.2.9.	BENEFITS OF PROFINET	97
4.3.	ISOCHRONOUS REAL TIME APPLICATIONS	97
4.3.1.	SERCOS	97
4.3.1.1.	INTRODUCTION	97
4.3.1.2.	HISTORY	98
4.3.1.3.	VERSIONS	98
4.3.1.4.	SERCOS INTERFACE FEATURES	98
4.3.2.	ETHERCAT	99
4.3.2.1.	INTRODUCTION	99
4.3.2.2.	FUNCTIONAL PRINCIPLE	99
4.3.2.3.	PROTOCOL	99
4.3.2.4.	PERFORMANCE	100
4.3.2.5.	TOPOLOGY	100
4.3.2.6.	ETHERCAT INSTEAD OF PCI	100
4.3.2.7.	SYNCHRONIZATION	101
4.3.2.8.	DISTRIBUTED CLOCK	101
4.3.2.9.	HOT CONNECT	102
4.3.2.10.	SAFETY OVER ETHERCAT	102
4.3.2.11.	OPENNESS	102
4.3.2.12.	DEVICE PROFILES	103
4.3.2.13.	GATEWAYS	103
4.3.2.14.	IMPLEMENTATION	103
5.	CONTENTS OF THE CML BOX	104
5.1.	PLC AND DISTRIBUTED I/OS	104
5.2.	SABO SIMULATOR	104
5.3.	MANAGEABLE SWITCHES	105
5.4.	NAT ROUTER	106
5.5.	RAIL PC	107
5.6.	WIRESHARK	108
6.	CABLES AND TOPOLOGY	109
6.1.	UNSHIELDED TWISTED PAIR (UTP) CABLE	109
6.1.1.	UNSHIELDED TWISTED PAIR CONNECTOR	109
6.2.	SHIELDED TWISTED PAIR (STP) CABLE	110
6.3.	COAXIAL CABLE	110
6.3.1.	COAXIAL CABLE CONNECTORS	111
6.4.	FIBER OPTIC CABLE	111
6.5.	WHAT IS A TOPOLOGY?	113
6.5.1.	MAIN TYPES OF PHYSICAL TOPOLOGIES	113
6.5.1.1.	LINEAR BUS	113
6.5.1.2.	STAR	114
6.5.1.3.	TREE OR EXPANDED STAR	114
6.5.1.4.	CONSIDERATIONS WHEN CHOOSING A TOPOLOGY	116
6.5.1.5.	SUMMARY CHART	116
6.6.	ETHERNET Crossover CABLE	116
6.6.1.	AUTOMATIC Crossover	121



1. Ethernet

There are number of factors that have helped ethernet to become so popular. These factors include cost, scalability, reliability, and widely available tools for management.

Cost: The widespread adoption of ethernet technology has created a large and fiercely competitive ethernet marketplace. This has driven down the cost of networking components, and consumers have won out in the process, with the marketplace providing a wide range of competitively priced Ethernet components to choose from.

Scalability: Applications tend to grow to fill all available bandwidth. To anticipate the rising demand, Gigabit Ethernet was developed in 1998, providing yet another tenfold increase in performance.

Desktop machines can be connected to the original 10 Mbps Ethernet, 100 Mbps Fast Ethernet, or Gigabit Ethernet as required.

Reliability: Ethernet uses a simple and robust transmission mechanism that reliably delivers data day in and day out at sites all over the world. Ethernet based on twisted-pair media was introduced in 1987, making it possible to provide Ethernet signals over a structured cabling system. Structured cabling provides a data delivery system for a building that is modeled on high-reliability cabling practices originally developed for the telephone system. This makes it possible to install a standards-based cabling system for Ethernet that is very reliable, as well as being simple, stable, and easy to manage.

Widely Available Management Tools: The widespread acceptance of Ethernet brings another advantage; namely the availability of Ethernet management and troubleshooting tools. Management tools based on standards, such as the Simple Network Management Protocol (SNMP), make it possible for network administrators to keep track of an entire campus full of Ethernet equipment from a central location. Management capabilities embedded in Ethernet repeaters, switching hubs, and computer interfaces provide powerful network monitoring and troubleshooting capabilities.

Half-duplex and Full Duplex

Half-duplex simply means that only one computer can send data over the Ethernet channel at any given time. In half-duplex mode, multiple computers share a single Ethernet channel by using the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) media access control (MAC) protocol. Until the introduction of switching hubs, the half-duplex system was the typical mode of operation for the vast majority of Ethernet LANs—tens of millions of Ethernet connections have been installed based on this system.

However, these days many computers are connected directly to their own port on an Ethernet switching hub and do not share the Ethernet channel with other systems.

Many computers and switching hub connections now use full-duplex mode, in which the CSMA/CD protocol is shut off and the two devices on the link can send data whenever they like.

1.1 Four Basics of Ethernet

An Ethernet Local Area Network (LAN) is made up of hardware and software working together to deliver digital data between computers. To accomplish this task and create a working Ethernet system, four basic elements are combined.

- The frame, which is a standardized set of bits used to carry data over the system.
- The media access control protocol, which consists of a set of rules embedded in each Ethernet interface that allow multiple computers to access the shared Ethernet channel in a fair manner.
- The signaling components, which consist of standardized electronic devices that send and receive signals over an Ethernet channel.
- The physical media, which consists of the cables and other hardware used to carry the digital Ethernet signals between computers attached to the network.

• The Ethernet Frame

The heart of the Ethernet system is the frame. The network hardware—which is comprised of the Ethernet interfaces, media cables, etc.—exists simply to move Ethernet frames between computers, or stations. The bits in the Ethernet frame are formed up in specified fields. Figure 1 shows the basic frame fields.

Ethernet



Figure 1. Basic Frame Fields

The basic Ethernet frame begins with a set of 64 bits called the preamble. The preamble gives all of the hardware and electronics in a 10 Mbps Ethernet system some signal start-up time to recognize that a frame is being transmitted, alerting it to start receiving the data. This is what a 10 Mbps network uses to clear its throat, so to speak. Newer Ethernet systems running at 100 and 1000 Mbps use constant signaling, which avoids the need for a preamble. However, the preamble is still transmitted in these systems to avoid making any changes in the structure of the frame.

Following the preamble are the destination and source addresses. Assignment of addresses is controlled by the IEEE Standards Association (IEEE-SA), which administers a portion of the address field. When assigning blocks of addresses for use by network vendors, the IEEE-SA provides a 24-bit Organizationally Unique Identifier (OUI). The OUI is a unique 24-bit identifier assigned to each organization that builds network interfaces. This allows a vendor of Ethernet equipment to provide a unique



address for each interface they build. Providing unique addresses during manufacturing avoids the problem of two or more Ethernet interfaces in a network having the same address. This also eliminates any need to locally administer and manage Ethernet addresses.

A manufacturer of Ethernet interfaces creates a unique 48-bit Ethernet address for each interface by using their assigned OUI for the first 24 bits of the address. The vendor then assigns the next 24 bits, being careful to ensure that each address is unique. The resulting 48-bit address is often called the hardware, or physical, address to make the point that the address has been assigned to the Ethernet interface. It is also called a Media Access Control (MAC) address, since the Ethernet media Access control system includes the frame and its addressing.

Following the addresses in an Ethernet frame is a 16-bit type or length field. Most often this field is used to identify what type of high-level network protocol is being carried in the data field, e.g., TCP/IP. This field may also be used to carry length information.

After the type field can come anywhere from 46 bytes to 1500 bytes of data. The data field must be at least 46 bytes long. This length assures that the frame signals stay on the network long enough for every Ethernet station on the network system to hear the frame within the correct time limits. Every station must hear the frame within the maximum round-trip signal propagation time of an Ethernet system, as described later in this chapter. If the high-level protocol data carried in the data field is shorter than 46 bytes, then padding data is used to fill out the data field.

Finally, at the end of the frame there's a 32-bit Frame Check Sequence (FCS) field. The FCS contains a Cyclic Redundancy Checksum (CRC) which provides a check on the integrity of the data in the entire frame. The CRC is a unique number that is generated by applying a polynomial to the pattern of bits that make up the frame. The same polynomial is used to generate another checksum at the receiving station. The receiving station checksum is then compared to the checksum generated at the sending station. This allows the receiving Ethernet interface to verify that the bits in the frame survived their trip over the network system intact.

• Media Access Control Protocol

The half-duplex mode of operation described in the original Ethernet standard uses the MAC protocol, which is a set of rules used to arbitrate access to the shared channel among a set of stations connected to that channel. The way the access protocol works is fairly simple: each Ethernet-equipped computer operates independently of all other stations on the network; there is no central controller. All stations attached to an Ethernet operating in half-duplex mode are connected to a shared signaling channel, also known as a *signal bus*.

Ethernet uses a broadcast delivery mechanism, in which each frame that is transmitted is heard by every station. While this may seem inefficient, the advantage is that putting the address-matching intelligence in the interface of each station allows the physical medium to be kept as simple as possible. On an Ethernet LAN, all that the physical



signaling and media system has to do is see that the bits are accurately transmitted to every station; the Ethernet interface in the station does the rest of the work.

Ethernet signals are transmitted from the interface and sent over the shared signal channel to every attached station. To send data, a station first listens to the channel, and if the channel is idle the station transmits its data in the form of an Ethernet frame or packet.

As each Ethernet frame is sent over the shared signal channel, or medium, all Ethernet interfaces connected to the channel read in the bits of the signal and look at the second field of the frame shown in Figure 1, which contains the destination address. The interfaces compare the destination address of the frame with their own 48-bit unicast address or a multicast address they have been enabled to recognize. The Ethernet interface whose address matches the destination address in the frame will continue to read the entire frame and deliver it to the networking software running on that computer. All other network interfaces will stop reading the frame when they discover that the destination address does not match their own unicast address or an enabled multicast address

After each frame transmission, all stations on the network with traffic to send must contend equally for the next frame transmission opportunity. This ensures that access to the network channel is fair and that no single station can lock out the others. Fair access of the shared channel is made possible through the use of the MAC system embedded in the Ethernet interface located in each station. The media access control mechanism for Ethernet uses the CSMA/CD protocol.

CSMA/CD Protocol (Carrier Sense Multiple Access and Collision Detection)

The Carrier Sense portion of the protocol means that before transmitting, each interface must wait until there is no signal on the channel. If there is no signal, it can begin transmitting. If another interface is transmitting, there will be a signal on the channel; this condition is called *carrier*. All other interfaces must wait until carrier ceases and the signal channel is idle before trying to transmit; this process is called *deferral*.

With Multiple Access, all Ethernet interfaces have the same priority when it comes to sending frames on the network, and all interfaces can attempt to access the channel at any time.

The next portion of the access protocol is called Collision Detect. Given that every Ethernet interface has equal opportunity to access the Ethernet, it's possible for multiple interfaces to sense that the network is idle and start transmitting their frames simultaneously. When this happens, the Ethernet signaling devices connected to the shared channel sense the collision of signals, which tells the Ethernet interfaces to stop transmitting. Each of the interfaces will then choose a random retransmission time and resend their frames in a process called back off.

The CSMA/CD protocol is designed to provide fair access to the shared channel so that all stations get a chance to use the network and no station gets locked out due to



some other station hogging the channel. After every packet transmission, all stations use the CSMA/CD protocol to determine which station gets to use the Ethernet channel next.

If more than one station happens to transmit on the Ethernet channel at the same moment, then the signals are said to *collide*. The stations are notified of this event and reschedule their transmission using a random time interval chosen by a specially designed back off algorithm. Choosing random times to retransmit helps the stations to avoid colliding again on the next transmission.

It's unfortunate that the original Ethernet design used the word 'collision' for this aspect of the Ethernet media access control mechanism. If it had been called something else, such as Distributed Bus Arbitration (DBA) events, then no one would worry about the occurrence of DBAs on an Ethernet. To most ears the word 'collision' sounds like something bad has happened, leading many people to incorrectly conclude that collisions are an indication of network failure and that lots of collisions must mean the network is broken.

Instead, the truth of the matter is that collisions are absolutely normal events on an Ethernet and are simply an indication that the CSMA/CD protocol is functioning as designed. As more computers are added to a given Ethernet, there will be more traffic, resulting in more collisions as part of the normal operation of an Ethernet. Collisions resolve quickly. For example, the design of the CSMA/CD protocol ensures that the majority of collisions on a 10 Mbps Ethernet will be resolved in microseconds, or millionths of a second. Nor does a normal collision result in lost data. In the event of a collision, the Ethernet interface backs off (waits) for some number of microseconds, and then automatically retransmit the frame.

Networks with very heavy traffic loads may experience multiple collisions for each frame transmission attempt. This is also expected behavior. Repeated collisions for a given packet transmission attempt indicate a very busy network. If repeated collisions occur, the stations involved will expand the set of potential back off times in order to retransmit the data. The expanding back off process, formally known as *truncated binary exponential back off*, is a clever feature of the Ethernet MAC protocol that provides an automatic method for stations to adjust to changing traffic conditions on the network. Only after 16 consecutive collisions for a given transmission attempt will the interface finally discard the Ethernet frame. This can happen only if the Ethernet channel is overloaded for a fairly long period of time or if it is broken.

So far, we've seen what an Ethernet frame looks like and how the CSMA/CD protocol is used to ensure fair access for multiple stations sending their frames over the shared Ethernet channel. The frame and the CSMA/CD protocol are the same for all varieties of Ethernet. Whether the Ethernet signals are carried over coaxial, twisted-pair, or fiber optic cable, the same frame is used to carry the data and the same CSMA/CD protocol is used to provide the half-duplex shared channel mode of operation. In full-duplex mode, the same frame format is used, but the CSMA/CD protocol is shut off.

• Signaling Components

The signaling components for a twisted-pair system include the Ethernet interface located in the computer, as well as a transceiver and its cable. An Ethernet may consist of a pair of stations linked with a single twisted-pair segment, or multiple stations connected to twisted-pair segments that are linked together with an Ethernet repeater. A repeater is a device used to repeat network signals onto multiple segments. Connecting cable segments with a repeater makes it possible for the segments to all work together as a single shared Ethernet channel.

The Ethernet interface connects to the media system using a transceiver, which can be built into the interface or provided as an external device. Of the two stations shown in Figure 2, one is provided with a built-in transceiver and one uses an external transceiver. (The word "transceiver" is a combination of transmitter and receiver.) A transceiver contains the electronics needed to take signals from the station interface and transmit them to the twisted-pair cable segment, and to receive signals from the cable segment and send them to the interface.

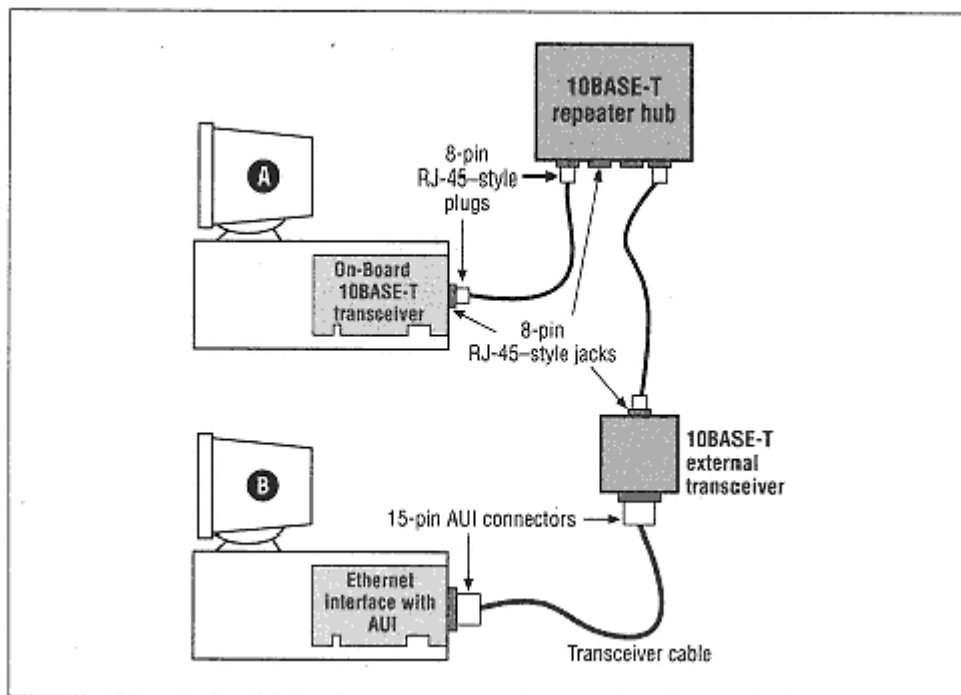


Figure 2. 10BASE-T Ethernet Connection

Figure 2 shows the Ethernet interface in Station A connected directly to the twisted-pair cable segment since it is equipped with an internal 10BASE-T transceiver. The twisted-pair cable uses an 8-pin connector that is also known as an RJ-45 plug. The Ethernet interface in Station B is connected to an outboard transceiver, which is a small box that contains the transceiver electronics. The outboard transceiver connects to the twisted-pair segment using an 8-pin connector. The outboard transceiver connects to the Ethernet interface in the station with a transceiver cable. The transceiver cable in the



10 Mbps Ethernet systems uses a 15-pin connector which is called the Attachment Unit Interface (AUI).

The final signaling component shown in Figure 2 is an Ethernet repeater hub which links multiple twisted-pair segments. This device is called a hub because it sits at the center, or hub, of a set of cable segments. The repeater connects to the cable segments using the same built-in 10BASE-T transceivers used by an ordinary station interface. The repeater operates by moving Ethernet signals from one segment to another one bit at a time; it does not operate at the level of Ethernet frames, but simply repeats the signals it sees on each segment. Repeaters make it possible to build larger Ethernet systems composed of multiple cable segments by making the segments function together as a single channel.

• Physical Media

The cables and other components used to build the signal-carrying portion of the shared Ethernet channel are called the physical media. The physical cabling components vary depending on which kind of media system is in use. For instance, a twisted-pair cabling system uses different components than a fiber optic cabling system. Just to make things more interesting, a given Ethernet system may include several different kinds of media systems all connected together with repeaters to make a single network channel.

By using repeaters, an Ethernet system of multiple segments can grow as a branching tree. This means that each media segment is an individual branch of the complete signal system. The 10 Mbps system allows multiple repeaters in the path between stations, making it possible to build repeated networks with multiple branches. Only one or two repeaters can be used in the path of higher speed Ethernet systems, limiting the size of the resulting network. A typical network design actually ends up looking less like a tree and more like a complex set of network segments that may be strung along hallways or throughout wiring closets in your building. The resulting system of connected segments may grow in any direction and does not have a specific root segment. However, it is essential not to connect Ethernet segments in a loop, as each frame would circulate endlessly until the system was saturated with traffic.

Ethernet was designed to be easily expandable to meet the networking needs of a given site. As we've just seen, the total set of segments and repeaters in the Ethernet LAN must meet round-trip timing specifications. To help extend half-duplex Ethernet systems, networking vendors sell repeater hubs that are equipped with multiple Ethernet ports. Each port of a repeater hub links individual Ethernet media segments together to create a larger network that operates as a single Ethernet LAN.

There is another kind of hub called a switching hub. Switching hubs use the 48-bit Ethernet destination addresses to make a frame forwarding decision from one port of the switch to another. As shown in Figure 3, each port of a switching hub provides a connection to an Ethernet media system that functions as an entirely separate Ethernet LAN.

In a repeater hub the individual ports combine segments together to create a single LAN channel. However, a switching hub makes it possible to divide a set of Ethernet media systems into multiple separate LANs. The separate LANs are linked together by way of the switching electronics in the hub. The round-trip timing rules for each LAN stop at the switching hub port, allowing you to link a large number of individual Ethernet LANs together.

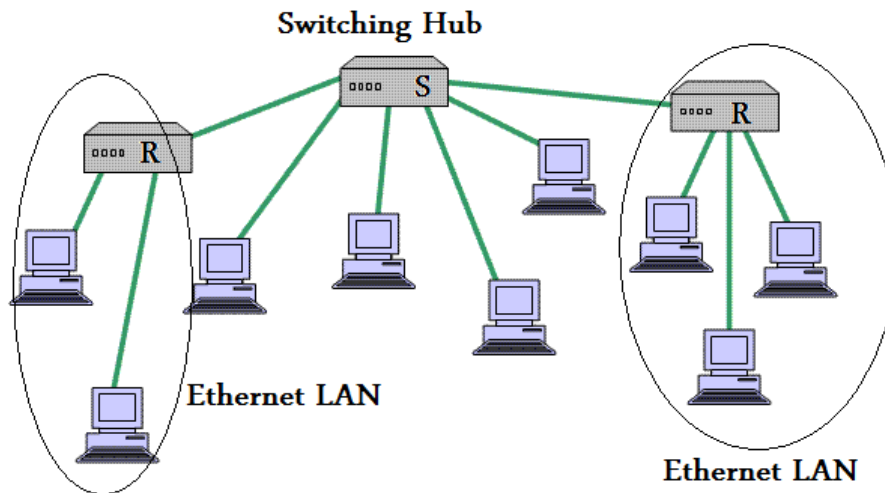


Figure 3. Switching Hub Creates Separate Ethernet LANs.

A given Ethernet LAN may consist of a repeater hub linking several media segments together. Whole Ethernet LANs can themselves be linked together to form extended network systems using switching hubs. Larger networks based on repeater hubs can be segmented into smaller LANs with switching hubs in a process called *network segmentation*. In this instance, the switching hub is used to segment a single LAN composed of network segments linked by repeater hubs into multiple LANs, to improve bandwidth and reliability.

Network segmentation can be extended all the way to connecting individual stations to single ports on the switching hub, in a process called micro-segmentation. As switching hub costs have dropped and computer performance has increased, more and more stations are being connected directly to their own port on the switching hub. This way, the station does not have to share the Ethernet channel bandwidth with another computer. Instead, it has its own dedicated Ethernet link to the switching hub.

Network Protocols and Ethernet

Data that is being sent between computers is carried in the data field of the Ethernet frame and structured as high-level network protocols. The high-level network protocol information carried inside the data field of each Ethernet frame is what actually establishes communications between applications running on computers attached to the network. The most widely used system of high-level network protocols is called the Transmission Control Protocol/ Internet Protocol (TCP/IP) suite.

The important thing to understand is that the high-level protocols are independent of the Ethernet system. There are several network protocols in use today, any of which

may send data between computers in the data field of an Ethernet frame. In essence, an Ethernet LAN with its hardware and Ethernet frame is simply a trucking service for data being sent by applications. The Ethernet LAN itself doesn't know or care about the high-level protocol data being carried in the data field of the Ethernet frame.

Since the Ethernet system is unaffected by the contents of the data field in the frame, different sets of computers running different high-level network protocols can share the same Ethernet. For example, you can have a single Ethernet that supports four computers, two of which communicate using TCP/IP, and two that use some other system of high-level protocols. All four computers can send Ethernet frames over the same Ethernet system without any problem.

Design of Network Protocols

To carry data between applications, the network software on your computer creates and sends a network protocol packet with its own private data field that corresponds to the message of the letter. The sender's and recipient's names (or protocol addresses) are added to complete the packet. After the network software has created the packet, the entire network protocol packet is stuffed into the data field of an Ethernet frame. Next, the 48-bit destination and source addresses are provided, and the frame is handed to the Ethernet interface and the Ethernet signal and cabling system for delivery to the right computer.

Figure 4 shows network protocol data traveling from Station A to Station B. The data is depicted as a letter that is placed in an envelope (i.e., a high-level protocol packet) that has network protocol addresses on it. This letter is stuffed into an Ethernet frame, shown here as a mailbag. The analogy is not exact, in that each Ethernet frame only carries one high-level protocol "letter" at a time and not a whole bagful, but you get the idea. The Ethernet frame is then placed on the Ethernet media system for delivery to Station B.

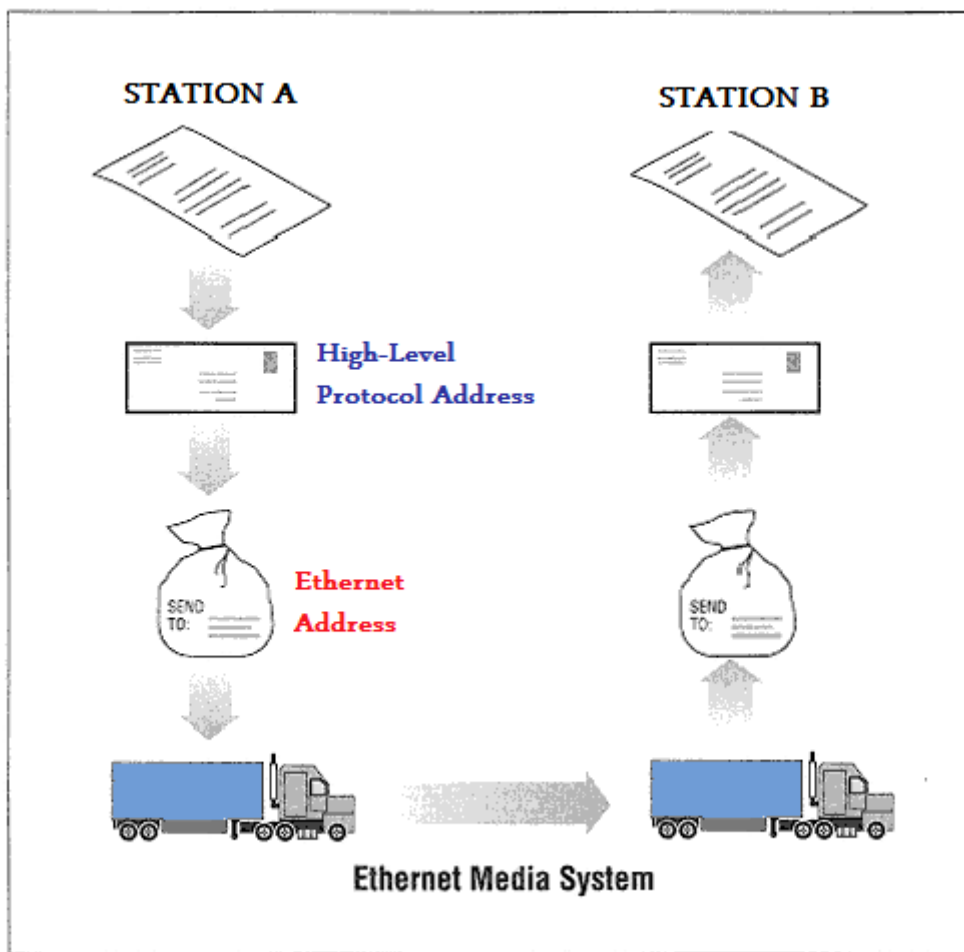


Figure 4. Ethernet and Networks

Internet Protocol and Ethernet Addresses

High-level network protocols have their own system of addresses, such as the 32-bit address used by the current version of the Internet Protocol (IPv4). The IP-based networking software in a given computer is aware of the 32-bit IP address assigned to that computer and can also read the 48-bit Ethernet address of its network interface. However, it doesn't know what the Ethernet addresses of the other stations on the network are when first trying to send a packet over the Ethernet.

To make things work, there needs to be a way to discover the Ethernet addresses of other IP-based computers on the network. The TCP/IP network protocol system accomplishes this task by using the Address Resolution Protocol (ARP).

1.1.1. Network Core: Circuit Switching & Packet Switching Circuit Switching

There are two fundamental approaches to moving data through a network of links and switches. These are "Circuit Switching and Packet Switching".

The term *circuit switching* refers to a communication mechanism that establishes a path between a sender and receiver with guaranteed isolation from paths used by other pairs of senders and receivers. Circuit switching is usually associated with telephone technology because a telephone system provides a dedicated connection between two telephones. In fact the term originated with early dialup telephone networks that were based on the use of circuit switching.

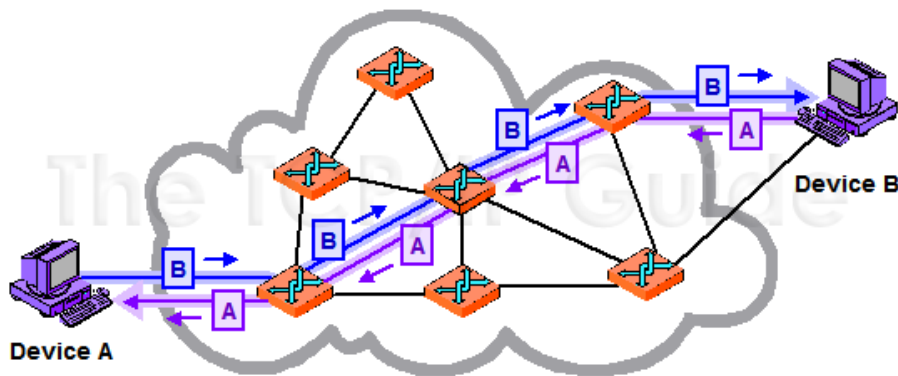


Figure 5. A circuit switched network that provides a direct connection between each pair of communicating entities

Currently, circuit switching networks use electronic devices to establish circuits. Furthermore, instead of having each circuit correspond to a physical path, multiple circuits are multiplexed over shared media, and the result is known as a *virtual circuit*. Thus, the distinction between circuit switching and other forms of networking does not arise from the existence of separate physical paths. Instead three general properties define a circuit switched paradigm:

- Point-to-Point Communication
- Separate steps for circuit creation, use and termination
- Performance equivalent to an isolated physical path.

The first property means that a circuit is formed between exactly two endpoints, and the second property distinguishes circuits that are switched to form circuits that are permanent. Switched circuits use a three-step process analogous to placing a phone call. In the first step, a circuit is established. In the second, two parties use the circuit to communicate, and in the third, the two parties terminate use.

The third property provides a crucial distinction between circuit switched networks and other types. Circuit switching means that the communication between two parties is not affected in any way by communication among other parties, even if all communication is multiplexed over a common medium. In particular, circuit switching must provide the illusion of an isolated path for each pair of communicating entities. Thus, techniques such as frequency division multiplexing or synchronous time division multiplexing must be used to multiplex circuits over a shared medium.

The point is:

Circuit switching provides the illusion of an isolated path between a pair of communicating entities; a path is created when needed, and discontinued after use.

A circuit in link is implemented with either Frequency Division Multiplexing (FDM) or Time Division Multiplexing (TDM). With FDM, the frequency spectrum of a link is shared among the connections established across the link. Specifically, the link dedicates a frequency band to each connection for the duration of the connection. In telephone networks, this frequency band typically has a width of 4 KHz. The width of the band is called, not surprisingly, the bandwidth. FM radio stations also use FDM to share microwave frequency spectrum.

The trend in modern technology is to replace FDM with TDM. The majority of the links in most telephone systems in the United States, Europe and in other developed countries currently employ. For a TDM link, time is divided into frames of fixed duration and each frame is divided into a fixed number of time slots. When the network establishes a connection across a link, the network dedicates one time slot in every frame to the connection. These slots are dedicated for the sole use of that connection, with a time slot available for use (in every frame) to transmit the data.

Figure 6 illustrates FDM and TDM for a specific network link. For FDM, the frequency domain is segmented into a number of circuits, each of bandwidth 4 KHz. For TDM, the time domain is segmented into four circuits; each circuit is equal to the frame rate multiplied by the number of bits in a slot. For example, if the link transmits 8,000 frames per second and each slot consists of 8 bits, then the transmission rate is 64 Kbps.

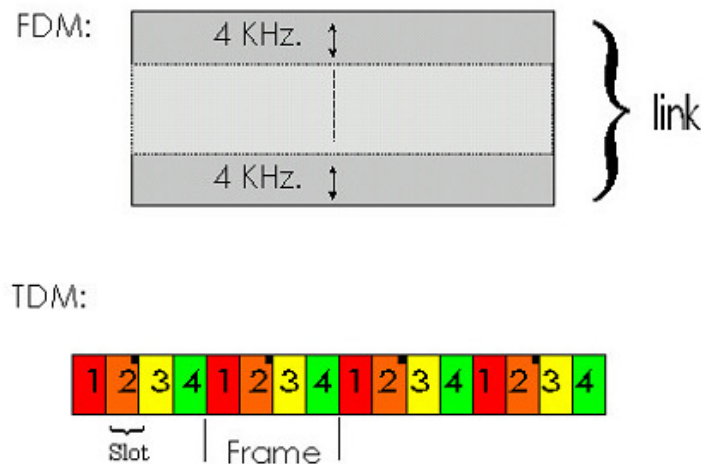


Figure 6. FDM and TDM

Proponents of packet switching have always argued that circuit switching is wasteful because the dedicated circuits are idle during silent periods. For example, when one of the conversants in a telephone call stops talking, the idle network resources (frequency bands or slots in the links along the connection's route) cannot be used by other ongoing connections. As another example of how these resources can be underutilized, consider a radiologist who uses a circuit-switched network to remotely access a series of x-rays. The radiologist sets up a connection, requests an image,

contemplates the image, and then requests a new image. Network resources are wasted during the radiologist's contemplation periods. Proponents of packet switching also enjoy pointing out that establishing end-to-end circuits and reserving end-to-end bandwidth is complicated and requires complex signaling software to coordinate the operation of the switches along the end-to-end path.

Packet Switching

In modern packet-switched networks, the source breaks long messages into smaller packets.

Between source and destination, each of these packets traverses communication links and packet switches (also known as routers). Packets are transmitted over each communication link at a rate equal to the full transmission rate of the link. Most packet switches use store and forward transmission at the inputs to the links. Store-and-forward transmission means that the switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link. Thus store-and-forward packet-switches introduce a store-and-forward delay at the input to each link along the packet's route. This delay is proportional to the packet's length in bits. In particular, if a packet consists of L bits, and the packet is to be forwarded onto an outbound link of R bps, then the store-and-forward delay at the switch is L/R seconds. Within each router there are multiple buffers (also called queues), with each link having an input buffer (to store packets that have just arrived to that link) and an output buffer. The output buffers play a key role in packet switching. If an arriving packet needs to be transmitted across a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output buffer queuing delays. These delays are variable and depend on the level of congestion in the network. Since the amount of buffer space is finite, an arriving packet may find that the buffer is completely filled with other packets waiting for transmission. In this case, packet loss will occur - either the arriving packet or one of the already-queued packets will be dropped. Packet loss is analogous to being told by the waiter that you must leave the premises because there are already too many other people waiting at the bar for a table.

Figure 7 illustrates a simple packet-switched network. Suppose Hosts A and B are sending packets to Host E. Hosts A and B first send their packets along 28.8 Kbps links to the first packet switch. The packet switch directs these packets to the 1.544 Mbps link. If there is congestion at this link, the packets queue in the link's output buffer before they can be transmitted onto the link. Consider now how Host A and Host B packets are transmitted onto this link. As shown in Figure 7, the sequence of A and B packets does not follow any periodic ordering; the ordering is random or statistical -- packets are sent whenever they happen to be present at the link. For this reason, we often say that packet switching employs *statistical multiplexing*. Statistical multiplexing sharply contrasts with *time-division multiplexing (TDM)*, for which each host gets the same slot in a revolving TDM frame.

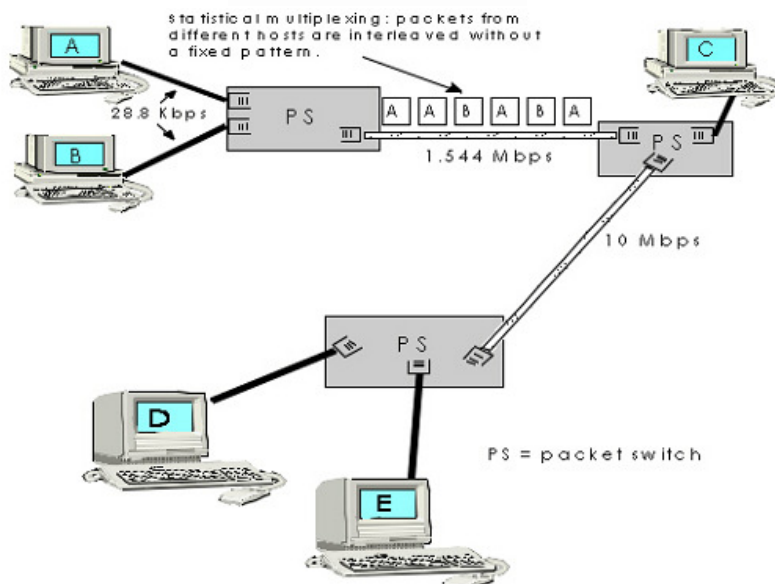


Figure 7. Packet-Switching

1.1.2. Overview of Delay in packet switched Networks

A packet starts in host (The source), and passes through a series of routers on its journey to another host (The destination). As a packet travels from one node (host or router) to the subsequent node (host or router) along this path, the packet suffers from several types of delays at each node along the path. The most important of these delays are the nodal processing delay, queuing delay, transmission delay and propagation delay. Together these delays accumulate to give a total nodal delay. In order to acquire a deeper understanding of packet switching and computer networks, we must understand the nature and importance of these delays.

Types of Delay:

1-Processing Delay (d_{proc}):

The time required to examine the packet's header and determine where to direct the packet is part of the processing delay. The processing delay can also include other factors, such as the time needed to check for bit-level errors in packets that might have occurred in transmitting the packet's bit from the upstream node to the router. A processing delay in high-speed routers is typically on the order of microseconds or less. After this nodal processing, the router directs the packet to the queue that precedes the link to the other router

2-Queuing Delay (d_{queue}):

At the queue, the packet experiences a queuing delay as it waits to be transmitted onwards. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission across the link. If the queue is empty and no other packet is currently being transmitted, then our packet's queuing delay will be zero. On the other hand, if traffic is heavy and many other packets are also waiting to be transmitted, the queuing delay will be long.



3-Transmission Delay (d_{trans}):

Assuming that packets are transmitted on a first come first served basis, as is common in packet-switched networks, our packet can be transmitted only after all the packets that have arrived before it have been transmitted. Denote the length of the packet by L bits, and denote transmission rate of link from router-1 to router-2 R bits/sec. For instance, for a 10Mbps. Ethernet link. The transmission delay is L/R . This is the amount of time required to push all of the packet's bits into the link. In practice, transmission delays are typically of the order of microseconds to milliseconds.

4- Propagation Delay (d_{prop}):

Once a bit is pushed into a link it needs to propagate to router 2. The time required to propagate from the beginning of the link to router 2 is the *propagation delay*. The bit propagates at the propagation speed of the link. The propagation speed depends on the physical medium of the *link but is in the range* of 2×10^8 meters/sec to 3×10^8 meters/sec which is equal to or a little less than the speed of light. The propagation delay is the distance between two routers divided by propagation speed. That is, propagation delay is d/s where d is the distance between two routers and s is the speed of propagation. In wide area networks, propagation delays are in the order of milliseconds.

The nodal delays accumulate and give an end-to-end delay

$$d_{nodal} = Q(d_{proc} + d_{queue} + d_{trans} + d_{prop})$$

where once again $d_{trans} = L/R$ and suppose there are $Q-1$ routers between source and destination.

2. Layers

Layer architecture allows discussing a well-defined, specific part of a large and complex system. The simplification itself is of considerable value by providing modularity, making it much easier to change the implementation of the service provided by the layer. As long as the layer provides the same service to the layer above it, and uses the same services from the layer below it, the remainder of the system remains unchanged when a layer's implementation is changed. (Changing the implementation of service is very different from changing the service itself). For large and complex systems that are constantly being updated, the ability to change the implementation of a service without affecting other components of the system is another important advantage of layering.

2.1. OSI Reference Model

The International Organization for Standardization (ISO) developed the OSI reference in the early 1980s. OSI is now the de facto standard for developing protocols that enable computers to communicate. Although not every protocol follows this model, most new protocols use this layered approach. In addition, when learning about networking, it is best to begin with this model to simplify understanding.

The OSI reference model breaks up the problem of intermachine communication into seven layers (see Figure 8). Each layer is concerned only with talking to its corresponding layer on the other machine. This means that Layer 5 has to worry only about talking to layer 5 on the receiving machine, and not what actual physical media might be.

In addition, each layer of the OSI Reference Model provides services to the layer above it and can request certain services from the layer directly below it.

This layer approach enables each layer to handle small pieces of information, make any necessary changes to the data, and add the necessary functions for that layer before passing the data along to the next layer. Data becomes less human-like and more computer-like the further down the OSI reference model it traverses, until it becomes 1s and 0s at the physical layer.

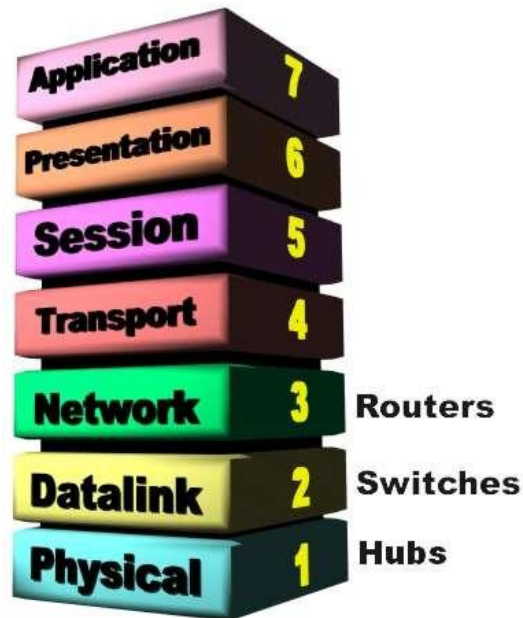


Figure 8. OSI Reference Model.

The Internet Protocol Suite with 5 layers maps to the corresponding OSI layers. From Figure 9, you will see how applications (FTP, E-mail) run atop protocols such as TCP before they are transmitted across some Layer 1 transport mechanism.

Internet Protocol Suite

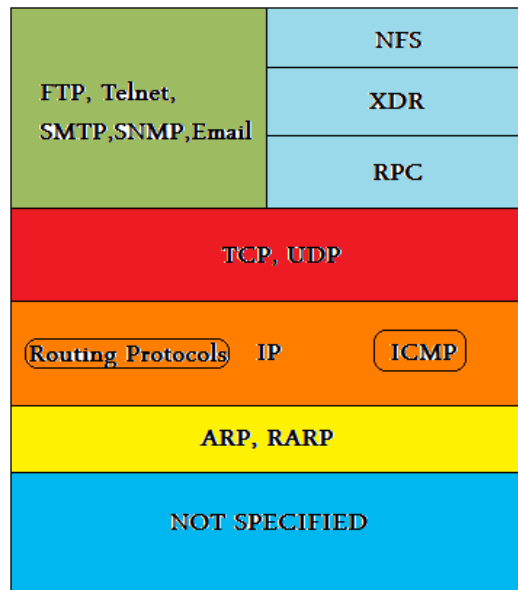


Figure 9. Internet Protocol Suite

2.1.1. Seven Layer ISO-OSI Reference Model

2.1.1.1. Application Layer

Most users are familiar with application layers. Some well-known applications include the following:

- E-mail
- Web Browsing
- Word Processing

2.1.1.2. Presentation Layer

The presentation layer ensures that information sent by the application layer of one system is readable by the application layer of another system. If necessary the presentation layer translates multiple data between formats by using a common data representation format.

The presentation layer concerns itself not only with the formal representation of actual user data, but also with data structures used by programs. Therefore, in addition to actual data format transformation, the presentation layer negotiates data transfer syntax for the application layer.

Common Examples include:

- Encryption
- Compression
- ASCII EBCDIC

2.1.1.3. Session Layer

At its name implies, the session layer establishes, manages and terminates between applications. A session consists of a dialogue between two or more presentation entities (recall that the session layer provides its services to the presentation layer). The Session layer synchronizes dialogue between presentation layer entities and manages their data exchange. In addition to basic regulation of conversations (sessions), the session layer offers provisions for data expedition and exception reporting of session-layer, presentation-layer and application-layer.

2.1.1.4. Transport Layer

The transport layer is responsible for ensuring reliable data transport on a network. This is accomplished through flow control, error checking (checksum), end-to-end acknowledgements, retransmissions and data sequencing.

Some transport layers, such as Transmission Control Protocol (TCP) have a mechanism for handling congestion. TCP adjusts its retransmission timer, for example, when congestion or packet loss occurs within a network. TCP slows down the amount of the traffic it sends when congestion is present. Congestions determined through the lack of acknowledgements, have been received from the destination node.

2.1.1.5. Network Layer

The network layer provides for logical addressing which enables two disparate systems on different logical Networks to determine a possible path to communicate. The network layer is the layer where routing protocols reside.

IP Addressing is by far the most common addressing scheme in use. Routing protocols such as Enhances Interior Gateway Protocol (Enhanced IGRP or EIGRP), Open Shortest Path First (OSPF), Border Gateway (BGP), Intermediary System To Intermediary System (IS-IS), and many others are used to determine the optimal routes between two logical sub-networks (Subnets).

Traditional routers route IP packets based on their Network layer address.

Key functions of the network layer include the following:

- Packet Formatting, addressing Networks and hosts, address resolution and routing.
- Creating and maintaining routing tables.

2.1.1.6. Data Link Layer

The data link layer provides reliable transport across a physical link. The link layer has its own addressing scheme. This addressing scheme is concerned with physical connectivity and can transport frames based upon the data link layer address.

Traditional ethernet switches switch network traffic based upon the data link layer address. Switching traffic based on a layer 2 address is generally known as *bridging*. In fact an ethernet switch is nothing more than a high-speed bridge with multiple interfaces.

2.1.1.7. Physical Layer

The physical layer is concerned with creating 1s and 0s on the physical medium with electrical impulses/voltages changes. Common physical layer communication specifications include the following:

- EIA/TIA-232: Electrical Industrial Association/Telecommunications Industry Association, a specification used for communicating between computing devices. This interface is often used for connecting computers to modems and might use different physical connectors.
- V.35: International Telecommunication Union Telecommunication Standardization Sector (ITU-T), a signaling mechanism that defines signaling rates from 192Kbps to 1,544 Mbps. This interface is a 34-pin and is also known as a Winchester Block.
- RS-499: Specification used for synchronous wide area communication. The physical connector uses a 37-pin and is capable of significantly longer runs than EIX/TIA-232
- 802,3: One of the most widely utilized physical mediums is ethernet. Currently, ethernet speeds are deployed from 10Mbps to 1000Mbps.

The transition to network switches enabled Networks to make better use of available bandwidth. Saving bandwidth was accomplished by preventing unnecessary IP packets from being transmitted to a physical port where the receiver device did not reside.

2.2. Five Layer Internet Protocol

2.2.1. Application Layer

The application layer is where networking applications and their application-layer protocols reside. The internet's application layer includes many protocols, such as the http protocol (which provides for web document request and transfer), SMTP (which provides for the transfer of e-mail messages), and FTP (which provides for the transfer of files between two end systems).

An application-layer protocol is distributed over multiple end systems, with the application in one end system using the protocol to exchange packets of information with the application in another end system.

2.2.2. Transport Layer

The Internet's transport layer transports application-layer messages between application end-points. In the Internet there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a connection-oriented service to applications. This service includes guaranteed delivery of application layer messages to the destination and flow control (that is sender/receiver speed matching). TCP breaks long messages into shorter segments and provides a congestion control mechanism, so that a source throttles (reduces) its transmission rate when the network is congested. The UD protocol provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control and no congestion control.



Transport-layer packets are also called *segments*.

2.2.3. Network Layer

The internet's network layer is responsible for moving network-layer packets known as *datagram's* from one host to another. The Internet transport-layer protocol is a source host which passes a transport-layer segment and destination address to the network layer. The network layer provides the service of delivering the segment to the transport layer in the destination host.

The Internet Network layer includes the celebrated IP protocol, which defines the fields in the datagram as well as how the end systems and routers act on these fields. There is only one IP protocol and Internet Components that have a network layer must run an IP protocol. The Internet's network layer also contains routing protocols that determine the routes that datagrams take between sources and destinations. The Internet has many routing protocols and the network layer contains both the IP protocol and numerous routing protocols.

2.2.4. Data Link Layer

The Internet's network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node (host or router) to the next node in the route, the network layer relies on the services of the link layer. In particular, at each node, the network layer passes the datagram down to the link layer, which delivers the datagram to the next node along the route. At this next node, the link passes the datagram up to the network layer.

The service provided by the link layer depends on the specific link-layer protocol that is employed over the link. As datagrams typically need to traverse several links to travel from source to destination, a datagram may be handled by different link-layer protocols at different links along its route.

Link-layer packets are also called *Frames*.

2.2.5. Physical Layer

While the job of the link layer is to move entire frames from one network element to an adjacent network element, the job of the physical layer is to move the individual bits within the frame from one node to the next node. The protocols in this link layer are again link dependent and further depend on the actual transmission medium of the link (twisted-pair copper wire, single mode fiber optics, etc). Ethernet has many physical-layer protocols: one for twisted pair copper wire, another for coaxial cable, another for fiber optics and so on.

2.3. Application Layer

2.3.1. HTTP (Hyper Text Transport Protocol)

The Hypertext Transfer Protocol (http), the Web's application-layer protocol, is at the heart of the Web. http is implemented in two programs: a client program and server program. The client program and server programs, executing on different end systems, talk to each other by exchanging http messages. http defines the structure of these messages and how the client and server exchange the messages. Before explaining http in detail, it is useful to review some Web terminology.

Http defines how web clients request Web pages from servers and how servers transfer Web pages to clients. We discuss the interaction between client and server in detail below, but the general idea is illustrated in Figure 10. When a user requests a Web page (e.g., clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects. Through 1997 essentially all browsers and Web servers implement version HTTP/1.0, which is defined in [RFC 1945]. Beginning in 1998 Web servers and browsers began to implement version HTTP/1.1, which is defined in [RFC 2068]. HTTP/1.1 is backward compatible with HTTP/1.0; a Web server running 1.1 can "talk" with a browser running 1.0, and a browser running 1.1 can "talk" with a server running 1.0.

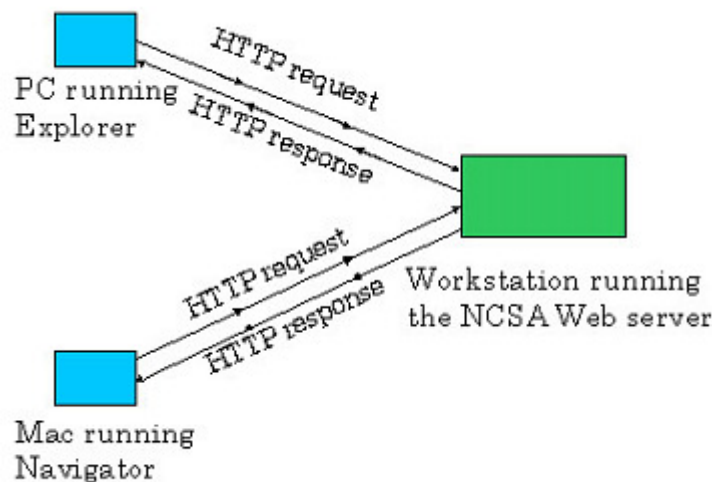


Figure 10. HTTP interaction between client and host.

Both HTTP/1.0 and HTTP/1.1 use TCP as their underlying transport protocol (rather than running on top of UDP). The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces. On the client side the socket interface is the "door" between the client process and the TCP connection; on the server side it is the "door" between the server process and the TCP connection. The client sends HTTP request messages into its socket interface and receives HTTP response messages from its socket interface. Similarly, the HTTP server receives request

messages from its socket interface and sends response messages into the socket interface. Once the client sends a message into its socket interface, the message is "out of the client's hands" and is "in the hands of TCP". TCP provides a reliable data transfer service to HTTP. This implies that each HTTP request message emitted by a client process eventually arrives intact at the server; similarly, each HTTP response message emitted by the server process eventually arrives intact at the client. Here we see one of the great advantages of a layered architecture - HTTP need not worry about lost data, or the details of how TCP recovers from loss or reordering of data within the network. That is the job of TCP and the protocols in the lower layers of the protocol stack.

We should also mention here that this mechanism forces each new TCP connection to initially transmit data at a relatively slow rate, but then allows each connection to ramp up to a relatively high rate when the network is uncongested. The initial slow-transmission phase is referred to as *slow start*.

It is important to note that the server sends requested files to clients without storing any state information about the client. If a particular client asks for the same object twice in a period of a few seconds, the server does not respond by saying that it just served the object to the client; instead, the server resends the object, as it has completely forgotten what it did earlier. Because an HTTP server maintains no information about the clients, HTTP is said to be a *stateless protocol*.

2.3.1.1. Non-Persistent and Persistent Connections

HTTP can use both non-persistent connections and persistent connections. Non-persistent connections are the default mode for HTTP/1.0. Conversely, persistent connections are the default mode for HTTP/1.1.

Non-Persistent Connections

Let us walk through the steps of transferring a Web page from server to client for the case of non-persistent connections. Suppose the page consists of a base HTML file and 10 JPEG images, and that all 11 of these objects reside on the same server.

Here is what happens:

1. The HTTP client initiates a TCP connection to the server www.yildiz.edu.tr Port number 80 is used as the default port number at which the HTTP server will be listening for HTTP clients that want to retrieve documents using HTTP.
2. The HTTP client sends a HTTP request message into the socket associated with the TCP connection that was established in step 1. The request message either includes the entire URL or simply the path name `/someDepartment/home.index`. (We will discuss the HTTP messages in some detail below.)
3. The HTTP server receives the request message via the socket associated with the connection that was established in step 1, retrieves the object `/someDepartment/home.index` from its storage (RAM or disk), encapsulates the object in a HTTP response message, and sends the response message into the TCP connection.



4. The HTTP server tells TCP to close the TCP connection. (But TCP doesn't actually terminate the connection until the client has received the response message intact.)
5. The HTTP client receives the response message. The TCP connection terminates. The message indicates that the encapsulated object is an HTML file. The client extracts the file from the response message, parses the HTML file and finds references to the ten JPEG objects.
6. The first four steps are then repeated for each of the referenced JPEG objects.

As the browser receives the Web page, it displays the page to the user. Two different browsers may interpret (i.e., display to the user) a Web page in somewhat different ways. HTTP has nothing to do with how a Web page is interpreted by a client. The HTTP specifications ([RFC 1945] and [RFC 2068]) only define the communication protocol between the client HTTP program and the server HTTP program. The steps above use non-persistent connections because each TCP connection is closed after the server sends the object -- the connection does not persist for other objects. Note that each TCP connection transports exactly one request message and one response message. Thus, in this example, when a user requests the Web page, 11 TCP connections are generated.

In the steps described above, we were intentionally vague about whether the client obtains the 10 JPEGs over ten serial TCP connections, or whether some of the JPEGs are obtained over parallel TCP connections. Indeed, users can configure modern browsers to control the degree of parallelism. In their default modes, most browsers open five to ten parallel TCP connections, and each of these connections handles one request-response transaction. If the user prefers, the maximum number of parallel connections can be set to one, in which case the ten connections are established serially. The use of parallel connections shortens the response time since it cuts out some of the RTT and slow-start delays. Parallel TCP connections can also allow the requesting browser to steal a larger share of the end-to-end transmission bandwidth.

Before continuing, let's do a 'back of the envelope' calculation to estimate the amount of time from when a client requests the base HTML file until the file is received by the client. To this end we define the round-trip time (RTT), which is the time it takes for a small packet to travel from client to server and then back to the client. The RTT includes packet propagation delays, packet queuing delays in intermediate routers and switches, and packet processing delays. Now consider what happens when a user clicks on a hyperlink. This causes the browser to initiate a TCP connection between the browser and the Web server; this involves a "three-way handshake" -- the client sends a small TCP message to the server, the server acknowledges and responds with a small message, and finally the client acknowledges back to the server. One RTT elapses after the first two parts of the three-way handshake. After completing the first two parts of the handshake, the client sends the HTTP request message into the TCP connection, and TCP "piggybacks" the last acknowledgment (the third part of the three-way handshake) onto the request message. Once the request message arrives at the server, the server sends the HTML file into the TCP connection. This HTTP request/response eats up another RTT. Thus, roughly, the total response time is 2RTT plus the transmission time at the server of the HTML file.



Non-persistent connections have some shortcomings. First, a brand new connection must be established and maintained for each requested object. For each of these connections, TCP buffers must be allocated and TCP variables must be kept in both the client and server. This can place a serious burden on the Web server, which may be serving requests from hundreds of different clients simultaneously. Second, as we just described, each object suffers two RTTs -- one RTT to establish the TCP connection and one RTT to request and receive an object. Finally, each object suffers from TCP slow start because every TCP connection begins with a TCP slow-start phase. However, the accumulation of RTT and slow start delays is partially alleviated by the use of parallel TCP connections.

Persistent Connections

With persistent connections, the server leaves the TCP connection open after sending responses. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page (in the example above, the base HTML file and the ten images) can be sent over a single persistent TCP connection; moreover, multiple Web pages residing on the same server can be sent over one persistent TCP connection. Typically, the HTTP server closes the connection when it isn't used for a certain time (the timeout interval), which is often configurable. There are two versions of persistent connections: without pipelining and with pipelining. For the version without pipelining, the client issues a new request only when the previous response has been received. In this case, each of the referenced objects (the ten images in the example above) experiences one RTT in order to request and receive the object. Although this is an improvement over non-persistent two RTTs, the RTT delay can be further reduced with pipelining. Another disadvantage of no pipelining is that after the server sends an object over the persistent TCP connection, the connection *hangs* -- does nothing -- while it waits for another request to arrive. This hanging wastes server resources.

The default mode of HTTP/1.1 uses persistent connections with pipelining. In this case, the HTTP client issues a request as soon as it encounters a reference. Thus the HTTP client can make back-to-back requests for the referenced objects. When the server receives the requests, it can send the objects back-to-back. If all the requests are sent back-to-back and all the responses are sent back-to-back, then only one RTT is expended for all the referenced objects (rather than one RTT per referenced object when pipelining isn't used). Furthermore, the pipelined TCP connection hangs for a smaller fraction of time. In addition to reducing RTT delays, persistent connections (with or without pipelining) have a smaller slow-start delay than non-persistent connections. This is because after sending the first object, the persistent server does not have to send the next object at the initial slow rate since it continues to use the same TCP connection. Instead, the server can pick up at the rate where the first object left off.

2.3.2. FTP (File Transfer Protocol)

FTP (File Transfer Protocol) is a protocol for transferring a file from one host to another host. The protocol dates back to 1971 (when the Internet was still an experiment), but remains enormously popular. FTP is described in [RFC 959]. Figure 11 provides an overview of the services provided by FTP.

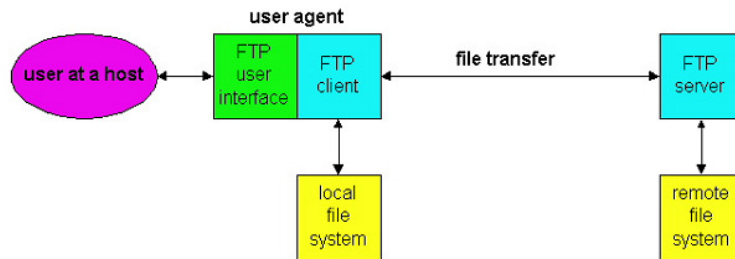


Figure 11. FTP moves files between local and remote file systems.

In a typical FTP session, the user is sitting in front of one host (the local host) and wants to transfer files to or from a remote host. In order for the user to access the remote account, the user must provide user identification and a password. After providing this authorization information, the user can transfer files from the local file system to the remote file system and vice versa. As shown in Figure 11, the user interacts with FTP through an FTP user agent. The user first provides the hostname of the remote host, which causes the FTP client process in the local host to establish a TCP connection with the FTP server process in the remote host. The user then provides the user identification and password, which get sent over the TCP connection as part of FTP commands. Once the server has authorized the user, the user copies one or more files stored in the local file system into the remote file system (or vice versa).

HTTP and FTP are both file transfer protocols and have many common characteristics; for example, they both run on top of TCP, the Internet's connection-oriented, transport-layer, reliable data transfer protocol. However, the two application-layer protocols have some important differences. The most striking difference is that FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection. The control connection is used for sending control information between the two hosts -- information such as user identification, password, commands to change remote directory, and commands to "put" and "get" files. The data connection is used to actually send a file. Because FTP uses a separate control connection, FTP is said to send its control information out-of-band. http recall sends request and response header lines into the same TCP connection that carries the transferred file itself. For this reason, HTTP is said to send its control information in-band.

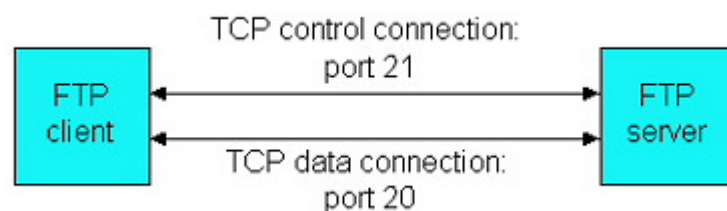


Figure 12. FTP control and data connections



When a user starts an FTP session with a remote host, FTP first sets up a control TCP connection on server port number 21. The client side of FTP sends the user identification and password over this control connection. The client side of FTP also sends, over the control connection, commands to change the remote directory. When the user requests a file transfer (either to, or from, the remote host), FTP opens a TCP data connection on server port number 20. FTP sends exactly one file over the data connection and then closes the data connection. If, during the same session, the user wants to transfer another file, FTP opens another data TCP connection. Thus, with FTP, the control connection remains open throughout the duration of the user session, but a new data connection is created for each file transferred within a session (i.e., the data connections are non-persistent).

Throughout a session, the FTP server must maintain state about the user. In particular, the server must associate the control connection with a specific user account, and the server must keep track of the user's current directory as the user wanders about the remote directory tree. Keeping track of this state information for each ongoing user session significantly impedes the total number of sessions that FTP can maintain simultaneously. HTTP, on the other hand, is stateless -- it does not have to keep track of any user state.

FTP Commands and Replies

We end this section with a brief discussion of some of the more common FTP commands. The commands, from client to server, and replies, from server to client, are sent across the control TCP connection in 7-bit ASCII format. Thus, like HTTP commands, FTP commands are readable by people. In order to delineate successive commands, a carriage return and line feed end each command (and reply). Each command consists of four uppercase ASCII characters, some with optional arguments. Some of the more common commands are given below (with options in *italics*):

- USER *username*: Used to send the user identification to server.
- PASS *password*: Used to send the user password to the server.
- LIST: Used to ask the server to send back a list of all the files in the current remote directory. The list of files is sent over a (new and non-persistent) data TCP connection and not over the control TCP connection.
- RETR *filename*: Used to retrieve (i.e., get) a file from the current directory of the remote host.
- STOR *filename*: Used to store (i.e., put) a file into the current directory of the remote host.

There is typically a one-to-one correspondence between the command that the user issues and the FTP command sent across the control connection. Each command is followed by a reply, sent from server to client. The replies are three-digit numbers, with an optional message following the number. This is similar in structure to the status code and phrase in the status line of the HTTP response message; the inventors of HTTP intentionally included this similarity in the HTTP response messages. Some typical replies, along with their possible messages, are as follows:



- 404 have not found anything matching the request URL.
- 331 Username OK, password required
- 125 Data connection already open; transfer starting
- 425 Can't open data connection
- 452 Error writing file

Readers who are interested in learning about the other FTP commands and replies are encouraged to read [RFC 959].

2.3.3. SMTP (Simple Mail Transfer Protocol)

SMTP, defined in [RFC 821], is at the heart of Internet electronic mail. As mentioned above, SMTP transfers messages from senders' mail servers to the recipients' mail servers. SMTP is much older than HTTP. (The SMTP RFC dates back to 1982, and SMTP was around long before that) Although SMTP has numerous wonderful qualities, as evidenced by its ubiquity in the Internet, it is nevertheless a legacy technology that possesses certain "archaic" characteristics. For example, it restricts the body (not just the headers) of all mail messages to be in simple seven-bit ASCII. This restriction was not bothersome in the early 1980s when transmission capacity was scarce and no one was emailing large attachments or large image, audio or video files. But today, in the multimedia era, the seven-bit ASCII restriction is a bit of a pain -- it requires binary multimedia data to be encoded to ASCII before being sent over SMTP; and it requires the corresponding ASCII message to be decoded back to binary after SMTP transport.

Let's now take a closer look at how SMTP transfers a message from a sending mail server to a receiving mail server. We will see that the SMTP protocol has many similarities with protocols that are used for face-to-face human interaction. First, the client SMTP (running on the sending mail server host) has TCP establish a connection on port 25 to the server SMTP (running on the receiving mail server host). If the server is down, the client tries again later. Once this connection is established, the server and client perform some application-layer handshaking. Just as humans often introduce themselves before transferring information from one to another, SMTP clients and servers introduce themselves before transferring information. During this SMTP handshaking phase, the SMTP client indicates the email address of the sender (the person who generated the message) and the email address of the recipient. Once the SMTP client and server have introduced themselves to each other, the client sends the message. SMTP can count on the reliable data transfer service of TCP to get the message to the server without errors. The client then repeats this process over the same TCP connection if it has other messages to send to the server; otherwise, it instructs TCP to close the connection.

2.3.4. DNS (Domain Name Service)

One identifier for a host is its hostname. Hostnames -- such as cnn.com, www.yahoo.com, gaia.cs.umass.edu and surf.eurecom.fr -- are mnemonic and are therefore appreciated by humans. However, hostnames provide little, if any, information about the location within the Internet of the host. (A hostname such as surf.eurecom.fr, which ends with the country code .fr, tells us that the host is in France,



but doesn't say much more.) Furthermore, because hostnames can consist of variable-length alpha-numeric characters, they would be difficult to process by routers. For these reasons, hosts are also identified by so-called IP addresses. An IP address consists of four bytes and has a rigid hierarchical structure. An IP address looks like 121.7.106.83, where each period separates one of the bytes expressed in decimal notation from 0 to 127. An IP address is hierarchical because as we scan the address from left to right, we obtain more and more specific information about where (i.e., within which network, in the network of networks) the host is located in the Internet. (Just as when we scan a postal address from bottom to top we obtain more and more specific information about where the residence is located). An IP address is included in the header of each IP datagram, and Internet routers use this IP address to route a datagram towards its destination.

There are two ways to identify a host -- a hostname and an IP address. People prefer the more mnemonic hostname identifier, while routers prefer fixed-length, hierarchically-structured IP addresses. In order to reconcile these different preferences, we need a directory service that translates hostnames to IP addresses. This is the main task of the Internet's Domain Name System (DNS). The DNS is a distributed database implemented in a hierarchy of name servers and an application-layer protocol that allows hosts and name servers to communicate in order to provide the translation service. Name servers are usually UNIX machines running the Berkeley Internet Name Domain (BIND) software. The DNS protocol runs over UDP and uses port 53.

DNS is commonly employed by other application-layer protocols -- including HTTP, SMTP and FTP - to translate user-supplied host names to IP addresses. As an example, consider what happens when a browser (i.e., an HTTP client), running on some user's machine, requests the URL `www.yildiz.edu.tr`. In order for the user's machine to be able to send an HTTP request message to the Web server `www.yildiz.edu.tr`, the user's machine must obtain the IP address of `www.yildiz.edu.tr`. This is done as follows. The same user machine runs the client-side of the DNS application. The browser extracts the hostname, `www.yildiz.edu.tr`, from the URL and passes the hostname to the client-side of the DNS application. As part of a DNS query message, the DNS client sends a query containing the hostname to a DNS server. The DNS client eventually receives a reply, which includes the IP address for the hostname. The browser then opens a TCP connection to the HTTP server process located at that IP address. All IP datagrams sent to from the client to server as part of this connection will include this IP address in the destination address field of the datagrams. In particular, the IP datagram(s) that encapsulate the HTTP request message use this IP address. We see from this example that DNS adds an additional delay -- sometimes substantial -- to the Internet applications that use DNS. Fortunately, the desired IP address is often cached in a "near by" DNS name server, which helps to reduce the DNS network traffic as well as the average DNS delay.

Like HTTP, FTP, and SMTP, the DNS protocol is an application-layer protocol since it runs between communicating end systems (again using the client-server paradigm), and it relies on an underlying end-to-end transport protocol (i.e., UDP) to transfer DNS messages between communicating end systems. In another sense, however, the role of the DNS is quite different from Web, file transfer, and email applications. Unlike



these applications, the DNS is not an application with which a user directly interacts. Instead, the DNS provides a core Internet function -- namely, translating hostnames to their underlying IP addresses, for user applications and other software in the Internet. The DNS, which implements the critical name-to-address translation process using clients and servers located at the edge of the network.

DNS provides a few other important services in addition to translating hostnames to IP addresses:

- **Host Aliasing:** A host with a complicated hostname can have one or more alias names. For example, a hostname such as relay1.west-coast.enterprise.com could have, say, two aliases such as enterprise.com and www.enterprise.com. In this case, the hostname relay1.west-coast.enterprise.com is said to be canonical hostname. Alias hostnames, when present, are typically more mnemonic than a canonical hostname. DNS can be invoked by an application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host.
- **Mail Server Aliasing:** For obvious reasons, it is highly desirable that email addresses be mnemonic. For example, if Bob has an account with Hotmail, Bob's email address might be as simple as bob@hotmail.com. However, the hostname of the Hotmail mail server is more complicated and much less mnemonic than simply hotmail.com (e.g., the canonical hostname might be something like relay1.west-coast.hotmail.com). DNS can be invoked by a mail application to obtain the canonical hostname for a supplied alias hostname as well as the IP address of the host. In fact, DNS permits a company's mail server and Web server to have identical (aliased) hostnames; for example, a company's Web server and mail server can both be called enterprise.com.
- **Load Distribution:** Increasingly, DNS is also being used to perform load distribution among replicated Web servers for example. Busy sites, such as cnn.com, are replicated over multiple servers, with each server running on a different end system, and having a different IP address. For replicated Web servers, a set of IP addresses is thus associated with one canonical hostname. The DNS database contains this set of IP addresses. When clients make a DNS query for a name mapped to a set of addresses, the server responds with the entire set of IP addresses, but rotates the ordering of the addresses within each reply. Because a client typically sends its HTTP request message to the IP address that is listed first in the set, DNS rotation distributes the traffic among all the replicated servers. DNS rotation is also used for email so that multiple mail servers can have the same alias name.

2.4. Transport Layer

A transport layer protocol provides for logical communication between application processes running on different hosts. By "logical" communication, we mean that although the communicating application processes are not physically connected to each other (indeed, they may be on different sides of the planet, connected via numerous routers and a wide range of link types), from the applications' viewpoint, it is as if they were physically connected. Application processes use the logical communication provided by the transport layer to send messages to each other, free of worry about the details of the physical infrastructure used to carry these messages. Figure 13 illustrates the notion of logical communication.

As shown in Figure 13, transport layer protocols are implemented in the end systems but not in network routers. Network routers only act on the network-layer fields of the layer-3 PDUs; they do not act on the transport-layer fields.

At the sending side, the transport layer converts the messages it receives from a sending application process into 4-PDUs (that is, transport-layer protocol data units). This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create 4-PDUs. The transport layer then passes the 4-PDUs to the network layer, where each 4-PDU is encapsulated into a 3-PDU. At the receiving side, the transport layer receives the 4-PDUs from the network layer, removes the transport header from the 4-PDUs, reassembles the messages and passes them to a receiving application process.

A computer network can make more than one transport layer protocol available to network applications. For example, the Internet has two protocols -- TCP and UDP. Each of these protocols provides a different set of transport layer services to the invoking application.

All transport layer protocols provide an application multiplexing/demultiplexing service. In addition to multiplexing/demultiplexing service, a transport protocol can possibly provide other services to invoking applications, including reliable data transfer, bandwidth guarantees, and delay guarantees.

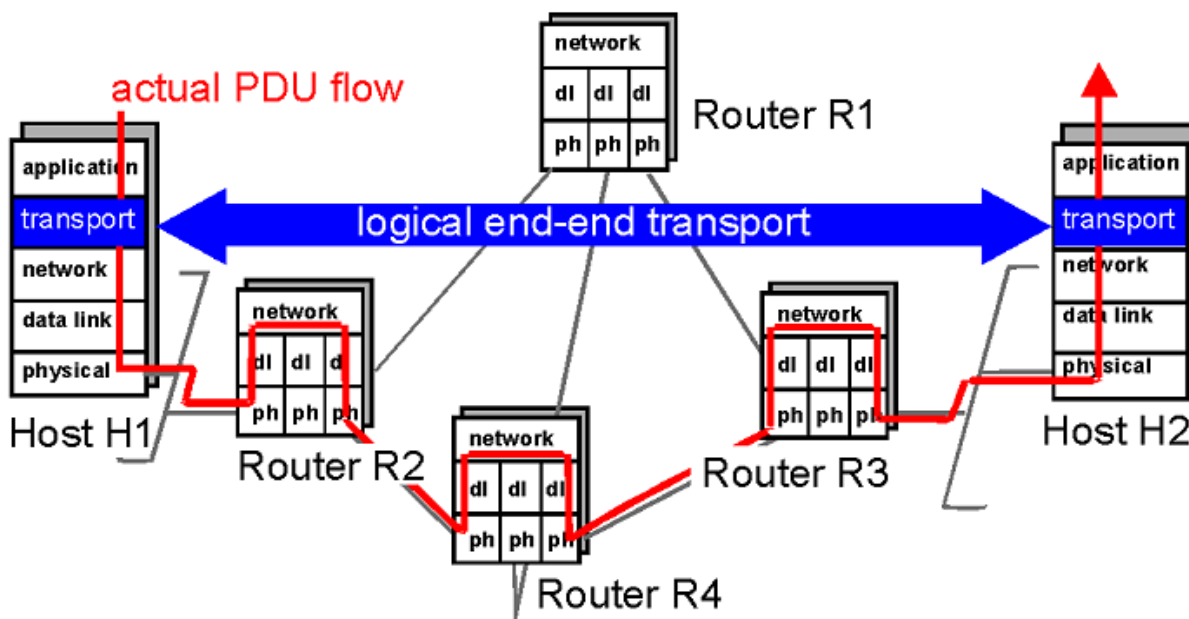


Figure 13. Transport Layer

2.4.1. Multiplexing and Demultiplexing

Each transport-layer segment has a field that contains information that is used to determine the process to which the segment's data is to be delivered. At the receiving end, the transport layer can then examine this field to determine the receiving process, and then direct the segment to that process. This job of delivering the data in a transport-layer segment to the correct application process is called *demultiplexing*. The job of gathering data at the source host from different application processes, enveloping the data with header information (which will later be used in demultiplexing) to create segments, and passing the segments to the network layer is called *multiplexing*.

UDP and TCP perform the demultiplexing and multiplexing jobs by including two special fields in the segment headers: the source port number field and the destination port number field. These two fields are illustrated in Figure 14. When taken together, the fields uniquely identify an application process running on the destination host.

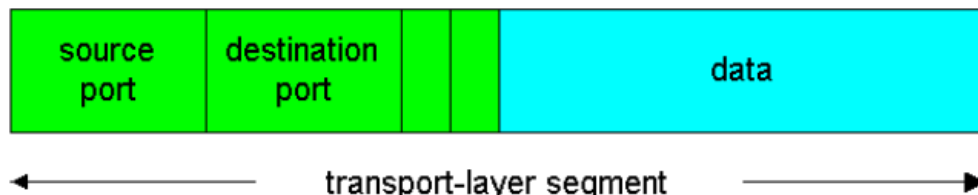


Figure 14. Transport Layer Segment.

2.4.2. Connectionless Transport: UDP (User Datagram Protocol)

UDP, defined in [RFC 768], does just about as little as a transport protocol can. Aside from the multiplexing/demultiplexing function and some light error checking, it adds nothing to IP. In fact, if the application developer chooses UDP instead of TCP, then the application is talking almost directly with IP. UDP takes messages from an application process, attaches source and destination port number fields for the multiplexing/demultiplexing service, adds two other fields of minor importance, and passes the resulting "segment" to the network layer. The network layer encapsulates the segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the port numbers and the IP source and destination addresses to deliver the data in the segment to the correct application process. Note that with UDP there is no handshaking between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be connectionless.

- **No Connection Establishment:** TCP uses a three-way handshake before it starts to transfer data. UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection. This is probably the principle reason why DNS runs over UDP rather than TCP -- DNS would be much slower if it



ran over TCP. HTTP uses TCP rather than UDP, since reliability is critical for Web pages with text. The TCP connection establishment delay in HTTP is an important contributor to the "world wide wait".

- **No Connection State:** TCP maintains connection state in the end systems. This connection state includes receive and send buffers, congestion control parameters, and sequence and acknowledgment number parameters. The state information is needed to implement TCP's reliable data transfer service and to provide congestion control. UDP, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.
- **Small Segment Header Overhead:** The TCP segment has 20 bytes of header overhead in every segment, whereas UDP only has 8 bytes of overhead.
- **Unregulated Send Rate:** TCP has a congestion control mechanism that throttles the sender when one or more links between sender and receiver becomes excessively congested. This throttling can have a severe impact on real-time applications, which can tolerate some packet loss but require a minimum send rate. On the other hand, the speed at which UDP sends data is only constrained by the rate at which the application generates data, the capabilities of the source (CPU, clock rate, etc.) and the access bandwidth to the Internet. We should keep in mind, however, that the receiving host does not necessarily receive all the data - when the network is congested, a significant fraction of the UDP-transmitted data could be lost due to router buffer overflow. Thus, the receive rate is limited by network congestion even if the sending rate is not constrained.

UDP is also commonly used today with multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video. We just mention now that all of these applications can tolerate a small fraction of packet loss, so that reliable data transfer is not absolutely critical for the success of the application. Furthermore, interactive real-time applications, such as Internet phone and video conferencing, react very poorly to TCP's congestion control. For these reasons, developers of multimedia applications often choose to run the applications over UDP instead of TCP. Finally, because TCP cannot be employed with multicast, multicast applications run over UDP.

Although commonly done today, running multimedia applications over UDP is controversial to say the least. As we mentioned above, UDP lacks any form of congestion control. But congestion control is needed to prevent the network from entering a congested state in which very little useful work is done. If everyone were to start streaming high bit-rate video without using any congestion control, there would be so much packet overflow at routers that no one would see anything. Thus, the lack of congestion control in UDP is a potentially serious problem. Many researchers have proposed new mechanisms to force all sources, including UDP sources, to perform adaptive congestion control.

Table 1 shows some popular internet applications and their underlying transport protocols.

Table 1. Popular Internet Applications and their underlying transport protocols.

Application	Application-Layer Protocol	Underlying Transport Protocol
electronic mail	SMTP	TCP
remote terminal access	Telnet	TCP
Web	HTTP	TCP
File Transfer	FTP	TCP
Remote File Server	NFS	typically UDP
Streaming Multimedia	proprietary	typically UDP
Internet Telephony	proprietary	typically UDP
Network Management	SNMP	typically UDP
Routing Protocol	RIP	typically UDP
Name Translation	DNS	typically UDP

UDP Segment Structure

The UDP segment structure is defined in [RFC 768].

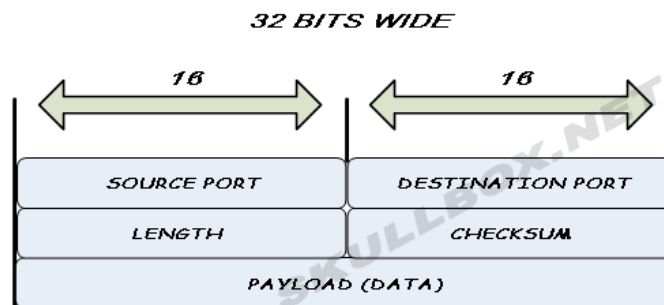


Figure 15. UDP Segment Structure

The application data occupies the data field of the UDP datagram. For example, for DNS, the data field contains either a query message or a response message. For a streaming audio application, audio samples fill the data field. The UDP header has only four fields, each consisting of four bytes. The port numbers allow the destination host to pass the application data to the correct process running on that host (i.e., perform the demultiplexing function). The checksum is used by the receiving host to check if errors have been introduced into the segment during the course of its transmission from source to destination.



2.4.3. Connection-Oriented Transport (Transmission Control Protocol)

The term *connection-oriented* means the two applications using TCP (normally considered a client and a server) must establish a TCP connection with each other before they can exchange data. The typical analogy is dialing a telephone number, waiting for the other party to answer the phone and say "hello," and then saying who's calling.

TCP connection is also always point-to-point, i.e., between a single sender and a single receiver. So called "multicasting" -- the transfer of data from one sender to many receivers in a single send operation -- is not possible with TCP. Let us now take a look at how a TCP connection is established. Suppose a process running in one host wants to initiate a connection with another process in another host. Recall that the host that is initiating the connection is called the client host, while the other host is called the server host. The client application process first informs the client TCP that it wants to establish a connection to a process in the server.

TCP provides reliability by doing the following:

- The application data is broken into what TCP considers the best sized chunks to send. This is totally different from UDP, where each is written by the application generates a UDP datagram of that size. The unit of information passed by TCP to IP is called a segment and TCP decides what this segment size is.
- When TCP sends a segment it maintains a timer, waiting for the other end to acknowledge reception of the segment. If an acknowledgment isn't received in time, the segment is retransmitted. TCP has adaptive timeout and retransmission strategy.
- When TCP receives data from the other end of the connection, it sends an acknowledgment. This acknowledgment is not sent immediately, but normally delayed a fraction of a second.
- TCP maintains a checksum on its header and data. This is an end-to-end checksum whose purpose is to detect any modification of the data in transit. If a segment arrives with an invalid checksum, TCP discards it and doesn't acknowledge receiving it - it expects the sender to time out and retransmit.
- Since TCP segments are transmitted as IP datagrams, and since IP datagrams can arrive out of order, TCP segments can arrive out of order. A receiving TCP re-sequences the data if necessary, passing the received data in the correct order to the application.
- Since IP datagrams can get duplicated, a receiving TCP must discard duplicate data.
- TCP also provides flow control. Each end of a TCP connection has a finite amount of buffer space. A receiving TCP only allows the other end to send as much data as the receiver has buffers for. This prevents a fast host from taking all the buffers on a slower host.

A stream of 8-bit bytes is exchanged across the TCP connection between the two applications. There are no record markers automatically inserted by TCP. This is what we called a *byte stream service*. If the application on one end writes 10 bytes, followed by a write of 20 bytes, followed by a write of 50 bytes, the application at the other end of the connection cannot tell what size 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 appear at the other end.

Also, TCP does not interpret the contents of the bytes at all. TCP has no idea if the data bytes being exchanged are binary data, ASCII characters, EBCDIC characters, or whatever. The interpretation of this byte stream is up to the applications on each end of the connection.

The TCP in the client then proceeds to establish a TCP connection with the TCP in the server. It suffices to know that the client first sends a special TCP segment; the server responds with a second special TCP segment; and finally the client responds again with a third special segment. The first two segments contain no "payload," i.e., no application-layer data; the third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection establishment procedure is often referred to as a *three-way handshake*.

Once a TCP connection is established, the two application processes can send data to each other; because TCP is full-duplex they can send data at the same time. Let us consider the sending of data from the client process to the server process. The client process passes a stream of data through the socket (the door of the process). Once the data passes through the door, the data is now in the hands of TCP running in the client. TCP directs this data to the connection's send buffer, which is one of the buffers that are set aside during the initial three-way handshake. From time to time, TCP will "grab" chunks of data from the send buffer. The maximum amount of data that can be grabbed and placed in a segment is limited by the Maximum Segment Size (MSS). The MSS depends on the TCP implementation (determined by the operating system) and can often be configured; common values are 1,500 bytes, 536 bytes and 512 bytes. Note that the MSS is the maximum amount of application-level data in the segment, not the maximum size of the TCP segment including headers. (This terminology is confusing, but we have to live with it, as it is well entrenched.)

2.4.3.1. TCP Segment Structure

The original specification for TCP is RFC 793 [Postel 1981c], although some errors in that RFC are corrected in the Host Requirements RFC.

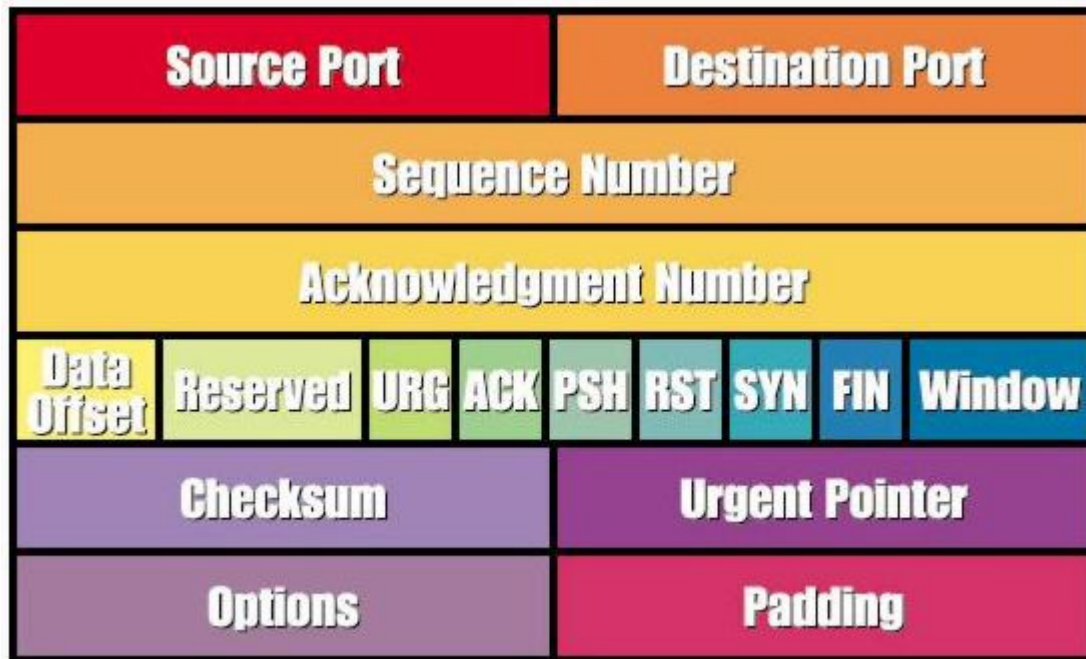


Figure 16. TCP Segment Structure

Figure 16 shows the structure of a segment packaged and sent over the TCP protocol. Each TCP segment contains the source and destination port number to identify the sending and receiving application. These two values, along with the source and destination IP addresses in the IP header, uniquely identify each connection.

The sequence number identifies the byte in the stream of data from the sending TCP to the receiving TCP that the first byte of data in this segment represents. If we consider the stream of bytes flowing in one direction between two applications, TCP numbers each byte with a sequence number. This sequence number is a 32-bit unsigned number that wraps back around to 0 after reaching $2^{32}-1$.

When a new connection is being established, the SYN flag is turned on. The sequence number field contains the initial sequence number (ISN) chosen by this host for this connection. The sequence number of the first byte of data sent by this host will be the ISN plus one because the SYN flag consumes a sequence number.

Since every byte that is exchanged is numbered, the acknowledgment number contains the next sequence number that the sender of the acknowledgment expects to receive. This is therefore the sequence number plus 1 of the last successfully received byte of data. This field is valid only if the ACK flag (described below) is on.

Sending an ACK costs nothing because the 32-bit acknowledgment number field is always part of the header, as is the ACK flag. Therefore we'll see that once a connection is established, this field is always set and the ACK flag is always on.

TCP provides a full-duplex service to the application layer. This means that data can be flowing in each direction, independent of the other direction. Therefore, each end of a connection must maintain a sequence number of the data flowing in each direction.



TCP can be described as a sliding-window protocol without selective or negative acknowledgments. TCP lacks selective acknowledgments because the acknowledgment number in the TCP header means that the sender has successfully received up to, but not including, that byte. There is currently no way to acknowledge selected pieces of the data stream. For example, if bytes 1-1024 are received OK and the next segment contains bytes 2049-3072, the receiver cannot acknowledge this new segment. All it can send is an ACK with 1025 as the acknowledgment number. There is no means for negatively acknowledging a segment. For example, if the segment with bytes 1025-2048 did arrive, but had a checksum error, all the receiving TCP can send is an ACK with 1025 as the acknowledgment number.

The header length gives the length of the header in 32-bit words. This is required because the length of the options field is variable. With a 4-bit field, TCP is limited to a 60-byte header. Without options, however, the normal size is 20 bytes.

There are six flag bits in the TCP header. One or more of them can be turned on at the same time.

URG: The urgent pointer is valid.

ACK: The acknowledgment number is valid.

PSH: The receiver should pass this data to the application as soon as possible.

RST: Reset the connection.

SYN: Synchronize sequence numbers to initiate a connection.

FIN: The sender is finished sending data.

TCP's flow control is provided by each end advertising a window size. This is the number of bytes, starting with the one specified by the acknowledgment number field, that the receiver is willing to accept. This is a 16-bit field, limiting the window to 65535 bytes. The new window scales option that allows this value to be scaled, providing larger windows.

The checksum covers the TCP segment: the TCP header and the TCP data. This is a mandatory field that must be calculated and stored by the sender, and then verified by the receiver. The TCP checksum is calculated similarly to the UDP checksum.

The urgent pointer is valid only if the URG flag is set. This pointer is a positive offset that must be added to the sequence number field of the segment to yield the sequence number of the last byte of urgent data. TCP's urgent mode is a way for the sender to transmit emergency data to the other end.

The most common option field is the maximum segment size option, called the MSS. Each end of a connection normally specifies this option on the first segment exchanged (the one with the SYN flag set to establish the connection). It specifies the maximum sized segment that the sender wants to receive.

2.4.3.2. Sequence Numbers and Acknowledge Numbers

Two of the most important fields in the TCP segment header are the sequence number field and the acknowledgment number field. These fields are a critical part of TCP's reliable data transfer service. But before discussing how these fields are used to provide reliable data transfer, let us first explain what exactly TCP puts in these fields.

TCP views data as an unstructured, but ordered, stream of bytes. TCP's use of sequence numbers reflects this view in that sequence numbers are over the stream of transmitted bytes and not over the series of transmitted segments. The sequence number for a segment is the byte-stream number of the first byte in the segment. Let's look at an example. Suppose that a process in host A wants to send a stream of data to a process in host B over a TCP connection. The TCP in host A will implicitly number each byte in the data stream. Suppose that the data stream consists of a file consisting of 500,000 bytes, that the MSS is 1,000 bytes, and that the first byte of the data stream is numbered zero. As shown in Figure 16, TCP constructs 500 segments out of the data stream. The first segment gets assigned sequence number 0, the second segment gets assigned sequence number 1000, and the third segment gets assigned sequence number 2000, and so on. Each sequence number is inserted in the sequence number field in the header of the appropriate TCP segment.

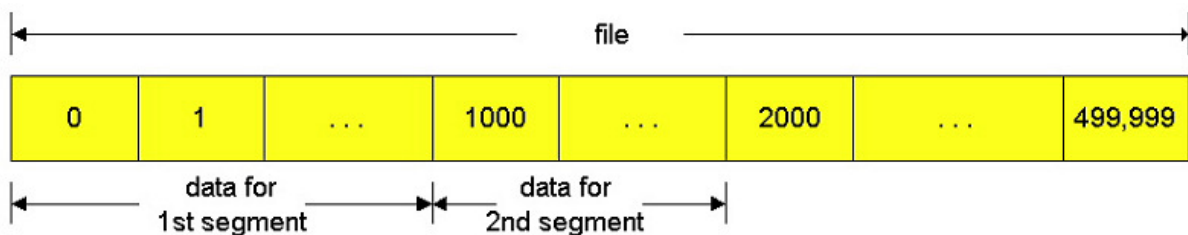


Figure 17. Dividing files into TCP Segments

Recall that TCP is full duplex, so that host A may be receiving data from host B while it sends data to host B (as part of the same TCP connection). Each of the segments that arrive from host B has a sequence number for the data flowing from B to A. The acknowledgment number that host A puts in its segment is the sequence number of the next byte host A is expecting from host B. It is good to look at a few examples to understand what is going on here. Suppose that host A has received all bytes numbered 0 through 535 from B and suppose that it is about to send a segment to host B. In other words, host A is waiting for byte 536 and all the subsequent bytes in host B's data stream. So host A puts 536 in the acknowledgment number field of the segment it sends to B.

As another example, suppose that host A has received one segment from host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000. For some reason host A has not yet received bytes 536 through 899. In this example, host A is still waiting for byte 536 (and beyond) in order to recreate B's data stream. Thus, A's next segment to B will contain 536 in the acknowledgment number field. Because TCP only acknowledges bytes up to the first missing byte in the stream, TCP is said to provide cumulative acknowledgements.

2.4.3.3. Principles of Reliable Data Transfer

Recall that the Internet's network-layer service (IP service) is unreliable. IP does not guarantee datagram delivery, does not guarantee in-order delivery of datagrams and does not guarantee the integrity of the data in the datagram. With IP service, datagrams can overflow router buffers and never reach their destination, datagrams can arrive out of order, and bits in the datagram can get corrupted. Because transport-layer segments are carried across the network by IP datagrams, transport-layer segments can suffer from these problems as well.

TCP creates a reliable data transfer service on top of IP's unreliable best-effort service. TCP's reliable data transfer ensures that the data stream that a process reads out of its TCP receive buffer is uncorrupted, without gaps, without duplication and in sequence; that is, the byte stream is exactly the same byte stream that was sent by the end system on the other side of the connection.

Reliable Data Transfer is usually accomplished using a combination of two fundamental mechanisms: acknowledgments and timeouts. An acknowledgment (ACK for short) is a small control frame that a protocol sends back to its peer saying that it has received an earlier frame. By control frame we mean a header without any data, although a protocol can piggyback an ACK on a data frame it just happens to be sending in the opposite direction. The receipt of an acknowledgment indicates to the sender of the original frame that its frame was successfully delivered. If the sender does not receive an acknowledgment after a reasonable amount of time, then it retransmits the original frame. This action of waiting a reasonable amount of time is called a *timeout*.

The general strategy of using acknowledgments and timeouts to implement reliable delivery is sometimes called *automatic repeat request* (normally abbreviated ARQ). This section describes three different ARQ algorithms using generic language; that is, we do not give detailed information about a particular protocol's header fields.

2.4.3.4. ARQ Algorithms

Stop-and-Wait

The simplest ARQ scheme is the stop-and-wait algorithm. The idea of stop-and-wait is straightforward: After transmitting one frame, the sender waits for an acknowledgement before transmitting the next frame. If the acknowledgment does not arrive after a certain period of time, the sender times out and retransmits the original frame.

Figure 17 illustrates four different scenarios that result from this basic algorithm. The figure shows a timeline, a common way to depict a protocol's behavior. The sending side is represented on the left, the receiving side is depicted on the right, and time flows from top to bottom. Figure 18(a) shows the situation in which the ACK is received before the timer expires, (b) and (c) show the situation in which the original frame and

the ACK, respectively, are lost, and (d) shows the situation in which the timeout fires too soon. Recall that by “lost” we mean that the frame was corrupted while in transit, that this corruption was detected by an error code on the receiver, and that the frame was subsequently discarded.

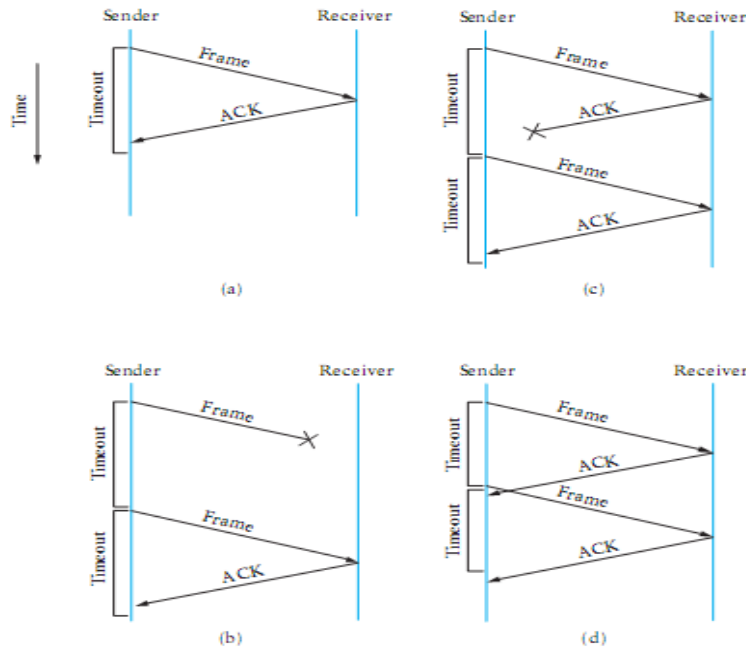


Figure 18. Timeline showing, four different scenarios for stop-and-wait algorithm. (a) ACK is received before timer expires; (b) the original frame is lost; (c) the ACK is lost; (d) the timeout fires too soon.

There is one important subtlety in the stop-and-wait algorithm. Suppose the sender sends a frame and the receiver acknowledges it, but the acknowledgment is either lost or delayed in arriving. This situation is illustrated in timelines (c) and (d) of Figure 19. In both cases, the sender times out and retransmits the original frame, but the receiver will think that it is the next frame, since it correctly received and acknowledged the first frame. This has the potential to cause duplicate copies of a frame to be delivered. To address this problem, the header for a stop-and-wait protocol usually includes a 1-bit sequence number—that is, the sequence number can take on the values 0 and 1—and the sequence numbers used for each frame alternate, as illustrated in Figure 19. Thus, when the sender retransmits frame 0, the receiver can determine that it is seeing a second copy of frame 0 rather than the first copy of frame 1 and therefore can ignore it (the receiver still acknowledges it, in case the first ACK was lost).

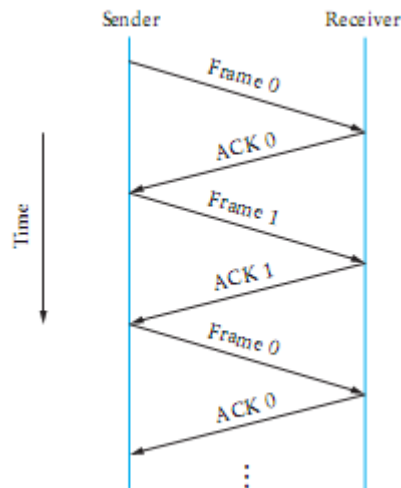


Figure 19. Timeline for stop-and-wait with 1 bit sequence number

The main shortcoming of the stop-and-wait algorithm is that it allows the sender to have only one outstanding frame on the link at a time, and this may be far below the link's capacity. Consider, for example, a 1.5-Mbps link with a 45-ms round-trip time. This link has a delay \times bandwidth product of 67.5 Kb, or approximately 8 KB. Since the sender can send only one frame per RTT (Round Trip Time), and assuming a frame size of 1 KB, this implies a maximum sending rate of

$$\begin{aligned} & \text{BitsPerFrame} \div \text{TimePerFrame} \\ & = 1024 \times 8 \div 0.045 \\ & = 182 \text{ Kbps} \end{aligned}$$

or about one-eighth of the link's capacity. To use the link fully, then, the sender should be able to transmit up to eight frames before having to wait for an acknowledgment.

Sliding Window

Consider again the scenario in which the link has a delay \times bandwidth product of 8 KB and frames are of 1 KB size. We would like the sender to be ready to transmit the ninth frame at pretty much the same moment that ACK for the first arrives. The Algorithm that allows us to do this is called sliding window, and an illustrative timeline is given in Figure20.

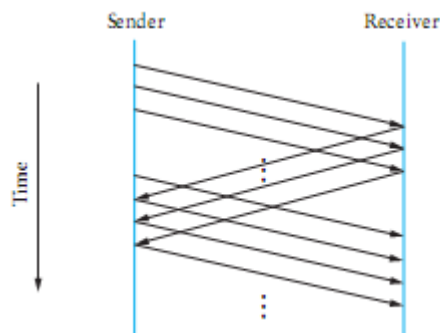


Figure 20. Timeline for the sliding window algorithm.

The sliding window algorithm works as follows. First, the sender assigns a sequence number, denoted SeqNum, to each frame. For now, let's ignore the fact that SeqNum is implemented by a finite-size header field and instead assume that it can grow infinitely large. The sender maintains three variables: The send window size, denoted SWS, gives the upper limit on the number of outstanding (unacknowledged) frames that the sender can transmit; LAR denotes the sequence number of the last acknowledgment received; and LFS denotes the sequence number of the last frame sent. The sender also maintains the following invariant:

$$\text{LFS} - \text{LAR} \leq \text{SWS}$$

This situation is illustrated in Figure 21.

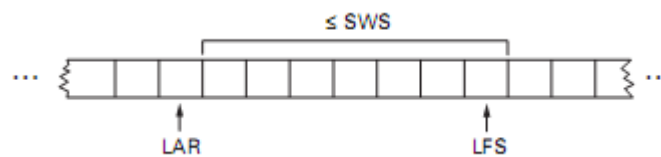


Figure 21. Sliding Window on sender

When an acknowledgment arrives, the sender moves LAR to the right, thereby allowing the sender to transmit another frame. Also, the sender associates a timer with each frame it transmits, and it retransmits the frame should the timer expire before an ACK is received. Notice that the sender has to be willing to buffer up to SWS frames since it must be prepared to retransmit them until they are acknowledged.

The receiver maintains the following three variables: The receiver window size, denoted RWS, gives the upper limit on the number of out-of-order frames that the receiver is willing to accept; LAF denotes the sequence number of the largest acceptable frame; and LFR denotes the sequence number of the last frame received. The receiver also maintains the following invariant:

$$\text{LAF} - \text{LFR} \leq \text{RWS}$$

This situation is illustrated in Figure 22.

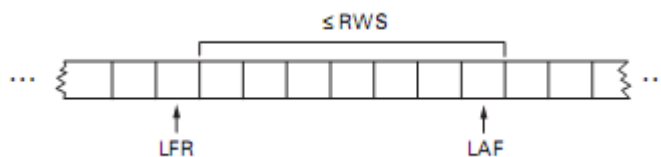


Figure 22. Sliding window on receiver

When a frame with sequence number SeqNum arrives, the receiver takes the following action. If $\text{SeqNum} \leq \text{LFR}$ or $\text{SeqNum} > \text{LAF}$, then the frame is outside the receiver's window and it is discarded. If $\text{LFR} < \text{SeqNum} \leq \text{LAF}$, then the frame is within the receiver's window and it is accepted. Now the receiver needs to decide whether or not to send an ACK. Let SeqNumToAck denote the largest sequence number not yet acknowledged, such that all frames with sequence numbers less than or equal to SeqNumToAck have been received. The receiver acknowledges the receipt of SeqNumToAck, even if higher-numbered packets have been received. This



acknowledgment is said to be cumulative. It then sets $LFR = SeqNumToAck$ and adjusts $LAF = LFR + RWS$.

For example, suppose $LFR = 5$ (i.e., the last ACK the receiver sent was for sequence number 5), and $RWS = 4$. This implies that $LAF = 9$. Should frames 7 and 8 arrive, they will be buffered because they are within the receiver's window. However no ACK needs to be sent since frame 6 is yet to arrive. Frames 7 and 8 are said to have arrived out of order (Technically, the receiver could resend an ACK for frame 5 when frames 7 and 8 arrive). Should frame 6 then arrive—perhaps it is late because it was lost the first time and had to be retransmitted, or perhaps it was simply delayed—the receiver acknowledges frame 8, bumps LFR to 8, and sets LAF to 12. If frame 6 was in fact lost, then a timeout will have occurred at the sender, causing it to retransmit frame 6.

We observe that when a timeout occurs, the amount of data in transit decreases, since the sender is unable to advance its window until frame 6 is acknowledged. This means that when packet losses occur, this scheme is no longer keeping the pipe full. The longer it takes to notice that a packet loss has occurred, the more severe this problem becomes.

Notice that in this example, the receiver could have sent a negative acknowledgment (NAK) for frame 6 as soon as frame 7 arrived. However, this is unnecessary since the sender's timeout mechanism is sufficient to catch this situation, and sending NAKs adds additional complexity to the receiver. Also, as we mentioned, it would have been legitimate to send additional acknowledgments of frame 5 when frames 7 and 8 arrived; in some cases, a sender can use duplicate ACKs as a clue that a frame was lost. Both approaches help to improve performance by allowing early detection of packet losses.

Yet another variation on this scheme would be to use selective acknowledgments. That is, the receiver could acknowledge exactly those frames it has received, rather than just the highest-numbered frame received in order. So, in the above example, the receiver could acknowledge the receipt of frames 7 and 8. Giving more information to the sender makes it potentially easier for the sender to keep the pipe full, but adds complexity to the implementation.

The sending window size is selected according to how many frames we want to have outstanding on the link at a given time; SWS is easy to compute for a given delay bandwidth product. On the other hand, the receiver can set RWS to whatever it wants. Two common settings are $RWS = 1$, which implies that the receiver will not buffer any frames that arrive out of order, and $RWS = SWS$, which implies that the receiver can buffer any of the frames the sender transmits. It makes no sense to set $RWS > SWS$ since it's impossible for more than SWS frames to arrive out of order.

Connect Establishment and Termination

TCP connection begins with a client (caller) doing an active open to a server (callee). Assuming that the server had earlier done a passive open, the two sides engage in an exchange of messages to establish the connection. Only after this connection

establishment phase is over do the two sides begin sending data. Likewise, as soon as a participant has finished sending data, it closes one direction of the connection, which causes TCP to initiate a round of connection termination messages. Notice that while connection setup is an asymmetric activity (one side does a passive open and the other side does an active open), connection teardown is symmetric (each side has to close the connection independently). Therefore, it is possible for one side to have done a close, meaning that it can no longer send data, but for the other side to keep the other half of the bidirectional connection open and to continue sending data.

Three-Way Handshake

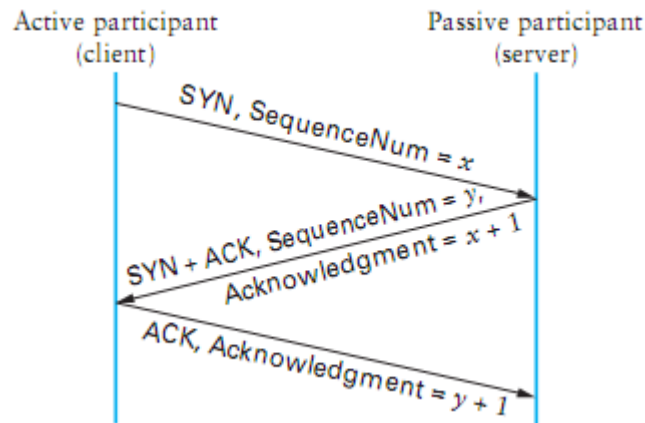


Figure 23. Time Line for Three-way handshake algorithm.

The algorithm used by TCP to establish and terminate a connection is called a three-way handshake. We first describe the basic algorithm and then show how it is used by TCP. The three-way handshake involves the exchange of three messages between the client and the server, as illustrated by the timeline given in Figure 23 above.

The idea is that two parties want to agree on a set of parameters, which, in the case of opening a TCP connection, are the starting sequence numbers the two sides plan to use for their respective byte streams. In general, the parameters might be any facts that each side wants the other to know about. First, the client (the active participant) sends a segment to the server (the passive participant) stating the initial sequence number it plans to use (Flags = SYN, SequenceNum = x). The server then responds with a single segment that both acknowledges the client's sequence number (Flags = ACK, Ack = $x + 1$) and states its own beginning sequence number (Flags = SYN, SequenceNum = y). That is, both the SYN and ACK bits are set in the Flags field of this second message. Finally, the client responds with a third segment that acknowledges the server's sequence number (Flags = ACK, Ack = $y + 1$). The reason that each side acknowledges a sequence number that is one larger than the one sent is that the Acknowledgment field actually identifies the "next sequence number expected," thereby implicitly acknowledging all earlier sequence numbers. Although not shown in this timeline, a timer is scheduled for each of the first two segments, and if the expected response is not received, the segment is retransmitted.

You may be asking yourself why the client and server have to exchange starting sequence numbers with each other at connection setup time. It would be simpler if



each side simply started at some “well-known” sequence number, such as 0. In fact, the TCP specification requires that each side of a connection select an initial starting sequence number at random. The reason for this is to protect against two incarnations of the same connection reusing the same sequence numbers too soon, that is, while there is still a chance that a segment from an earlier incarnation of a connection might interfere with a later incarnation of the connection.

2.5. Network Layer

2.5.1. Internet Protocol (IPv4)

Internet's network layer does not provide a virtual-circuit service, but instead a connectionless datagram service. When the network layer at the sending host receives a segment from the transport layer, it encapsulates the segment within an IP datagram, writes the destination address of the host (as well as other fields) on the datagram, and drops the datagram into the network. This process is similar to a person writing a letter, inserting the letter in an envelope, writing the destination address on the envelope, and dropping the envelope into a mailbox. Neither the Internet's network layer nor the postal service make any kind of preliminary contact with the destination before moving its "parcel" to the destination. Furthermore, the network layer service is a best effort service. It does not guarantee that the datagram will arrive within a certain time, it does not guarantee that a series of datagrams will arrive in the same order sent; in fact, it does not even guarantee that the datagram will ever arrive at its destination.

The network layer for a datagram network, such as the Internet, has two major components. First, it has a network protocol component, which defines network-layer addressing, the fields in the datagram (i.e., the network layer PDU), and how the end systems and routers act on these fields. The network protocol in the Internet is called the *Internet Protocol*, or more commonly, the IP Protocol. There are currently two versions of the IP protocol in use today. In this section we examine the more widespread version, namely, Internet Protocol version 4, which is specified in [RFC 791] and which is more commonly known as IPv4. IPv6, is expected to slowly replace IPv4 in the upcoming years. For example; World IPv6 Day will be held on June 8, 2011. This will be the first global test of IPv6. Google, Facebook, Yahoo and others will participate. The second major component of the network layer is the path determination component, which determines the route a datagram follows from origin to destination. We study the path determination component in the next section.

2.5.1.1. IP Addressing

Before discussing IP addressing, we need to say a few words about hosts and routers. A host (also called an end system) has one link into the network. When IP in the host wants to send a datagram, it passes the datagram to its link. The boundary between the host and the link is called the interface. A router is fundamentally different from a host in that it has two or more links that connect to it. When a router forwards a datagram, it forwards the datagram over one of its links. The boundary between the router and any one of its links is also called an interface. Thus, a router has multiple interfaces, one for each of its links. Because every interface (for a host or router) is

capable of sending and receiving IP datagrams, IP requires each interface to have an IP address.

Each IP address is 32 bits long (equivalently, four bytes) long. IP addresses are typically written in so-called "dot-decimal notation", whereby each byte of the address is written in its decimal form and is separated by a period. For example, a typical IP address would be 193.32.216.9. The 193 is the decimal equivalent for the first 8 bits of the address; the 32 is the decimal equivalent for the second 8 bits of the address, etc. Thus, the address 193.32.216.9 in binary notation is:

11000001 00100000 11011000 00001001

(A space as been added between the bytes for visual purposes) Because each IP address is 32 bits long, there are 232 possible IP addresses.

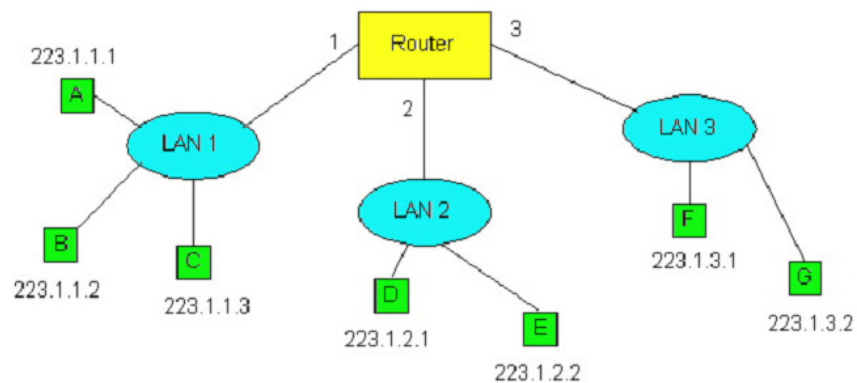


Figure 24. LANs are networks in IP jargon.

Figure 24 provides an example of IP addressing and interfaces. In this figure there is one router which interconnects three LANs. In the jargon of IP, each of these LANs is called an IP network or more simply a "network". There are several things to observe from this diagram. First, the router has three interfaces, labeled 1, 2 and 3. Each of the router interfaces has its own IP address, which is shown in Table 2; each host also has its own interface and IP address. Second, all of the interfaces attached to LAN 1, including a router interface, have an IP address of the form 223.1.1.xxx. Similarly, all the interfaces attached to LAN 2 and LAN 3 have IP addresses of the form 223.1.2.xxx and 233.1.3.xxx, respectively. In other words, each address has two parts: the first part (the first three bytes in this example) that specifies the network; and the second part (the last byte in this example) that addresses a specific host on the network.

Table 2. IP addresses for router interfaces.

Router Interface	IP Address
1	223.1.1.4
2	223.1.2.9
3	223.1.3.27

The IP definition of a "network" is not restricted to a LAN. To get some insight here, let us now take a look at another example.

Figure 25 shows several LANs interconnected with three routers. All of the interfaces attached to LAN 1, including the router R1 interface that is attached to LAN 1, have an IP address of the form 223.1.1.xxx. Similarly, all the interfaces attached to LAN 2 and to LAN 3 have the form 223.1.2.xxx and 223.1.3.xxx, respectively. Each of the three LANs again constitutes their own network (i.e., IP network). But note that there are three additional "networks" in this example: one network for the interfaces that connect Router 1 to Router 2; another network for the interfaces that connect Router 2 to Router 3; and a third network for the interfaces that connect Router 3 to Router 1.

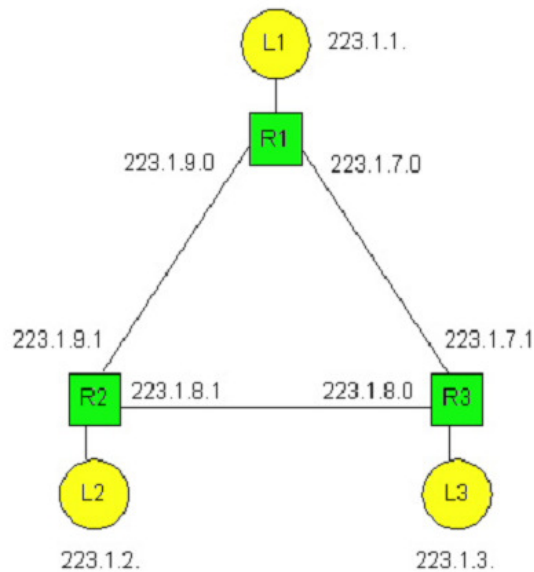


Figure 25. An interconnected system consisting of six networks.

For a general interconnected system of routers and hosts (such as the Internet), we use the following recipe to define the "networks" in the system. We first detach each router interface from its router and each host interface from its host. This creates "islands" of isolated networks, with "interfaces" terminating all the leaves of the isolated networks. We then call each of these isolated networks a network. Indeed, if we apply this procedure to the interconnected system, we get six islands or "networks". The current Internet consists of millions of networks. (In the next chapter we will consider bridges. We mention here that when applying this recipe, we do not detach interfaces from bridges. Thus each bridge lies within the interior of some network.)

Now that we have defined a network, we are ready to discuss IP addressing in more detail. IP addresses are globally unique, that is, no two interfaces in the world have the same IP address. Figure 26 shows the four possible formats of an IP address. (A fifth address, beginning with 11110, is reserved for future use.) In general, each interface (for a host or router) belongs to a network; the network part of the address identifies the network to which the interface belongs. The host part identifies the specific interface within the network. (We would prefer to use the terminology "interface part of the address" rather than "host part of the address" because the IP address is really for an interface and not for a host; but the terminology "host part" is commonly used in practice.) For a class A address, the first 8 bits identify the network, and the last 24 bits

identify the interface within that network. Thus with a class A we can have up to 27 networks (the first of the eight bits is fixed as 0) and 224 interfaces. The class B address space allows for 214 networks, with up to 216 interfaces within each network. A class C address uses 21 bits to identify the network and leaves only 8 bits for the interface identifier. Class D addresses are reserved for so-called multicast addresses. These addresses do not identify a specific interface but rather provide a mechanism through which multiple hosts can receive a copy of each single packet sent by a sender.

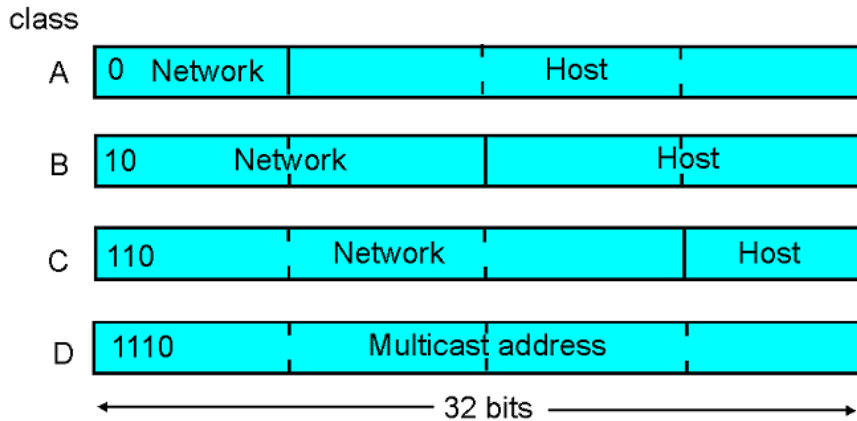


Figure 26. IP Classes

2.5.1.2. Assigning Addresses

Having introduced IP addressing, one question that immediately comes to mind is how does a host get its own IP address? We have just learned that an IP address has two parts, a network part and a host part. The host part of the address can be assigned in several different ways, including:

- **Manual configuration:** The IP address is configured into the host (typically in a file) by the system administrator.
- **Dynamic Host Configuration Protocol (DHCP):** [RFC 2131]. DHCP is an extension of the BOOTP [RFC 1542] protocol, and is sometimes referred to as Plug and Play. With DHCP, a DHCP server in a network (e.g., in a LAN) receives DHCP requests from a client and in the case of dynamic address allocation, allocates an IP address back to the requesting client. DHCP is used extensively in LANs and in residential Internet access.

The network part of the address is the same for all the hosts in the network. To obtain the network part of the address for a network, the network administrator might first contact the network's ISP, which would provide addresses from a larger block of addressees that have already been allocated to the ISP. But how does an ISP get a block of addresses? IP addresses are managed under the authority of the Internet Assigned Numbers Authority (IANA), under the guidelines set forth in [RFC 2050]. The actual assignment of addresses is now managed by regional Internet registries. As of mid-1998, there are three such regional registries: the American Registry for Internet Number (ARIN, which handles registrations for North and South America, as well as parts of Africa. ARIN has recently taken over a number of the functions previously

provided by Network Solutions), the Reseaux IP Europeans (RIPE, which covers Europe and nearby countries), and the Asia Pacific Network Information Center (APNIC).

Before leaving our discussion of addressing, we want to mention that mobile hosts may change the network to which they are attached, either dynamically while in motion or on a longer time scale. Because routing is to a network first, and then to a host within the network, this means that the mobile host's IP address must change when the host changes networks. Techniques for handling such issues are now under development within the IETF and the research community [RFC2002] [RFC2131].

2.5.1.3. Transporting a Datagram from Source to Destination

Now that we have defined interfaces and networks, and that we have a basic understanding of IP addressing, we take a step back and discuss how IP transports a datagram from source to destination. To this end, a high level view of an IP datagram is shown in Figure 27. Note that every IP datagram has a destination address field and a source address field. The source host fills the source address field with its own 32-bit IP address and fills the destination address field with the 32-bit IP address of the host to which it wants to send the datagram. Note that these actions are analogous to what you do when you send a letter: on the envelope of the letter, you provide a destination address and a return (source) address. The data field of the datagram is typically filled with a TCP or UDP segment. We will discuss the remaining IP datagram fields a little later in this section.

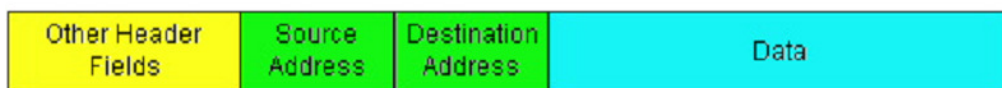


Figure 27. The key fields in an IP Datagram

Once the source host creates the IP datagram, how does the network layer transport the datagram from the source host to the destination host? First suppose host A wants to send an IP datagram to host B. The datagram is transported from host A to host B as follows. IP in host A first extracts the network portion of the address, 223.1.1. , and scans its routing table, which is shown below in Table 3. In this table, the "number of hops to destination" is defined to be the number of networks that need to be traversed, including the destination network. Scanning the table, host A finds a match in the first row, and observes that the number of hops to the destination is 1. This indicates to host A that the destination host is on the same network. Host A then passes the IP datagram to the link layer protocol and indicates to the link layer protocol that the destination is on the same LAN. The link layer protocol then has the responsibility of transporting the datagram to host B.

Table 3. Routing Table in Host A

destination network	next router	number of hops to destination
223.1.1.	-	1
223.1.2.	223.1.1.4	2
223.1.3.	223.1.1.4	2

Now consider the more interesting case of host A sending an IP datagram to host E, which has IP address 223.1.2.2 and is on a different LAN. Host A again scans its routing table, but now finds a match in the second row. Because the number of hops to the destination is 2, host A knows that the destination is on another network. The routing table also tells host A that in order to get the datagram to host E, host A should first send the datagram to router address 223.1.1.4. IP in host A then passes the datagram down to the link layer, and indicates to the link layer that it should first send the datagram to IP address 223.1.1.4. The link layer then transports the datagram to the router interface 1. The datagram is now in the router, and it is the job of the router to move the datagram towards the datagram's ultimate destination. The router extracts the network portion of the destination address of the IP datagram, namely 223.1.2. , and scans its routing table, which is shown in Table 4. The router finds a match in the second row of the table. The table tells the router that the datagram should be forwarded on router interface 2; also the number of hops to the destination is 1, which indicates to the router that the destination host is on the LAN directly attached to interface 2. The router moves the datagram to interface 2. Once the datagram is at interface 2, the router passes the datagram to link layer protocol and indicates to the link layer protocol that the destination host is on the same LAN. The link layer protocol has the job of transporting the datagram from the router interface 2 to host E, both of which are attached to the same LAN.

Table 4. Routing Table in Router

destination network	next router	number of hops to destination	interface
223.1.1.	-	1	1
223.1.2.	-	1	2
223.1.3.	-	1	3

In Table 4, note that the entries in the "next router" column are all empty. This is because all of the networks (223.1.1. , 223.1.2. , and 223.1.3.) are each directly attached to the router, that is, there is no need to go through an intermediate router to get to the destination host. However, if host A and host E were separated by two routers, then within the routing table of the first router along the path from A to B, the appropriate row would indicate 2 hops to the destination and would specify the IP address of the second router along the path. The first router would then forward the datagram to the second router, using the link layer protocol that connects the two

routers. The second router then forwards the datagram to the destination host, using the link layer protocol that connects the second router to the destination host.

Routing a datagram in the Internet is similar to a person driving a car and asking gas station attendants at each intersection along the way how to get to the ultimate destination. It should now be clear why this is an appropriate analogy for routing in the Internet. As a datagram travels from source to destination, it visits a series of routers. At each router in the series, it stops and asks the router how to get to its ultimate destination. Unless the router is on the same LAN as the ultimate destination, the routing table essentially says to the datagram: "I don't know exactly how to get to the ultimate destination, but I do know that the ultimate destination is in the direction of the link (analogous to a road) connected to interface 3." The datagram then sets out on the link connected to interface 3, arrives at a new router, and again asks for new directions.

From this discussion we see that the routing tables in the routers play a central role in routing datagrams through the Internet. But how are these routing tables configured and maintained for large networks with multiple paths between sources and destinations (such as in the Internet)? Clearly, these routing tables should be configured so that the datagrams follow good (if not optimal) routes from source to destination. As you probably guessed, routing algorithms have the job of configuring and maintaining the routing tables. Furthermore, the Internet is partitioned into autonomous systems (ASs): intra-AS routing algorithms independently configure the routing tables within the autonomous systems; inter-AS routing algorithms have the job of configuring routing tables so that datagrams can pass through multiple autonomous systems. We will discuss the Internet's intra-AS and inter-AS routing algorithms in Section 4.5. But before moving on to routing algorithms, we cover three more important topics for the IP protocol, namely, the datagram format, datagram fragmentation, and the Internet Control Message Protocol (ICMP).

2.5.1.4. Datagram Format

The IPv4 Datagram format is shown in Figure 28.

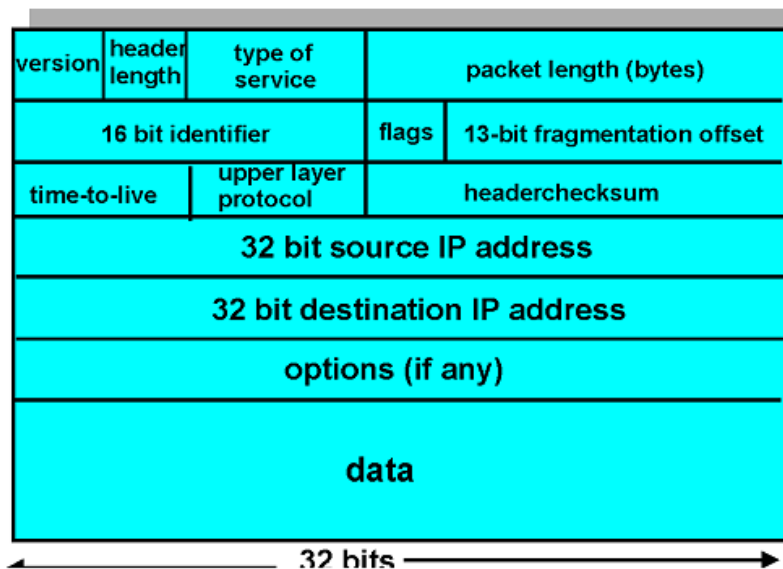


Figure 28. IPv4 Datagram Format



- **Version Number:** These 4 bits specify the IP protocol version of the datagram. By looking at the version number, the router can then determine how to interpret the remainder of the IP datagram. Different versions of IP use different datagram formats. The datagram format for the "current" version of IP, IPv4, is shown in Figure 27.
- **Header Length:** Because an IPv4 datagram can contain a variable number of options (which are included in the IPv4 datagram header) these 4 bits are needed to determine where in the IP datagram the data actually begins. Most IP datagrams do not contain options so the typical IP datagram has a 20 byte header.
- **TOS:** The type of service (TOS) bits were included in the IPv4 header to allow different "types" of IP datagrams to be distinguished from each other, presumably so that they could be handled differently in times of overload. When the network is overloaded, for example, it would be useful to be able to distinguish network control datagrams from datagrams carrying data (e.g., HTTP messages). It would also be useful to distinguish real-time datagrams (e.g., used by an IP telephony application) from non-real-time traffic (e.g., FTP). More recently, one major routing vendor (Cisco) interprets the first three ToS bits as defining differential levels of service that can be provided by the router. The specific level of service to be provided is a policy issue determined by the router's administrator.
- **Datagram Length:** This is the total length of the IP datagram (header plus data) measured in bytes. Since this field is 16 bits long, the theoretical maximum size of the IP datagram is 65,535 bytes. However, datagrams are rarely greater than 1500 bytes, and are often limited in size to 576 bytes.
- **Identifier, Flags, Fragmentation Offset:** These three fields have to do with so-called IP fragmentation, a topic we will consider in depth shortly. Interestingly, the new version of IP, IPv6, simply does not allow for fragmentation.
- **Time-to-live:** The time-to-live (TTL) field is included to insure that datagrams do not circulate forever (due to, for example, a long lived router loop) in the network. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, the datagram must be dropped.
- **Protocol:** This field is only used when an IP datagram reaches its final destination. The value of this field indicates the transport-layer protocol at the destination to which the data portion of this IP datagram will be passed. For example, a value of 6 indicates that the data portion is passed to TCP, while a value of 17 indicates that the data is passed to UDP. For a listing of all possible numbers, see [RFC 1700]. Note that the protocol number in the IP datagram has a role that is fully analogous to the role of the port number field in the transport-layer segment. The protocol number is the "glue" that holds the network and transport layers together, whereas port number is the "glue" that holds the transport and application layers together.
- **Header Checksum:** The header checksum aids a router in detecting bit errors in a received IP datagram. The header checksum is computed by treating each 2 bytes in



the header as a number and summing these numbers using 1's to complement the arithmetic. The 1's complement of this sum, known as the Internet checksum, is stored in the checksum field. A router computes the Internet checksum for each received IP datagram and detects an error condition if the checksum carried in the datagram does not equal the computed checksum. Routers typically discard datagrams for which an error has been detected. Note that the checksum must be recomputed and restored at each router, as the TTL field, and possibly options fields as well, may change. An interesting discussion of fast algorithms for computing the Internet checksum is [1071]. A question often asked at this point is, why does TCP/IP perform error checking at both the transport and network layers? There are many reasons for this. First, routers are not required to perform error checking, so the transport layer cannot count on the network layer to do the job. Second, TCP/UDP and IP do not necessarily have to both belong to the same protocol stack. TCP can, in principle, run over a different protocol (e.g., ATM) and IP can carry data without passing through TCP/UDP (e.g., RIP data).

- **Source and Destination IP Address:** These fields carry the 32 bit IP address of the source and final destination for this IP datagram. The use and importance of the destination address is clear. The source IP address (along with the source and destination port numbers) is used at the destination host to direct the application data in the proper socket.

- **Options:** The optional options fields allow an IP header to be extended. Header options were meant to be used rarely -- hence the decision to save overhead by not including the information in options fields in every datagram header. However, the mere existence of options does complicate matters -- since datagram headers can be of variable length, one can not determine a priori where the data field will start. Also, since some datagrams may require options processing and others may not, the amount of time needed to process an IP datagram can vary greatly. These considerations become particularly important for IP processing in high performance routers and hosts. For these reasons and others, IP options were dropped in the IPv6 header.

- **Data (payload):** Finally, we come to the last, and most important field - the *raison d'être* for the datagram in the first place! In most circumstances, the data field of the IP datagram contains the transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages.

Note that an IP datagram has a total of 20 bytes of header (assuming it has no options). If the IP datagram carries a TCP segment, then each (non-fragmented) datagram carries a total of 40 bytes of header (20 IP bytes and 20 TCP bytes) along with the application-layer data.



2.5.1.5. IP fragmentation and Reassembly

Not all link layer protocols can carry packets of the same size. Some protocols can carry "big" packets whereas other protocols can only carry "little" packets. For example, Ethernet packets can carry no more than 1500 bytes of data, whereas packets for many wide-area links can carry no more than 576 bytes. The maximum amount of data that a link-layer packet can carry is called the *Maximum Transfer Unit* (MTU). Because each IP datagram is encapsulated within the link-layer packet for transport from one router to the next router, the MTU of the link-layer protocol places a hard limit on the length of an IP datagram. Having a hard limit on the size of an IP datagram is not much of a problem. What is a problem is that each of the links along the route between sender and destination can use different link-layer protocols, and each of these protocols can have different MTUs.

To understand the problem better, imagine that you are a router that interconnects several links, each running different link-layer protocols with different MTUs. Suppose you receive an IP datagram from one link, you check your routing table to determine the outgoing link, and this outgoing link has an MTU that is smaller than the length of the IP datagram. Time to panic -- how are you going to squeeze this oversized IP packet into the payload field of the link-layer packet? The solution to this problem is to "fragment" the data in the IP datagram among two or more smaller IP datagrams, and then send these smaller datagrams over the outgoing link. Each of these smaller datagrams is referred to as a *fragment*.

Fragments need to be reassembled before they reach the transport layer at the destination. Indeed, both TCP and UDP are expecting to receive from the network layer complete, un-fragmented segments. The designers of IPv4 felt that reassembling (and possibly re-fragmenting) datagrams in the routers would introduce significant complication into the protocol and put a damper on router performance. (If you were a router, would you want to be reassembling fragments on top of everything else you have to do?) Sticking to end-to-end principle for the Internet, the designers of IPv4 decided to put the job of datagram reassembly in the end systems rather than in the network interior.

When a destination host receives a series of datagrams from the same source, it needs to determine if any of these datagrams are fragments of some "original" bigger datagram. If it does determine that some datagrams are fragments, it must further determine when it has received the last fragment and how the fragments it has received should be pieced back together to form the original datagram. To allow the destination host to perform these reassembly tasks, the designers of IPv4 put identification, flag and fragmentation fields in the IP datagram. When a datagram is created, the sending host stamps the datagram with an identification number as well as a source and destination address. The sending host increments the identification number for each datagram it sends. When a router needs to fragment a datagram, each resulting datagram (i.e., "fragment") is stamped with the source address, destination address and identification number of the original datagram. When the destination receives a series of datagrams from the same sending host, it can examine

the identification numbers of the datagrams to determine which of the datagrams are actually fragments of the same bigger datagram. Because IP is an unreliable service, one or more of the fragments may never arrive at the destination. For this reason, in order for the destination host to be absolutely sure it has received the last fragment of the original datagram, the last fragment has a flag bit set to 0 whereas all the other fragments have this flag bit set to 1. Also, in order for the destination host to determine if a fragment is missing (and also to be able to reassemble the fragments in the proper order), the offset field is used to specify where the fragment fits within the original IP datagram. This bit is set to 1 in all except the last fragment.



Figure 29. IP Fragmentation

Figure 29 illustrates an example. A datagram of 4,000 bytes arrives at a router and this datagram must be forwarded to a link with a MTU of 1500 bytes. This means that the 3,980 data bytes in the original datagram must be allocated to three separate fragments (each of which is also an IP datagram). Suppose that the original datagram is stamped with an identification number of 777. Then the characteristics of the three fragments are as follows:

1st fragment

- 1480 bytes in the data field of the IP datagram.
- identification = 777
- offset = 0 (meaning the data should be inserted beginning at byte 0)
- flag = 1 (meaning there is more)

2nd fragment

- 1480 byte information field
- identification = 777
- offset = 1,480 (meaning the data should be inserted beginning at byte 1,480)
- flag = 1 (meaning there is more)

3rd fragment

- 1020 byte (=3980-1480-1480) information field
- identification = 777
- offset = 2,960 (meaning the data should be inserted beginning at byte 2,960)
- flag = 0 (meaning this is the last fragment)

The payload of the datagram is only passed to the transport layer once the IP layer has fully reconstructed the original IP datagram. If one or more of the fragments does not arrive at the destination, the datagram is "lost" and not passed to the transport layer. If TCP is being used at the transport layer, then TCP will recover from this loss by having the source retransmit the data in the original datagram.



Fragmentation and reassembly puts an additional burden on Internet routers (the additional effort to create fragments out of a datagram) and on the destination hosts (the additional effort to reassemble fragments). For this reason it is desirable to keep fragmentation to a minimum. This is often done by limiting the TCP and UDP segments to a relatively small size, so that the fragmentation of the corresponding datagrams is unlikely. Because all data link protocols supported by IP are supposed to have MTUs of at least 576 bytes, fragmentation can be entirely eliminated by using a MSS of 536 bytes, 20 bytes of TCP segment header and 20 bytes of IP datagram header. This is why most TCP segments for bulk data transfer (such as with HTTP) are 512-536 bytes long. (You may have noticed while surfing the Web that 500 or so bytes of data often arrive at a time.)

2.5.1.6. Internet Control Message Protocol (ICMP)

ICMP is used by hosts, routers, and gateways to communicate network layer information to each other. ICMP is specified in [RFC 792]. The most typical use of ICMP is for error reporting. For example, when running a Telnet, FTP, or HTTP session, you may have encountered an error message such as "Destination network unreachable." This message had its origins in ICMP. At some point, an IP router was unable to find a path to the host specified in your Telnet, FTP or HTTP application. That router created and sent a type-3 ICMP message to your host indicating the error. Your host received the ICMP message and returned the error code to the TCP code that was attempting to connect to the remote host. TCP in turn returned the error code to your application.

ICMP is often considered part of IP, but architecturally lies just above IP, as ICMP messages are carried inside IP packets. That is, ICMP messages are carried as IP payload, just as TCP or UDP packets are carried at IP payload. Similarly, when a host receives an IP packet with ICMP specified as the upper layer protocol, it demultiplexes the packet to ICMP, just as it would demultiplex a packet to TCP or UDP.

ICMP messages have a type and a code field, and also contain the first 8 bytes of the IP packet that caused the IP message to be generated in the first place (so that the sender can determine which packet is sent that caused the error). Selected ICMP messages are shown below in Table 5. Note that ICMP messages are used not only for signaling error conditions. The well-known ping program uses ICMP. Ping sends an ICMP type 8 code 0 message to the specified host. The destination host, seeing the echo request sends back a type 0 code 0 ICMP echo reply. Another interesting ICMP message is the source quench message. This message is seldom used in practice. Its original purpose was to perform congestion control -- to allow a congested router to send an ICMP source quench message to a host to force that host to reduce its transmission rate. TCP has its own congestion control mechanism that operates at the transport layer, without the use of network layer support such as the ICMP source quench message.

Table 5. Selected ICMP Messages

ICMP type	code	description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

The interesting Traceroute program enables you to trace the route from a few given hosts to any host in the world. Traceroute also uses ICMP messages. To determine the names and addresses of the routers between source and destination, Traceroute in the source sends a series of ordinary IP datagrams to the destination. The first of these datagrams has a TTL of 1, the second of 2, the third of 3, etc. The source also starts timers for each of the datagrams. When the nth datagram arrives at the nth router, the nth router observes that the TTL of the datagram has just expired. According to the rules of the IP protocol, the router discards the datagram (because there may be routing loops) and sends an ICMP warning message to the source (type 11 code 0). This warning message includes the name of the router and its IP address. When the ICMP message corresponding to the nth datagram arrives at the source, the source obtains the round-trip time from the timer and the name and IP address from the ICMP message.

2.6. Link Layer and Local Area Networks (LAN)

The purpose of the data link layer is to transfer block soft data without error between two adjacent devices. Adjacent devices are physically connected by a communication channel such as telephone lines, coaxial cables, optical fibres, or satellites. The implication of such a physical link is that the data bits are delivered in exactly the same order in which they are sent. The physical link has no inherent storage capacity; therefore the delay involved is the propagation delay over the link.

Transmission of data over the link would be very simple indeed if no error ever occurred. Unfortunately, this is not so in a real physical link for a number of reasons:



- Natural phenomena such as noises and interference are introduced into the link causing errors in detecting the data.
- There is a propagation delay in the link.
- There is a finite data processing time required by the transmitting and receiving stations.

A data link protocol thus has to be designed to ensure error-free transmission and as efficient as possible data transfer.

The basic service of the link layer is to "move" a datagram from one node to an adjacent node over a single communication link. But the details of the link-layer service depend on the specific link-layer protocol that is employed over the link. Possible services that can be offered by a link-layer protocol include:

- Framing and link access: Almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission onto the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields. (A frame may also include trailer fields; however, we will refer to both header and trailer fields as header fields.) A data link protocol specifies the structure of the frame, as well as a channel access protocol that specifies the rules by which a frame is transmitted onto the link. For point-to-point links that have a single sender on one end of the link and a single receiver at the other end of the link, the link access protocol is simple (or non-existent) - the sender can send a frame whenever the link is idle. The more interesting case is when multiple nodes share a single broadcast link - the so-called multiple access problems. Here, the channel access protocol serves to coordinate the frame transmissions of the many nodes. We'll see that frame headers also often include fields for a node's so-called physical address, which is completely distinct from the node's network layer (e.g., IP) address.
- Reliable delivery: If a link-layer protocol provides reliable-delivery service, then it guarantees to move each network-layer datagram across the link without error. Recall that transport-layer protocols (such as TCP) may also provide a reliable-delivery service. Similar to a transport-layer reliable-delivery service, a link-layer reliable-delivery service is achieved with acknowledgments and retransmissions. A link-layer reliable-delivery service is often used for links that are prone to high error rates, such as a wireless link, with the goal of correcting an error locally, on the link at which the error occurs, rather than forcing an end-to-end retransmission of the data by transport- or application-layer protocol. However, link-layer reliable delivery is often considered to be an unnecessary overhead for low bit-error links, including fiber, coax and many twisted-pair copper links. For this reason, many of the most popular link-layer protocols do not provide a reliable-delivery service.
- Flow control: The nodes on each side of a link have a limited amount of packet buffering capacity. This is a potential problem, as a receiving node may receive frames at a rate faster than it can process the frames (over some time interval). Without flow control, the receiver's buffer can overflow and frames can get lost.

Similar to the transport layer, a link-layer protocol can provide flow control in order to prevent the sending node on one side of a link from overwhelming the receiving node on the other side of the link.

- Error detection: A node's receiver can incorrectly decide that a bit in a frame was a zero when it was transmitted as a one (and vice versa). These errors are introduced by signal attenuation and electromagnetic noise. Because there is no need to forward a datagram that has an error, many link-layer protocols provide a mechanism for a node to detect the presence of one or more errors. This is done by having the transmitting node set error detection bits in the frame, and having the receiving node perform an error check. Error detection is a very common service among link-layer protocols. The transport layer and network layers in the Internet also provide a limited form of error detection. Error detection in the link layer is usually more sophisticated and implemented in hardware.
- Error correction: Error *correction* is similar to error *detection*, except that a receiver can not only detect whether errors have been introduced in the frame but can also determine exactly where in the frame the errors have occurred (and hence correct these errors). Some protocols (such as ATM) provide link-layer error correction for the packet header rather than for the entire packet.
- Half-Duplex and Full-Duplex: With full-duplex transmission, both nodes at the ends of a link may transmit packets at the same time. With half-duplex transmission, a node cannot both transmit and receive at the same time

As noted above, many of the services provided by the link layer have strong parallels with services provided at the transport layer. For example, both the link layer and the transport layer can provide reliable delivery. Although the mechanisms used to provide reliable delivery in the two layers are similar, the two reliable delivery services are not the same. A transport protocol provides reliable delivery between two processes on an end-to-end basis; a reliable link-layer protocol provides reliable-delivery service between two nodes connected by a single link. Similarly, both link-layer and transport-layer protocols can provide flow control and error detection; again, flow control in a transport-layer protocol is provided on an end-to-end basis, whereas in a link-layer protocol it is provided on a node-to-adjacent-node basis.

For a given communication link, the link-layer protocol is for the most part implemented in a pair of adapters. An adapter is a board (or a PCMCIA card) that typically contains RAM, DSP chips, a host bus interface and a link interface. Adapters are also commonly known as network interface cards or NICs. As shown in Figure 30, the network layer in the transmitting node (i.e., a host or router) passes a network-layer datagram to the adapter that handles the sending side of the communication link. The adapter encapsulates the datagram in a frame and then transmits the frame into the communication link. At the other end, the receiving adapter receives the entire frame, extracts the network-layer datagram, and passes it to the network layer. If the link-layer protocol provides error detection, then it is the sending adapter that sets the error detection bits and it is the receiving adapter that performs the error checking. If the link-

layer protocol provides reliable delivery, then the mechanisms for reliable delivery (e.g., sequence numbers, timers and acknowledgments) are entirely implemented in the adapters. If the link-layer protocol provides random access, then the random access protocol is entirely implemented in the adapters.

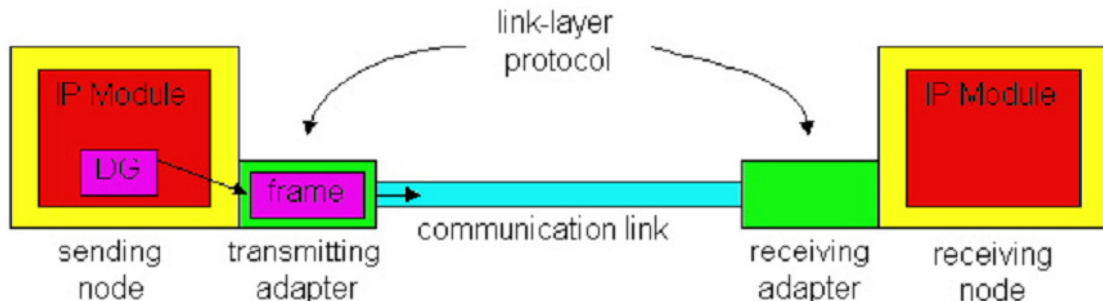


Figure 30. The link-layer protocol for a communication link is implemented in the adapters at the two ends of the link. DG abbreviates "datagram".

A computer in itself, an adapter is a semi-autonomous unit. For example, an adapter can receive a frame, determine if a frame is in error and discard the frame without notifying its "parent" node. An adapter that receives a frame only interrupts its parent node when it wants to pass a network-layer datagram up the protocol stack. Similarly, when a node passes a datagram down the protocol stack to an adapter, the node fully delegates to the adapter the task of transmitting the datagram across that link. On the other hand, an adapter is not a completely autonomous unit. Although we have shown the adapter as a separate "box" in Figure 31, the adapter is typically housed in the same physical box as rest of the node, shares power and busses with the rest of the node, and is ultimately under the control of the node.

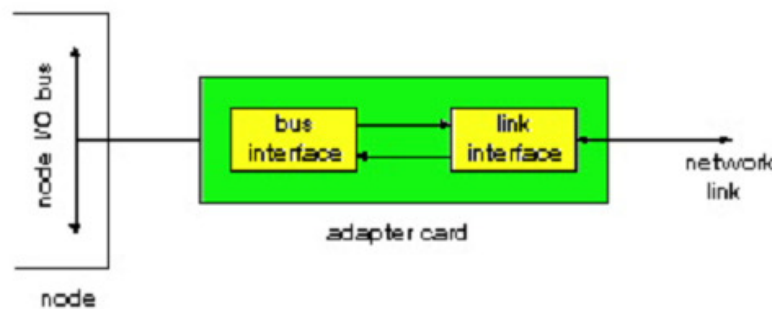


Figure 31. The Adapter is a semi-autonomous unit

As shown in Figure 31, the main components of an adapter are the bus interface and the link interface. The bus interface is responsible for communicating with the adapter's parent node. It sends to and receives from the parent node network-layer datagrams and control information. The link interface is responsible for implementing the link-layer protocol. In addition to framing and de-framing datagrams, it may provide error detection, random access and other link-layer functions. It also includes transmit and receive circuitry. For popular link-layer technologies, such as Ethernet, the link interface is implemented by chip set that can be bought on the commodity market. For this reason, Ethernet adapters are incredibly cheap.

Adapter design has become very sophisticated over the years. One of the critical issues in adapter performance has always been whether the adapter can move data in and out of a node at the full line speed, that is, at the transmission rate of the link.

2.6.1. Error-Detection and Correction Techniques

We noted that bit-level error detection and correction - detecting and correcting the corruption of bits in a data-link-layer frame sent from one node to another physically-connected neighboring node - are two services often provided by the data link layer. Error detection and correction services are also often offered at the transport layer. A full treatment of the theory and implementation of this topic is itself the topic of many textbooks (e.g., Schwartz 1980) and our treatment here is necessarily brief. Our goals here are to develop an intuitive feel for the capabilities that error detection and correction techniques provide, and to see how a few simple techniques work and are used in practice in the data link layer.

Figure 32 illustrates the setting for our study. At the sending node, data, D , to be "protected" against bit errors is augmented with error detection and correction bits, EDC. Typically, the data to be protected includes not only the datagram passed down from the network layer for transmission across the link, but also link-level addressing information, sequence numbers, and other fields in the data link frame header. Both D and EDC are sent to the receiving node in a link-level frame. At the receiving node, a sequence of bits, D' and EDC' are received. Note that D' and EDC' may differ from the original D and EDC as a result of in-transit bit flips.

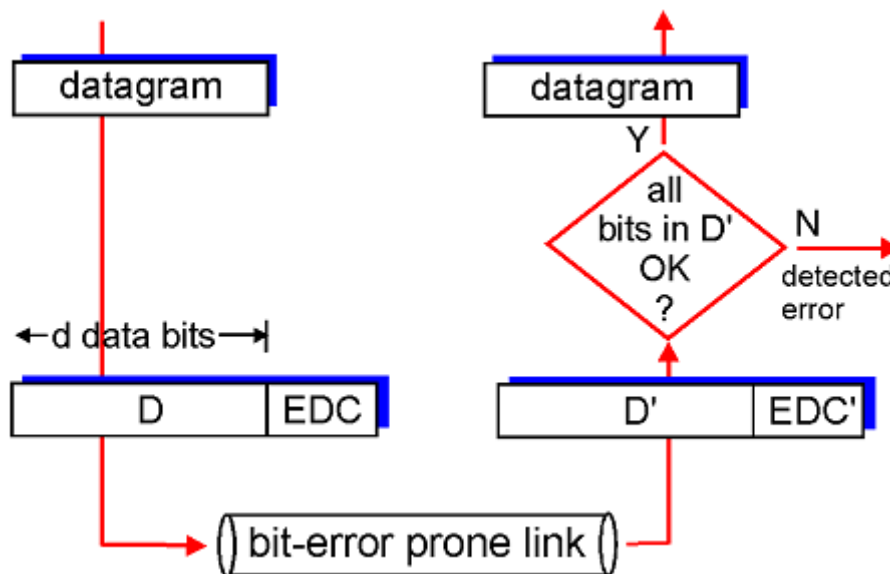


Figure 32. Error Detection and Correction Scenario

The receiver's challenge is to determine whether or not D' is the same as the original D , given that it has only received D' and EDC'. The exact wording of the receiver's decision in Figure 32 (we ask whether an error is detected, not whether an error has occurred!) is important. Error detection and correction techniques allow the receiver to sometimes, but not always, detect that bit errors have occurred. That is, even with the use of error detection bits there will still be a possibility that undetected bit errors will

occur, i.e., that the receiver will be unaware that the received information contains bit errors. As a consequence, the receiver might deliver a corrupted datagram to the network layer, or be unaware that the contents of some other field in the frame's header have been corrupted. We thus want to choose an error detection scheme so that the probability of such occurrences is small. Generally, more sophisticated error detection and correction techniques (i.e., those that have a smaller probability of allowing undetected bit errors) incur a larger overhead - more computation is needed to compute and transmit a larger number of error detection and correction bits.

Let's now examine three techniques for detecting errors in the transmitted data -- parity checks (to illustrate the basic ideas behind error detection and correction), check summing methods (which are more typically employed in the transport layer) and cyclic redundancy checks (which are typically employed in the data link layer).

2.6.1.1. Parity Checks

Perhaps the simplest form of error detection is the use of a single parity bit. Suppose that the information to be sent, D in Figure 32, has d bits. In an even parity scheme, the sender simply includes one additional bit and chooses its value such that the total number of 1's in the $d+1$ bits (the original information plus a parity bit) is even. For odd parity schemes, the parity bit value is chosen such that there are an odd number of 1's. Figure 33 illustrates an even parity scheme, with the single parity bit being stored in a separate field.

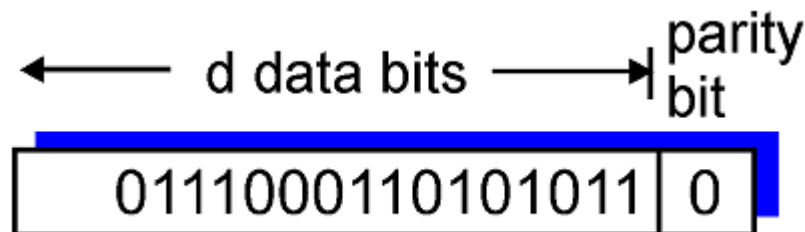


Figure 33. One bit even parity bit.

Receiver operation is also simple with a single parity bit. The receiver need only count the number of 1's in the received $d+1$ bit. If an odd number of 1-valued bits are found with an even parity scheme, the receiver knows that at least one bit error has occurred. More precisely, it knows that some odd numbers of bit errors have occurred.

The ability of the receiver to both detect and correct errors is known as forward error correction (FEC). These techniques are commonly used in audio storage and playback devices such as audio CD's. In a network setting, FEC techniques can be used by themselves, or in conjunction with the ARQ techniques. EC techniques are valuable because they can decrease the number of sender retransmissions required. Perhaps more importantly, they allow for immediate correction of errors at the receiver. This avoids having to wait the round-trip propagation delay needed for the sender to receive a NAK packet and for the retransmitted packet to propagate back to the receiver -- a potentially important advantage for real-time network applications.

2.6.1.2. Checksum Methods

In check summing techniques, the d bits of data in Figure 32 are treated as a sequence of k -bit integers. One simple checksumming method is to simply sum these k -bit integers and use the resulting sum as the error detection bits. The so-called Internet checksum [RFC 1071] is based on this approach -- bytes of data are treated as 16-bit integers and their ones-complement sum forms the Internet checksum. A receiver calculates the checksum it calculates over the received data and checks whether it matches the checksum carried in the received packet. RFC1071 [RFC 1071] discusses the Internet checksum algorithm and its implementation in detail. In the TCP/IP protocols, the Internet checksum is computed over all fields (header and data fields included). In other protocols, e.g., XTP, one checksum is computed over the header, with another checksum computed over the entire packet. McAuley describe improved weighted checksum codes that are suitable for high-speed software implementation and Feldmeier presents fast software implementation techniques for not only weighted checksum codes, but CRC (see below) and other codes as well.

2.6.1.3. Cyclic Redundancy Check (CRC)

An error detection technique used widely in today's computer networks is based on cyclic redundancy check (CRC) codes. CRC codes are also known as polynomial codes, since it is possible to view the bit string to be sent as a polynomial whose coefficients are the 0 and 1 values in the bit string, with operations on the bit string interpreted as polynomial arithmetic.

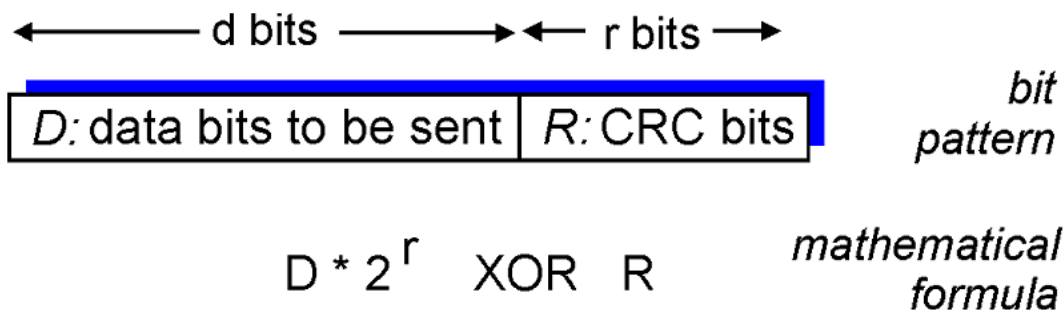


Figure 34. CRC codes.

2.6.2. Link Layer Addressing

In truth, it is not a node that has a LAN address but instead a node's adapter that has a LAN address. This is illustrated in Figure 35. A LAN address is also variously called a physical address an Ethernet address, or a media access control (MAC) address. For most LANs (including Ethernet and token-passing LANs), the LAN address is six-bytes long, giving 248 possible LAN addresses. These six-byte addresses are typically expressed in hexadecimal notation, with each byte of the address expressed by a pair of hexadecimal numbers. An adapter's LAN address is permanent -- when an adapter is manufactured, a LAN address is burned into the adapter's ROM.

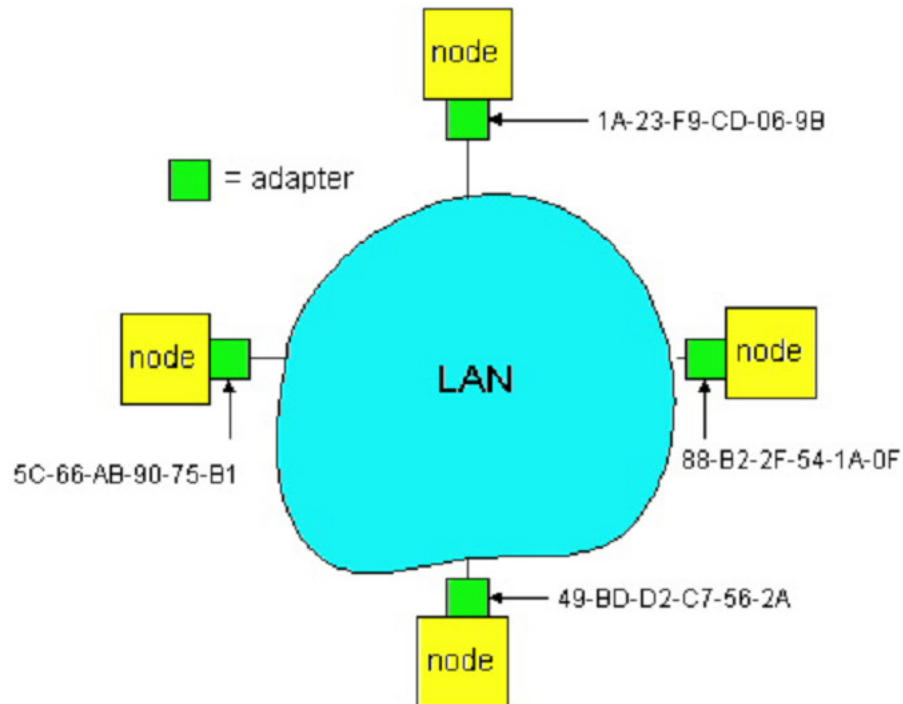


Figure 35. Each adapter connected to a LAN has a unique LAN address.

One interesting property of LAN addresses is that no two adapters have the same address. This might seem surprising given that adapters are manufactured in many different countries by many different companies. How does a company manufacturing adapters in Taiwan make sure that it is using different addresses from a company manufacturing adapters in Belgium? The answer is that IEEE manages the physical address space. In particular, when a company wants to manufacture adapters, it purchases a chunk of the address space consisting of 224 addresses for a nominal fee. IEEE allocates the chunk of 224 addresses by fixing the first 24 bits of a physical address and letting the company create unique combinations of the last 24 bits for each adapter.

An adapter's LAN address has a flat structure (as opposed to a hierarchical structure), and doesn't change no matter where the adapter goes. A portable computer with an Ethernet card always has the same LAN address, no matter where the portable goes. Recall that, in contrast, an IP address has a hierarchical structure (i.e., a network part and a host part), and a node's IP address needs to be changed when the host moves. An adapter's LAN address is analogous to a person's social security number, which also has a flat addressing structure and which also doesn't change no matter where the person goes. An IP address is analogous to a person's postal address, which is hierarchical and which needs to be changed whenever a person moves.

One natural question at this juncture is, because all nodes also have IP addresses, why do they have to have LAN addresses as well? There are several good answers to this question. First, LANs are designed for "arbitrary" network-layer protocols, not just for IP. If adapters were to get assigned IP addresses rather than "neutral" LAN addresses, then the adapters would not be able to easily support other network-layer protocols (e.g., IPX or DECNet). Second, if adapters were to use IP addresses instead



of LAN addresses, the IP address would have to be stored in adapter RAM and be configured every time the adapter were moved (or powered up). Another option is to not use any addresses in the adapters, and have each adapter pass the data (i.e., the IP datagram) of each frame it receives to its parent node. The parent node could then check for a matching IP address. One problem with this option is that the parent node will be interrupted by every frame sent on the LAN, including by the frames that are destined for other nodes on the LAN.

As we described at the beginning of this section, when an adapter wants to send a frame to some destination adapter on the same LAN, the sending adapter inserts the destination LAN address into the frame. When the destination adapter receives the frame, it extracts the enclosed datagram and passes the datagram up the protocol stack. All the other adapters on the LAN also receive the frame; but these other adapters discard the frame without passing the network-layer datagram up the protocol stack. Thus, these other adapters do not have to interrupt their hosts when they receive datagrams destined to other hosts. Having said this, sometimes a sending adapter does want all the other adapters on the LAN to receive and process the frame it is about to send. In this case, the sending adapter inserts a special LAN broadcast address into the destination address field of the frame. For LANs that use the six-byte addresses (such as Ethernet and token-passing LANs), the broadcast address is a string of 48 consecutive 1s (i.e., FF-FF-FF-FF-FF-FF in hexadecimal notation).

2.6.2.1. Address Resolution Protocol (ARP)

Because there are both network-layer addresses (e.g., Internet IP addresses) and link-layer addresses (i.e., LAN addresses), there is a need to translate between them. For the Internet, this is the job of the Address Resolution Protocol (ARP) [RFC 826]. Every Internet host and router on a LAN has an ARP module. To motivate ARP, consider the network shown in Figure 36. In this figure each node has an IP address and each node's adapter has a LAN address. As usual, IP addresses are shown in dotted-decimal notation and LAN addresses are shown in hexadecimal notation. Now suppose that the node with IP address 222.222.222.220 wants to send an IP datagram to node 222.222.222.222. To accomplish this task, the sending node must give its adapter not only the IP datagram but also the LAN address for node 222.222.222.222. When passed the IP datagram and the LAN address, the sending node's adapter can construct a data link layer frame and broadcast the frame into the LAN. But how does the sending node determine the LAN address for the node with IP address 222.222.222.222? It does this by providing its ARP module with the IP address 222.222.222.222. ARP then responds with the corresponding LAN address, namely, 49-BD-D2-C7-56-2A.

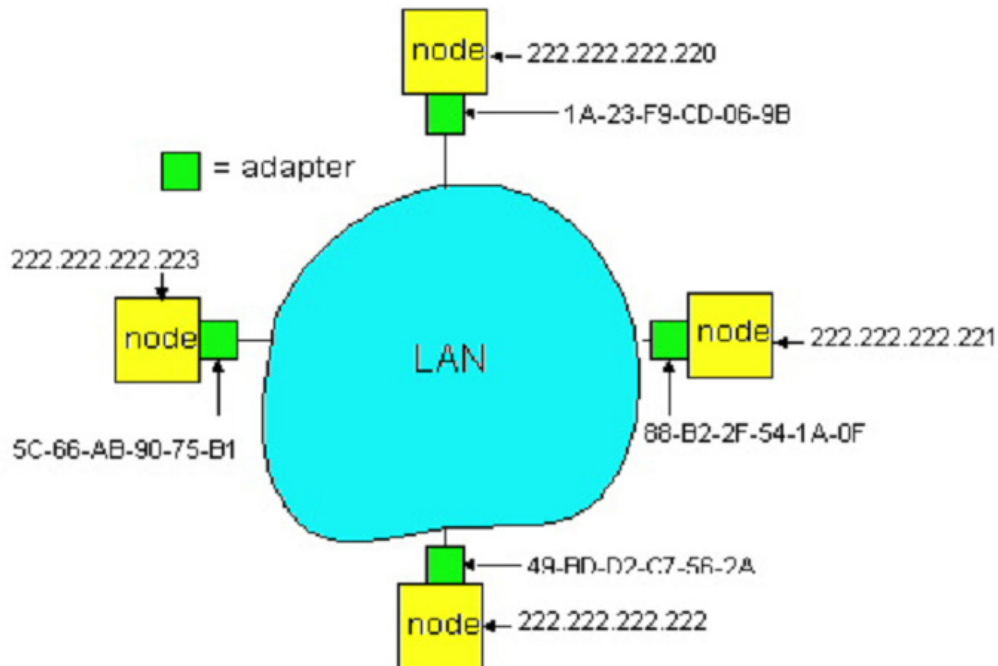


Figure 36. Each node on a LAN has an IP address, and each node's adapter has a LAN address

So we see that ARP resolves an IP address to a LAN address. In many ways it is analogous to DNS which resolves hostnames to IP addresses. However, one important difference between the two resolvers is that DNS resolves hostnames for hosts anywhere in the Internet, whereas ARP only resolves IP addresses for nodes on the same LAN. If a node in California were to try to use ARP to resolve the IP address for a node in Mississippi, ARP would return with an error.

Now that we have explained what ARP does, let's look at how it works. The ARP module in each node has a table in its RAM called an ARP table. This table contains the mappings of IP addresses to LAN addresses. Table 6 shows what an ARP table in node 222.222.222.220 might look like. For each address mapping the table also contains a time-to-live (TTL) entry, which indicates when the entry will be deleted. Note that the table does not necessarily contain an entry for every node on the LAN; some nodes may have had entries that expired over time, whereas other nodes may have never been entered into the table. We note that a typical expiration time for an entry is 20 minutes from when an entry is placed in an ARP table.

Table 6. A possible ARP table in node 222.222.222.220.

IP address	LAN address	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Now suppose that node 222.222.222.220 wants to send a datagram that is IP-addressed to another node on that LAN. As we indicated above, the sending node needs to obtain the LAN address of the destination node, given the IP address of that



node. This task is easy if the destination node has an entry in the sending node's ARP table. But what if the destination node does not currently have an entry in the ARP table? In particular, suppose node 222.222.222.220 wants to send a datagram to node 222.222.222.222. In this case, the sending node uses the ARP protocol to resolve the address. First, the sending node constructs a special packet called an ARP packet. An ARP packet has several fields, including the sending and receiving IP and LAN addresses. Both ARP query and response packets have the same format. The purpose of the ARP query packet is to query all the other nodes on the LAN to determine the LAN address corresponding to the IP address that is being resolved.

Returning to the example, node 222.222.222.220 passes an ARP query packet to the adapter along with an indication that the adapter should send the packet to the LAN broadcast address, namely, FF-FF-FF-FF-FF-FF. The adapter encapsulates the ARP packet in a data link frame, uses the broadcast address for the frame's destination address, and transmits the frame into the LAN. Recalling our social security number / postal address analogy, note that an ARP query is equivalent to a person shouting out in a crowded room of cubicles in some company (say, AnyCorp): "What is the social security number of the person whose postal address is cubicle 13, Room 112, AnyCorp, Palo Alto CA?"

The frame containing the ARP query is received by all the other adapters on the LAN, and (because of the broadcast address) each adapter passes the ARP packet within the frame up to its parent node. Each node that receives the ARP packet checks to see if its IP address matches the destination IP address in the ARP packet. The one node with a match sends back to the querying node a response ARP packet with the desired mapping. The querying node (222.222.222.220) can then update its ARP table and send its IP datagram.

There are a couple of interesting things to note about the ARP protocol. First, the query ARP message is sent within a broadcast frame whereas the response ARP message is sent within a standard frame. Before reading on you should think about why this is so. Second, ARP is plug-and-play, that is, a node's ARP table gets built automatically -- it doesn't have to be configured by a systems administrator. And if a node is disconnected from the LAN, its entry is eventually deleted from the table.

2.6.2.2. Sending a Datagram to a Node off the LAN

It should now be clear how ARP operates when a node wants to send a datagram to another node on the same LAN. But now let's look at the more complicated situation when a node on a LAN wants to send a network-layer datagram to a node off the LAN. Let us discuss this issue in the context of Figure 37, which shows a simple network consisting of two LANs interconnected by a router.

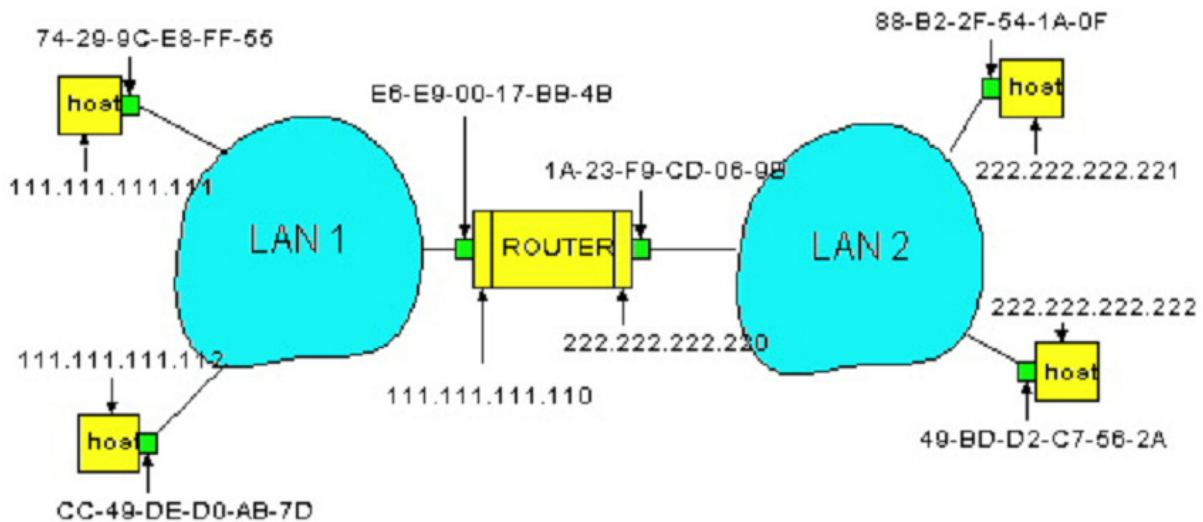


Figure 37. Two LANs interconnected by a router.

There are several interesting things to note about Figure 37. First, there are two types of nodes: hosts and routers. Each host has exactly one IP address and one adapter. A router has an IP address for each of its interfaces. Each router interface also has its own ARP module (in the router) and its own adapter. Because the router in Figure 37 has two interfaces, it has two IP addresses, two ARP modules and two adapters. Of course, each adapter in the network has its own LAN address.

Also note that all of the interfaces connected to LAN 1 have addresses of the form 111.111.111.xxx and all of the interfaces connected to LAN 2 have the form 222.222.222.xxx. Thus, in this example, the first three bytes of the IP address specifies the "network" whereas the last byte specifies the specific interface on a network.

Now suppose that host 111.111.111.111 wants to send an IP datagram to host 222.222.222.222. The sending host passes the datagram to its adapter, as usual. But the sending host must also indicate to its adapter an appropriate destination LAN address. What LAN address should the adapter use? One might venture to guess that the appropriate LAN address is the address of the adapter for host 222.222.222.222, namely, 49-BD-D2-C7-58-2A. This guess is, however, wrong. If the sending adapter were to use that LAN address, then none of the adapters on LAN 1 would bother to pass the IP datagram up to its network layer; the datagram would just die and go to datagram heaven.

If we look carefully at Figure 37, we see that in order for a datagram to go from 111.111.111.111 to a node on LAN 2, the datagram must first be sent to the router interface 111.111.111.110. Thus, the appropriate LAN address for the frame is the address of the adapter for router interface 111.111.111.110, namely, E6-E9-00-17-BB-4B. How does the sending host acquire the LAN address of 111.111.111.110? By using ARP, of course! Once the sending adapter has this LAN address, it creates a frame and sends the frame into LAN 1. The router adapter on LAN 1 sees that the data link frame is addressed to it, and therefore passes the frame to the network layer of the router. Hooray, the IP datagram has successfully been moved from source host to the

router! But we are not done. We still have to move the datagram from the router to the destination! The router now has to determine the correct interface on which the datagram is to be forwarded. This is done by consulting a routing table in the router. The routing table tells the router that the datagram is to be forwarded to router interface 222.222.222.220. This interface then passes the datagram to its adapter, which encapsulates the datagram in a new frame and sends the frame into LAN 2. This time, the destination LAN address of the frame is indeed the LAN address of the ultimate destination. And how does the router obtain this destination LAN address? From ARP, of course!

3. Bridges and Switches

3.1. Hubs

The simplest way to interconnect LANs is to use a hub. A hub is a simple device that takes an input (i.e., a frame's bits) and retransmits the input on the hub's outgoing ports. Hubs are essentially repeaters, operating on bits. They are thus physical-layer devices. When a bit comes into a hub interface, the hub simply broadcasts the bit on all the other interfaces. In this section we investigate bridges, which are another type of interconnection device.

Figure 38 shows how three academic departments in a university might interconnect their LANs. In this figure, each of the three departments has a 10BaseT Ethernet that provides network access to the faculty, staff and students of the departments. Each host in a department has a point-to-point connection to the departmental hub. A fourth hub, called a backbone hub, has point-to-point connections to the departmental hubs, interconnecting the LANs of the three departments. The design shown in Figure 38 is a multi-tier hub design because the hubs are arranged in a hierarchy. It is also possible to create multi-tier designs with more than two tiers -- for example, one tier for the departments, one tier for the schools within the university (e.g., engineering school, business school, etc.) and one tier at the highest university level. Multiple tiers can also be created out of 10Base2 (bus topology Ethernets) with repeaters.

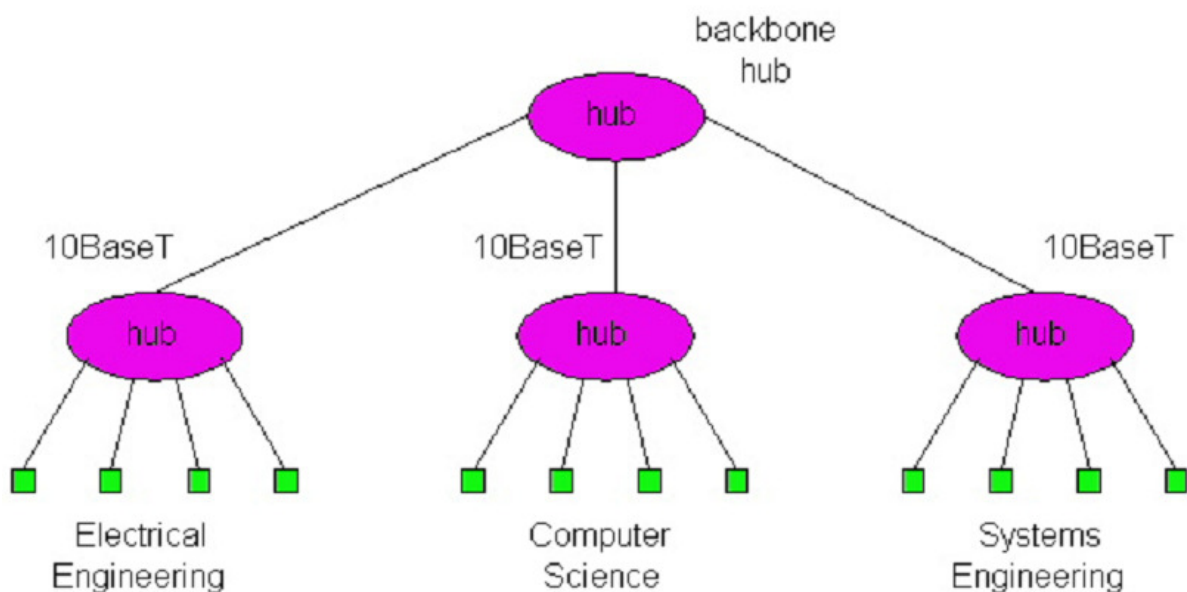


Figure 38. Three departmental Ethernets interconnected with a hub.



In a multi-tier design, we refer to the entire interconnected network as a LAN, and we refer to each of the departmental portions of the LAN (i.e., the departmental hub and the hosts that connect to the hub) as a LAN segment. It is important to note that all of the LAN segments in Figure 38 belong to the same collision domain, that is, whenever two or more nodes on the LAN segments transmit at the same time, there will be a collision and all of the transmitting nodes will enter exponential back off.

Interconnecting departmental LANs with a backbone hub has many benefits. First and foremost, it provides inter-departmental communication to the hosts in the various departments. Second, it extends the maximum distance between any pair of nodes on the LAN. For example, with 10BaseT the maximum distance between a node and its hub is 100 meters; therefore, in a single LAN segment the maximum distance between any pair of nodes is 200 meters. By interconnecting the hubs, this maximum distance can be extended, since the distance between directly-connected hubs can also be 100 meters when using twisted pair (and more when using fiber). Third, the multi-tier design provides a degree of graceful degradation. Specifically, if any one of the departmental hubs starts to malfunction, the backbone hub can detect the problem and disconnect the departmental hub from the LAN; in this manner, the remaining departments can continue to operate and communicate while the faulty departmental hub gets repaired.

Although a backbone hub is a useful interconnection device, it has three serious limitations that hinder its deployment. First, and perhaps more important, when departmental LANs are interconnected with a hub (or a repeater), then the independent collision domains of the departments are transformed into one large and common collision domain. Let us explore this latter issue in the context of Figure 38. Before interconnecting the three departments, each departmental LAN had a maximum throughput of 10 Mbps, so that maximum aggregate throughput of the three LANs was 30 Mbps. But once the three LANs are interconnected with a hub, all of the hosts in the three departments belong to the same collision domain, and the maximum aggregate throughput is reduced to 10 Mbps.

A second limitation is that if the various departments use different Ethernet technologies, then it may not be possible to interconnect the departmental hubs with a backbone hub. For example, if some department's use 10BaseT and the remaining department's use 100BaseT, so it is impossible to interconnect all the departments without some frame buffering at the interconnection point; since hubs are essentially repeaters and do not buffer frames, they cannot interconnect LAN segments operating at different rates.

A third limitation is that each of the Ethernet technologies (10Base2, 10BaseT, 100BaseT, etc.) has restrictions on the maximum number of nodes that can be in a collision domain, the maximum distance between two hosts in a collision domain, and the maximum number of tiers that can be present in a multi-tier design. These restrictions constrain both the total number of hosts that connect to a multi-tier LAN as well as geographical reach of the multi-tier LAN.

3.2. Bridges

In contrast to hubs, which are physical-level devices, bridges operate on Ethernet frames and thus are layer-2 devices. In fact, bridges are full-fledged packet switches that forward and filter frames using the LAN destination addresses. When a frame comes into a bridge interface, the bridge does not just copy the frame onto all of the other interfaces. Instead, the bridge examines the destination address of the frame and attempts to forward the frame on the interface that leads to the destination.

Figure 39 shows how the three academic departments of our previous example might be interconnected with a bridge. The three numbers next to the bridge are the interface numbers for the three bridge interfaces. When the departments are interconnected by a bridge, as in Figure 39, we again refer to the entire interconnected network as a LAN, and we again refer to each of the departmental portions of the network as LAN segments. But in contrast to the multi-tier hub design in Figure 38, each LAN segment is now an isolated collision domain.

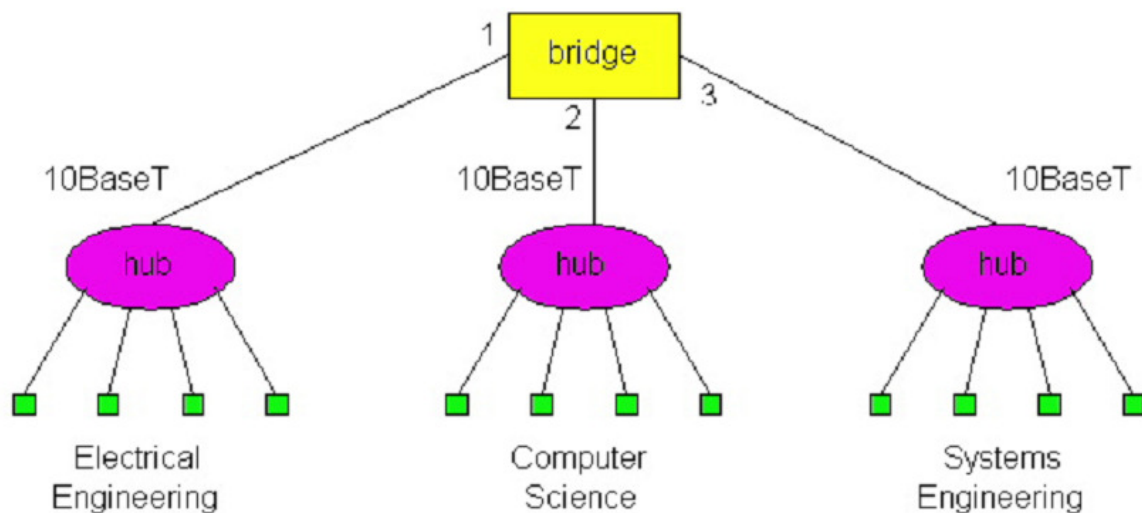


Figure 39. Three departmental LANs interconnected with a bridge.

Bridges can overcome many of the problems that plague hubs. First, bridges permit inter-departmental communication while preserving isolated collision domains for each of the departments. Second, bridges can interconnect different LAN technologies, including 10 Mbps and 100 Mbps Ethernets. Third, there is no limit to how big a LAN can be when bridges are used to interconnect LAN segments: in theory, using bridges, it is possible to build a LAN that spans the entire globe.

3.2.1. Forwarding and Filtering

Filtering is the ability to determine whether a frame should be forwarded to an interface or should just be dropped. When the frame should be forwarded, forwarding is the ability to determine which of the interfaces the frame should be directed to. Bridge filtering and forwarding are done with a bridge table. For each node on the LAN, the bridge table contains (1) the LAN address of the node, (2) the bridge interface that leads towards the node, (3) and the time at which the entry for the node was placed in the table. Table 7 shows an example for the LAN in Figure 39. This description of frame forwarding may sound similar to our discussion of datagram forwarding. We note

here that the addressees used by bridges are physical addresses (not network addresses). We will also see shortly that a bridge table is constructed in a very different manner than routing tables.

Table 7. Portion of a bridge table for the LAN in Figure 39

Address	Interface	Time
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
...

To understand how bridge filtering and forwarding works, suppose a frame with destination address DD-DD-DD-DD-DD-DD arrives to the bridge on interface x. The bridge indexes its table with the LAN address DD-DD-DD-DD-DD-DD and finds the corresponding interface y.

- If x equals y, then the frame is coming from a LAN segment that contains adapter DD-DD-DD-DD-DD-DD. There being no need to forward the frame to any of the other interfaces, the bridge performs the filtering function by discarding the frame.
- If x does not equal y, then the frame needs to be routed to the LAN segment attached to interface y. The bridge performs its forwarding function by putting the frame in an output buffer that precedes interface y.

These simple rules allow a bridge to preserve separate collision domains for each of the different LAN segments connected to its interfaces. The rules also allow the nodes on different LAN segments to communicate.

Let's walk through these rules for the network in Figures 39 and its bridge table in Table 7. Suppose that a frame with destination address 62-FE-F7-11-89-A3 arrives at the bridge from interface 1. The bridge examines its table and sees that the destination is on the LAN segment connected to interface 1 (i.e., the Electrical Engineering LAN). This means that the frame has already been broadcast on the LAN segment that contains the destination. The bridge therefore filters (i.e., discards) the frame. Now suppose a frame with the same destination address arrives from interface 2. The bridge again examines its table and sees that the destination is the direction of interface 1; it therefore forwards the frame to the output buffer preceding interface 1. It should be clear from this example that as long as the bridge table is complete and accurate, the bridge isolates the departmental collision domains while permitting the departments to communicate.

Recall that when a hub (or a repeater) forwards a frame onto a link, it just sends the bits onto the link without bothering to sense whether another transmission is currently taking place on the link. In contrast, when a bridge wants to forward a frame onto a link, it runs the CSMA/CD algorithm. In particular, the bridge refrains from transmitting if it senses that some other node on the LAN segment is transmitting; furthermore, the bridge uses exponential back off when one of its transmissions results in a collision.



Thus bridge interfaces behave very much like node adapters. But technically speaking, they are not node adapters because neither a bridge nor its interfaces have LAN addresses. Recall that a node adapter always inserts its LAN address into the source address of every frame it transmits. This statement is true for router adapters as well as host adapters. A bridge, on the other hand, does not change the source address of the frame.

One significant feature of bridges is that they can be used to combine Ethernet segments using different Ethernet technologies. For example, if in Figure 37, Electrical Engineering has a 10Base2 Ethernet, Computer Science has a 100BaseT Ethernet, and Electrical Engineering has a 10BaseT Ethernet, then a bridge can be purchased that can interconnect the three LANs. With Gigabit Ethernet bridges, it is possible to have an additional 1 Gbps connection to a router, which in turn connects to a larger university network. As we mentioned earlier, this feature of being able to interconnect different link rates is not available with hubs.

Also, when bridges are used as interconnection devices, there is no theoretical limit to the geographical reach of a LAN. In theory, we can build a LAN that spans the globe by interconnecting hubs in a long, linear topology, with each pair of neighboring hubs interconnected by a bridge. Because in this design each of the hubs has its own collision domain, there is no limit on how long the LAN can be. We shall see shortly, however, that it is undesirable to build very large networks exclusively using bridges as interconnection devices -- large networks need routers as well.

3.3. Bridges versus Routers

Routers are store-and-forward packet switches that forward packets using IP addresses.

Although a bridge is also a store-and-forward packet switch, it is fundamentally different from a router in that it forwards packets using LAN addresses. Whereas a router is layer-3 packet switch, a bridge is a layer-2 packet switch.

Even though bridges and routers are fundamentally different, network administrators must often choose between them when installing an interconnection device. For example, for the network in Figure 39, the network administrator could have just as easily used a router instead of a bridge. Indeed, a router would have also kept the three collision domains separate while permitting interdepartmental communication. Given that both bridges and routers are candidates for interconnection devices, what are the pros and cons of the two approaches?

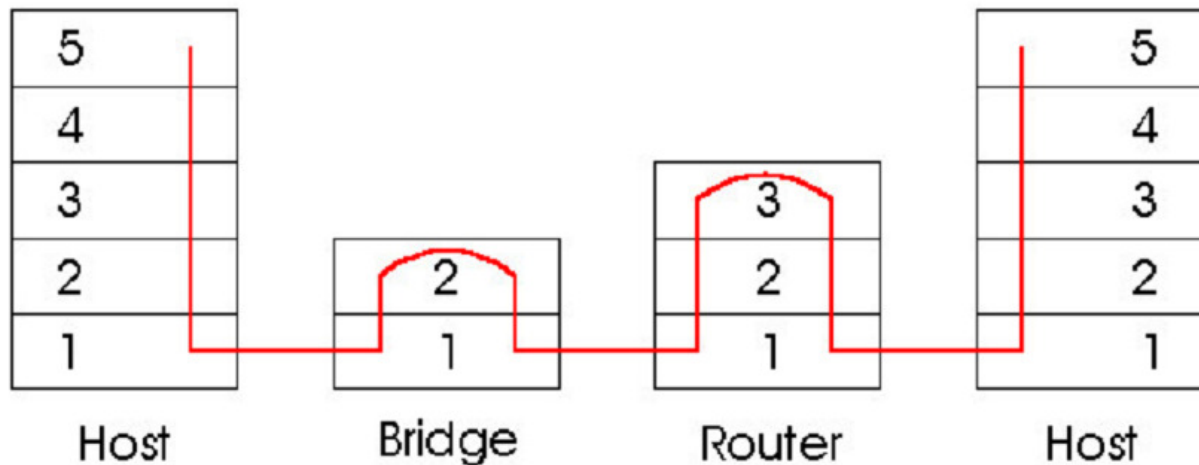


Figure 40. Packet processing and bridges, routers and hosts.

First consider the pros and cons of bridges. As mentioned above, bridges are plug and play, a property that is cherished by all the over-worked network administrators of the world. Bridges can also have relatively high packet filtering and forwarding rates -- as shown in Figure 40, bridges only have to process packets up through layer 2, whereas routers have to process frames up through layer 3. On the other hand, the spanning tree protocol restricts the effective topology of a bridged network to a spanning tree. This means that all frames most flow along the spanning tree, even when there are more direct (but disconnected) paths between source and destination. The spanning tree restriction also concentrates the traffic on the spanning tree links when it could have otherwise been spread through all the links of the original topology. Furthermore, bridges do not offer any protection against broadcast storms -- if one host goes haywire and transmits an endless stream of Ethernet broadcast packets, the bridges will forward all of the packets and the entire network will collapse.

Now consider the pros and cons of routers. Because IP addressing is hierarchical (and not flat like LAN addressing), packets do not normally cycle through routers even when the network has redundant paths. (Actually, packets can cycle when router tables are misconfigured; but IP uses a special datagram header field to limit the cycling.) Thus, packets are not restricted to a spanning tree and can use the best path between source and destination. Because routers do not have the spanning tree restriction, routers have allowed the Internet to be built with a rich topology which includes, for example, multiple active links between Europe and North America. Another feature of routers is that they provide firewall protection against layer-2 broadcast storms. Perhaps the most significant drawback of routers is that they are not plug and play-- they and the hosts that connect to them need their IP addresses to be configured. Also, routers often have a larger prepackage processing time than bridges, because they have to process up through the layer-3 fields. Finally, there are two different ways to pronounce the word "router", either as "rooter" or as "rowter", and people waste a lot of time arguing over the proper pronunciation.

Given that both bridges and routers have their pros and cons, when should an institutional network (e.g. university campus network or a corporate campus network)

use bridges, and when should it use routers? Typically, small networks consisting of a few hundred hosts have a few LAN segments. Bridges suffice for these small networks, as they localize traffic and increase aggregate throughput without requiring any configuration of IP addresses. But larger networks consisting of thousands of hosts typically include routers within the network (in addition to bridges). The routers provide a more robust isolation of traffic, control broadcast storms, and use more "intelligent" routes among the hosts in the network.

3.4. Switches

Up until the mid-1990s, three types of LAN interconnection devices were essentially available: hubs (and their cousins, repeaters), bridges and routers. More recently yet another interconnection device became widely available, namely, Ethernet switches. Ethernet switches, often trumpeted by network equipment manufacturers with great fanfare, are in essence high-performance multi-interface bridges. Like bridges, they forward and filter frames using LAN destination addresses, and they automatically build routing tables using the source addresses in the traversing frames. The most important difference between a bridge and a switch is that bridges usually have a small number of interfaces (i.e. 2-4), whereas switches may have dozens of interfaces. A large number of interfaces generates a high aggregate forwarding rate through the switch fabric, therefore necessitating a high-performance design (especially for 100 Mbps and 1 Gbps interfaces).

Switches can be purchased with various combinations of 10 Mbps, 100 Mbps and 1 Gbps interfaces. For example, you can purchase switches with four 100 Mbps interfaces and twenty 10 Mbps interfaces; or switches with four 100 Mbps interfaces and one 1 Gbps interface. Of course, the more the number of interfaces and the higher the transmission rates of the various interfaces, the more you pay. Many switches also operate in a full-duplex mode; that is, they can send and receive frames at the same time over the same interface. With a full duplex switch (and corresponding full duplex Ethernet adapters in the hosts), host A can send a file to host B while that host B simultaneously sends to host A.

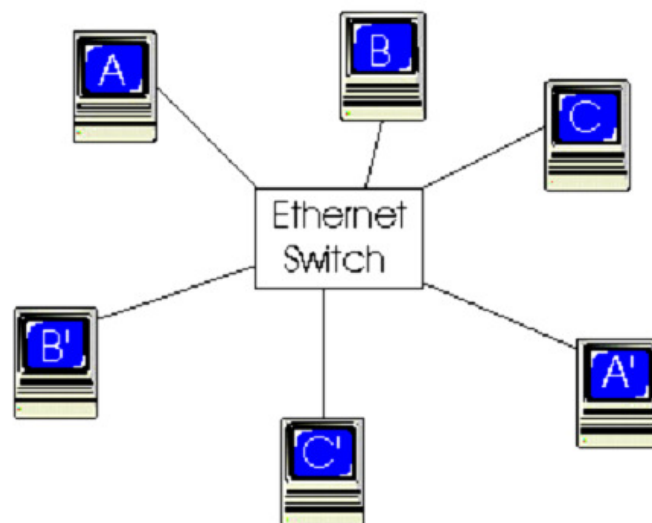


Figure 41. An Ethernet switch providing dedicated Ethernet access to six hosts.

One of the advantages of having a switch with a large number of interfaces is that it creates direct connections between hosts and the switch. When a host has a full-duplex direct connection to a switch, it can transmit (and receive) frames at the full transmission rate of its adapter; in particular, the host adapter always senses an idle channel and never experiences a collision. When a host has a direct connection to a switch (rather than a shared LAN connection), the host is said to have dedicated access. In Figure 41, an Ethernet switch provides dedicated access to six hosts. This dedicated access allows A to send a file to A' while B is sending a file to B' and C is sending a file to C'. If each host has a 10Mbps adapter card, then the aggregate throughput during the three simultaneous file transfers is 30 Mbps. If A and A' have 100 Mbps adapters and the remaining hosts have 10 Mbps adapters, then the aggregate throughput during the three simultaneous file transfers is 120 Mbps.

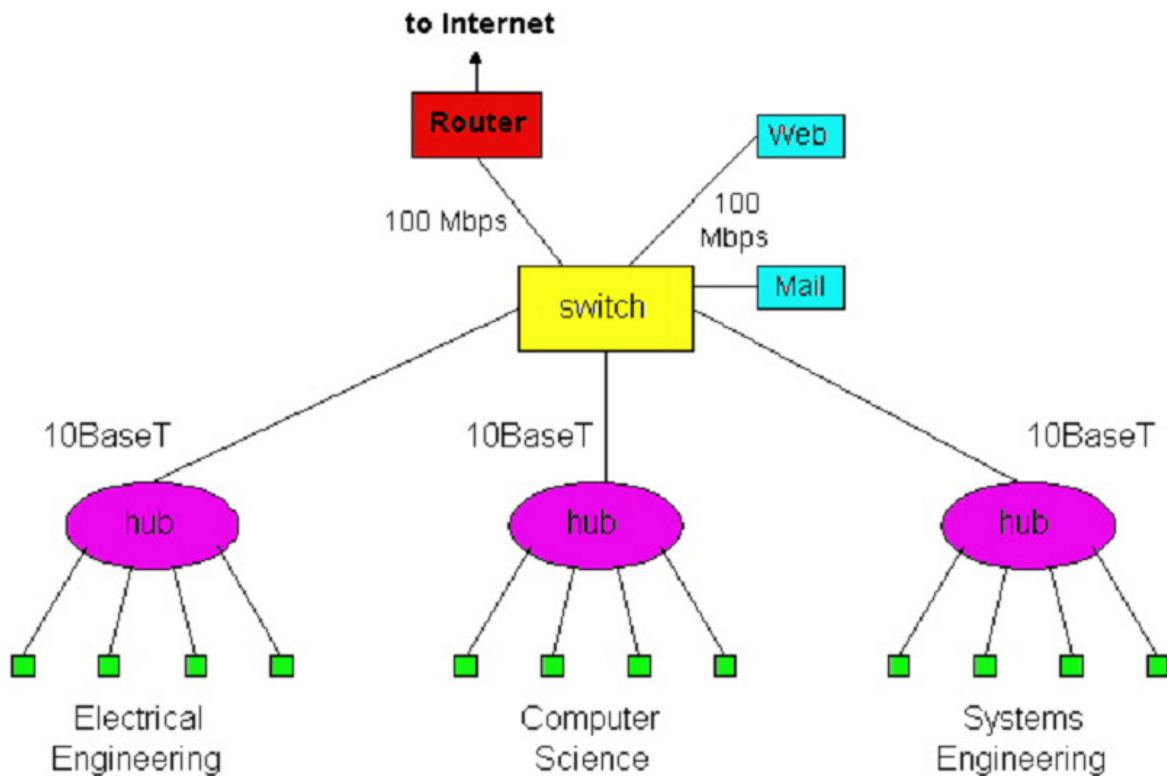


Figure 42. An institutional network using a combination of hubs, Ethernet switches and a router.

Figure 42 shows how an institution with several departments and several critical servers might deploy a combination of hubs, Ethernet switches and routers. In Figure 42, each of the three departments has its own 10 Mbps Ethernet segment with its own hub. Because each departmental hub has a connection to the switch, all intra-departmental traffic is confined to the Ethernet segment of the department (assuming the routing tables in the Ethernet switch are complete). The Web and each mail servers have dedicated 100 Mbps access to the switch. Finally, a router, leading to the Internet, has dedicated 100 Mbps access to the switch. Note that this switch has at least three 10 Mbps interfaces and three 100 Mbps interfaces.

4. Industrial Ethernet Technologies

Industrial Ethernet is the name given to the use of the Ethernet network protocol in an industrial environment, for automation and process control. A number of techniques are used to adapt the Ethernet protocol to the needs of industrial processes, which must provide real time behavior. By using non-proprietary protocols, automation systems from different manufacturers can be interconnected throughout a process plant. Industrial Ethernet takes advantage of the relatively larger marketplace for computer interconnections to reduce cost and improve performance of communications between industrial controllers.

Industrial Ethernet (IE) components used in plant process areas must be designed to work in harsh environments of temperature extremes, humidity, and vibration that exceed the ranges for IT equipment intended for installation in controlled environments.

Some of the advantages are:

- Increased speed, up from 9.6 kbit/s with RS-232 to 1 Gbit/s with IEEE 802 over Cat5e/Cat6 cables or optical fiber
- Increased overall performance
- Increased distance
- Ability to use standard access points, routers, switches, hubs, cables and optical fiber, which are far cheaper than the equivalent serial-port devices
- Ability to have more than two nodes on link, which was possible with RS-485 but not with RS-232
- Peer-to-peer architectures may replace master-slave ones
- Better interoperability

The difficulties of using Industrial Ethernet are:

- Migrating existing systems to a new protocol (however, many adapters are available)
- Real-time uses may suffer for protocols using TCP (but some use UDP and layer 2 protocols for this reason)
- Managing a whole TCP/IP stack is more complex than just receiving serial data
- The minimum Fast Ethernet frame size including inter-frame spacing is about 80 bytes, while typical industrial communication data sizes can be closer to 1-8 bytes. This often results in a data transmission efficiency of less than 5%, negating any advantages of the higher bitrate.
 - On Gigabit Ethernet the minimum frame size is 512 Bytes, reducing the typical efficiency to less than 1%.
 - Some of the Industrial Ethernet protocols introduce modifications to the Ethernet protocol to improve efficiency.

Table 8. Main protocols

Serial	Ethernet	Protocol	Network
Modbus-RTU	Modbus-TCP	TCP/IP	
Profibus	PROFINET IO	Isochronous real time protocol (IRT), Real time protocol (RT), Real time over UDP protocol (RTU)	Switches, router and wireless, from 100 Mbit/s up to 1 Gbit/s
DeviceNet (CIP); ControlNet (CIP)	Ethernet/IP (CIP)	TCP/IP; UDP/IP	Switches, router and wireless, from 100 Mbit/s up to 1 Gbit/s
Foundation Fieldbus H1	Foundation Fieldbus High Speed Ethernet (HSE)		
CANopen	Ethernet Powerlink		Ethernet 100Mbit/s
CANopen	EtherCAT	EtherCAT, EtherCAT/UDP	Ethernet 100Mbit/s
	VARAN Versatile Automation Random Access Network	VARAN, TCP/IP, Safety	Ethernet 100Mbit/s
SERCOS I / II	SERCOS III		Ethernet 100Mbit/s
	FL-Net (OPCN-2)	UDP/IP	Ethernet 10Mbit/s

4.1. Non-Real Time Applications

4.1.1. Modbus

Modbus is a serial communications protocol published by Modicon in 1979 for use with its programmable logic controllers (PLCs). It has become a de facto standard communications protocol in industry, and is now the most commonly available means of connecting industrial electronic devices. The main reasons for the extensive use of Modbus over other communications protocols are:

1. It is openly published and royalty-free
2. Relatively easy industrial network to deploy
3. It moves raw bits or words without placing many restrictions on vendors

Modbus allows for communication between many devices connected to the same network, for example a system that measures temperature and humidity and communicates the results to a computer. Modbus is often used to connect a supervisory computer with a remote terminal unit (RTU) in supervisory control and data acquisition (SCADA) systems.

4.1.1.1. Protocol versions

Versions of the Modbus protocol exist for serial port and for Ethernet and other networks that support the Internet protocol suite. Most Modbus devices communicate over a serial EIA-485 physical layer. There are many variants of Modbus protocols

- Modbus RTU: This is used in serial communication and makes use of a compact, binary representation of the data for protocol communication. The RTU format follows the commands/data with a cyclic redundancy check checksum as an error check mechanism to ensure the reliability of data. Modbus RTU is the most common implementation available for Modbus. A Modbus RTU message must be transmitted continuously without inter-character hesitations. Modbus messages are framed (separated) by idle (silent) periods.
- Modbus ASCII: This is used in serial communication and makes use of ASCII characters for protocol communication. The ASCII format uses a longitudinal redundancy check checksum. Modbus ASCII messages are framed by leading colon (':') and trailing newline (CR/LF).
- Modbus TCP/IP or Modbus TCP: This is a modbus variant used for communications over TCP/IP networks. It does not require a checksum calculation as lower layer takes care of the same.
- Modbus over TCP/IP or Modbus over TCP: This is a modbus variant that differs from Modbus TCP in that a checksum is included in the payload as with Modbus RTU.
- Modbus Plus (Modbus+, MB+ or MBP): An extended version, Modbus Plus (Modbus+ or MB+), also exists, but remains proprietary to Schneider Electric. It requires a dedicated co-processor to handle fast HDLC-like token rotation. It uses twisted pair at 1 Mbit/s and includes transformer isolation at each node, which makes it transition/edge triggered instead of voltage/level triggered. Special interfaces are required to connect Modbus Plus to a computer, typically a card made for the ISA (SA85), PCI or PCMCIA bus.

Data model and function calls are identical for the first 4 variants of protocols; only the encapsulation is different. However the variants are not interoperable as the frame formats are different.

4.1.1.2. Communication and Devices

Each device intended to communicate using Modbus is given a unique address. In serial and MB+ networks only the node assigned as the Master may initiate a command, but on Ethernet, any device can send out a Modbus command, although usually only one master device does so. A Modbus command contains the Modbus address of the device it is intended for. Only the intended device will act on the command, even though other devices might receive it (an exception is specific broadcastable commands sent to node 0 which are acted on but not acknowledged).

All Modbus commands contain checking information, ensuring that a command arrives undamaged. The basic Modbus commands can instruct an RTU to change a value in one of its registers, control or read an I/O port, as well as commanding the device to send back one or more values contained in its registers.

There are many modems and gateways that support Modbus, as it is a very simple protocol and often copied. Some of them were specifically designed for this protocol. Different implementations use wireline, wireless communication and even SMS or GPRS. Typical problems the designers have to overcome include high latency and timing problems.

4.1.1.3. Frame Format

All modbus variants choose different frame format.

Table 9. Frame formats for modbus variants.

Modbus RTU Frame Format		
Name	Length	Function
Start	3.5c idle	<i>at least 3-1/2 character times of silence (MARK condition)</i>
Address	8 bits	<i>Station Address</i>
Function	8 bits	<i>Indicates the function codes like read coils / inputs</i>
Data	n*8 bits	<i>Data + length will be filled depending on the message type</i>
CRC Check	16 bits	<i>Error checks</i>
End	3.5c idle	<i>at least 3-1/2 character times of silence between frames</i>

Name	Length	Function
Start	1 char	<i>starts with colon (:) (ASCII value is 3A hex)</i>
Address	2 chars	<i>Station Address</i>
Function	2 chars	<i>Indicates the function codes like read coils / inputs</i>
Data	n chars	<i>Data +length will be filled depending on the message type</i>
LRC Check	2 chars	<i>Error checks</i>
End	2 chars	<i>carriage return – line feed(CRLF) pair (ASCII values of 0D & 0A hex)</i>

Modbus TCP Frame Format		
Name	Length	Function
Transaction Identifier	2 bytes	<i>For synchronization between messages of server & client</i>
Protocol Identifier	2 bytes	<i>Zero for MODBUS/TCP</i>
Length Field	2 bytes	<i>Number of remaining bytes in this frame</i>
Unit Identifier	1 byte	<i>Slave Address (255 if not used)</i>
Function code	1 byte	<i>Function codes as in other variants</i>
Data bytes	n bytes	<i>Data as response or commands</i>

Unit identifier is used with MODBUS/TCP devices that are composites of several MODBUS devices, e.g. on MODBUS/TCP to MODBUS RTU gateways. In such cases, the unit identifier tells the Slave Address of the device behind the gateway. MODBUS/TCP-capable devices usually ignore the Unit Identifier.

4.1.1.4. Supported Function Codes

Modbus function codes / data types include the following types. Most commonly used are given in *italics*.

- *01 Read Coil Status*
- *02 Read Input Status*
- *03 Read Holding Registers*
- *04 Read Input Registers*
- *05 Force Single Coil*
- *06 Preset Single Register*
- *07 Read Exception Status*

- 08 Diagnostics
- 09 Program 484
- 10 Poll 484
- 11 Fetch Communication Event Counter
- 12 Fetch Communication Event Log
- 13 Program Controller
- 14 Poll Controller
- *15 Force Multiple Coils*
- *16 Preset Multiple Registers*
- 17 Report Slave ID
- 18 Program 884/M84
- 19 Reset Comm. Link
- 20 Read General Reference
- 21 Write General Reference
- 22 Mask Write 4X Register
- 23 Read/Write 4X Registers
- 24 Read FIFO Queue

4.1.1.5. Implementations

Almost all implementations have variations from the official standard. Different varieties might not communicate correctly between equipment of different suppliers. Some of the most common variations are:

- Data types
 - Floating point IEEE
 - 32-bit integer
 - 8-bit data
 - Mixed data types
 - Bit fields in integers
 - Multipliers to change data to/from integer. 10, 100, 1000, 256 ...
- Protocol extensions
 - 16-bit slave addresses
 - 32-bit data size (1 address = 32 bits of data returned.)
 - Word swapped data

4.1.1.6. Limitations

- Since Modbus was designed in the late 1970s to communicate to programmable logic controllers, the number of data types is limited to those understood by PLCs at the time. Large binary objects are not supported.
- No standard way exists for a node to find the description of a data object, for example, to determine if a register value represents a temperature between 30 and 175 degrees.

- Since Modbus is a master/slave protocol, there is no way for a field device to "report by exception" (except over Ethernet TCP/IP, called open-mpbus) - the master node must routinely poll each field device, and look for changes in the data. This consumes bandwidth and network time in applications where bandwidth may be expensive, such as over a low-bit-rate radio link.
- Modbus is restricted to addressing 247 devices on one data link, which limits the number of field devices that may be connected to a master station (once again Ethernet TCP/IP proving the exception).
- Modbus transmissions must be contiguous which limits the types of remote communications devices to those that can buffer data to avoid gaps in the transmission.
- Modbus protocol provides no security against unauthorized commands or interception of data.

4.1.2. Ethernet IP

EtherNet/IP (Ethernet Industrial Protocol) is a communications protocol developed by Rockwell Automation, managed by ODVA and designed for use in process control and other industrial automation applications. EtherNet/IP is an application layer protocol and it considers all the devices on the network as a series of "objects". EtherNet/IP is built on the widely used (CIP), which gives seamless access to objects from controlNet and DeviceNet networks.

EtherNet/IP makes use of existing network infrastructure (Ethernet) and the entire EtherNet/IP stack can be implemented in software on a microprocessor. No special hardware such as (application-specific integrated circuits) or (field programmable gate arrays) are required for implementing EtherNet/IP.

EtherNet/IP is built on the standard TCP/IP stack, and makes use of all 7 layers of the OSI reference model. Since EtherNet/IP makes use of Ethernet for the physical layer, it becomes easier to tightly couple enterprise servers with the device data (sensors, actuators, drives, valves etc...).

As the technology for improving Ethernet physical layer progresses, EtherNet/IP will benefit indirectly. So, as the Ethernet physical layer technology advances from 10/100 Mbit/s to 1 Gbit/s, EtherNet/IP data transfers will also get faster.

EtherNet/IP can be used in automation networks which can tolerate some amount of non-determinism. This is because Ethernet physical media, by definition, is not deterministic.

EtherNet/IP can be easily confused as a combination of Ethernet (the physical layer, link, or medium used in most office and many industrial networking environments) and the Internet Protocol, the world's most ubiquitous (internet) networking protocol and part of the TCP/IP model, comprising a suite of protocols operating at the link, internet (or networking), transport, and application layers.

By comparison, EtherNet/IP is an industrial application layer protocol operating over the Ethernet medium and used for communication between industrial control systems and their components, such as a programmable automation controller, programmable logic controller or an I/O system. Furthermore, the "IP" in EtherNet/IP, is not an abbreviation for "Internet Protocol" but instead stands for "Industrial Protocol", referring to Rockwell's adoption of Common Industrial Protocol (CIP) standards while EtherNet/IP was being developed.

Confusing matters further is the fact that when using the OSI Reference Model, EtherNet/IP, at the application layer, operates over both Ethernet (at the physical layer) and IP (at the network layer) and thus complements each of these technologies.

4.1.2.1. Common Industrial Protocol (CIP)

CIP provides a wide range of standard objects and services for access to data and for control of network devices via so-called "implicit" and "explicit" messages. The CIP data packets are encapsulated before they are sent with standard TCP or UDP telegrams on the Ethernet.

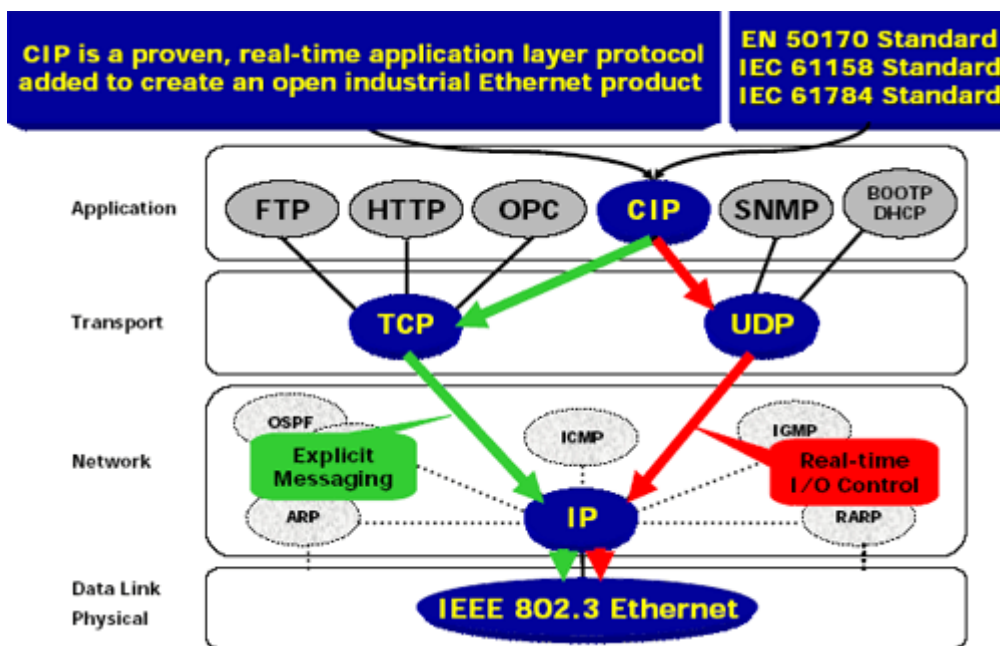


Figure 43. Common Industrial Protocol.

Ethernet/IP uses all the transport and control protocols of standard Ethernet including the Transport Control Protocol (TCP), the User Datagram Protocol (UDP), the Internet Protocol (IP) and the media access and signaling technologies found in off-the-shelf Ethernet technology. Building on these standard communication technologies means that Ethernet/IP works transparently with all the standard Ethernet devices found in today's market-place. It also means that Ethernet/IP automatically benefits from all further technology enhancements such as Gigabit Ethernet and Wireless technologies

4.1.2.2. Configuration with EDS-Files

During the setup phase of the Ethernet/IP network, the Ethernet/IP Master Scanner must be configured with a special configuration tool such as Rockwell's RSNetWorx. The configuration process is based on electronic device data sheets (EDS-Files) which are required for each Ethernet/IP device. EDS-Files are provided by the device manufacturers and contain electronic descriptions all relevant communication parameters and objects of the Ethernet/IP device. Table 10 shows some of the basic features of Ethernet/IP.

Table 10. Ethernet/IP Facts

ETHERNET/IP FACTS	
Network Type	Ethernet based Control Level network with CIP application protocol
Topology	Very flexible with Star, Tree and line topologies Switched Ethernet preferred for good Real-Time behavior
Installation	-Standard Off the Shelf (COTS) Ethernet cables and connectors -Shielded 10/100 Mbit/s TX Ethernet cable or Fibre Optics -RJ45, M12 or Fibre Optic connectors
Speed	10, 100, 1000 Mbit/s
max. Stations	nearly unlimited
Data	cyclic and acyclic process and parameter data up to 1.500 Byte per telegram frame
Network Features	Advanced Ethernet based communication system using standard Ethernet-TCP/IP and UDP protocols in Layer 1-4 and the CIP protocol in Layer 7. Transparent coupling with DeviceNet and ControlNet.
User Organization	Open DeviceNet User Organization (ODVA)

4.1.2.3. Technical Details

EtherNet/IP classifies Ethernet nodes as predefined device types with specific behaviors. Among other things, this enables:

- Transfer of basic I/O data via UDP-based implicit messaging
- Uploading and downloading of parameters, setpoints, programs and recipes via TCP (i.e., explicit messaging.)
- Polled, cyclic and change-of-state monitoring via UDP, such as RPI and COS in Allen Bradley's ControlLogix control systems.
- One-to-one (unicast), one-to-many (multicast), and one-to-all (broadcast) communication via TCP.
- EtherNet/IP makes use of the well-known TCP port number 44818 for explicit messaging and UDP port number 2222 for implicit messaging

The EtherNet/IP application layer protocol is based on the Common Industrial Protocol (CIP) standard used in DeviceNet, CompoNet and ControlNet.

As EtherNet/IP is now an open technology, it was suggested to publish the Level 2 source codes via sourceforge.net. However, in lieu of this, freeware source code was available to be downloaded from ODVA's website. At this point in time the ODVA requires that users be registered which means that a vendor ID is required and the code and the standard can no longer be considered free.

4.2. Real Time Applications

4.2.1. Ethernet Powerlink

Ethernet POWERLINK is a deterministic real-time protocol for standard Ethernet. It is an open protocol managed by the Ethernet POWERLINK Standardization Group (EPSG). It was introduced by the Austrian automation company B&R in 2001.

This protocol has nothing to do with power distribution via Ethernet cabling or power over Ethernet (PoE), power line communication or Bang & Olufsens PowerLink cable.

4.2.1.1. Overview

Ethernet POWERLINK was created taking care of standard-compliance. It expands Ethernet with a mixed Polling- and Timeslicing mechanism. That brings:

- Guaranteed transfer of time-critical data in very short isochronic cycles with configurable response time
- Time-synchronisation of all nodes in the network with very high precision of sub-microseconds
- Transmission of less timecritical data in a reserved asynchronous channel

Modern implementations reach cycle-times of fewer than 200 μ s and a time-precision (jitter) of less than 1 μ s.

4.2.1.2. Standardization

POWERLINK is standardized by the opened user- and producer-group EPSG (Ethernet POWERLINK Standardization Group) as a public standard. EPSG was founded in June 2003 as an independent association. Its focus is to leverage the advantages of Ethernet for high performance Real-Time networking systems based on the Ethernet POWERLINK Real-Time protocol, introduced by B&R towards the end of 2001. Various working groups are focusing on different tasks such as safety, technology, marketing, certification and end users. The EPSG is cooperating with leading standardization bodies and associations, such as the CAN in Automation (CIA) Group and the IEC.

4.2.1.3. Physical layer

The original physical layer specified was 100Base-X Fast Ethernet (IEEE 802.3). Since the end of 2006 Ethernet Powerlink with Gigabit Ethernet supporting a transmission rate ten times higher (1,000 Mbit/s) is in use. That offers a better performance for the

future to extended systems with major production performance, a series of module control systems, numerous drives and completely integrated security equipment. Gigabit Ethernet is on the threshold of general dissemination in IT systems. There is no need for major changes in system designs, components or cabling. Networked units that master this high transmission rate and a somewhat better cable (Cat6) must be used. The transition to higher speed Ethernet variants is always possible since Ethernet Powerlink is also attached as a standard Ethernet design and can launch its applications with standard modules such as microcomputers and FPGA modules. Usage of repeating hubs instead of switches within the Real-time domain is recommended to minimize delay and jitter. Ethernet Powerlink uses IAONA's Industrial Ethernet Planning and Installation Guide for clean cabling of industrial networks and both industrial Ethernet connectors 8P8C (RJ45) and M1 are accepted.

4.2.1.4. Data Link Layer

The Standard Ethernet Data Link Layer of Ethernet Powerlink is extended by an additional bus scheduling mechanism which ensures that secures that only one node is accessing the network at a time. The schedule is divided into an isochronous phase and an asynchronous phase. During the isochronous phase, time-critical data is transferred, while the asynchronous phase provides bandwidth for the transmission of non time-critical data. The Managing Node (MN) grants access to the physical medium via dedicated poll request messages. As a result, only one single node (CN) has access to the network at a time, which avoids collisions, usually present on standard Ethernet. The CSMA/CD mechanism of standard Ethernet, which causes non-deterministic Ethernet behaviour, is deactivated by the collision avoidance mechanism of the Ethernet Powerlink scheduling mechanism.

4.2.1.5. Basic Cycle

After system start-up is finished, the Real-Time domain is operating under Real-Time conditions. The scheduling of the basic cycle is controlled by the Managing Node (MN). The overall cycle time depends on the amount of isochronous data, asynchronous data and the number of nodes to be polled during each cycle.

The basic cycle consists of the following phases:

- **Start Phase:** The Managing Node sends out a synchronization message to all nodes. The frame is called SoC - Start of Cycle.
- **Isochronous Phase:** The Managing Node calls each node to transfer time-critical data for process or motion control by sending the Preq - Poll Request - frame. The addressed node answers with the Pres - Poll Response - frame. Since all other nodes are listening to all data during this phase, the communication system provides a producer-consumer relationship.

The time frame which includes Preq-n and Pres-n is called *time slot* for the addressed node.

- Asynchronous Phase:** The Managing Node grants the right to one particular node for sending ad-hoc data by sending out the SoA - Start of Asynchronous - frame. The addressed node will answer with ASnd. Standard IP-based protocols and addressing can be used during this phase.

The quality of the Real-Time behaviour depends on the precision of the overall basic cycle time. The length of individual phases can vary as long as the total of all phases remain within the basic cycle time boundaries. Adherence to the basic cycle time is monitored by the Managing Node. The duration of the isochronous and the asynchronous phase can be configured.

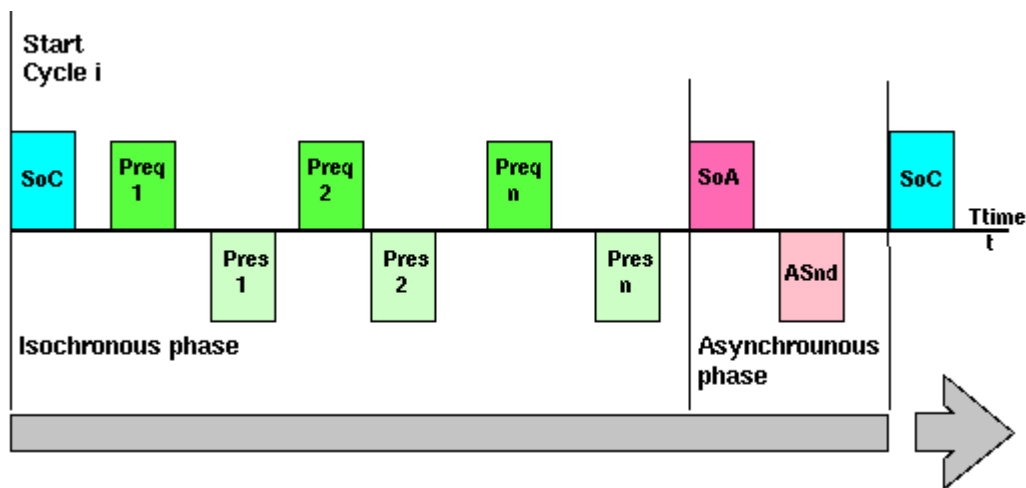


Figure 44: Frames above the time line are sent by the MN, below the time line by different CNs.

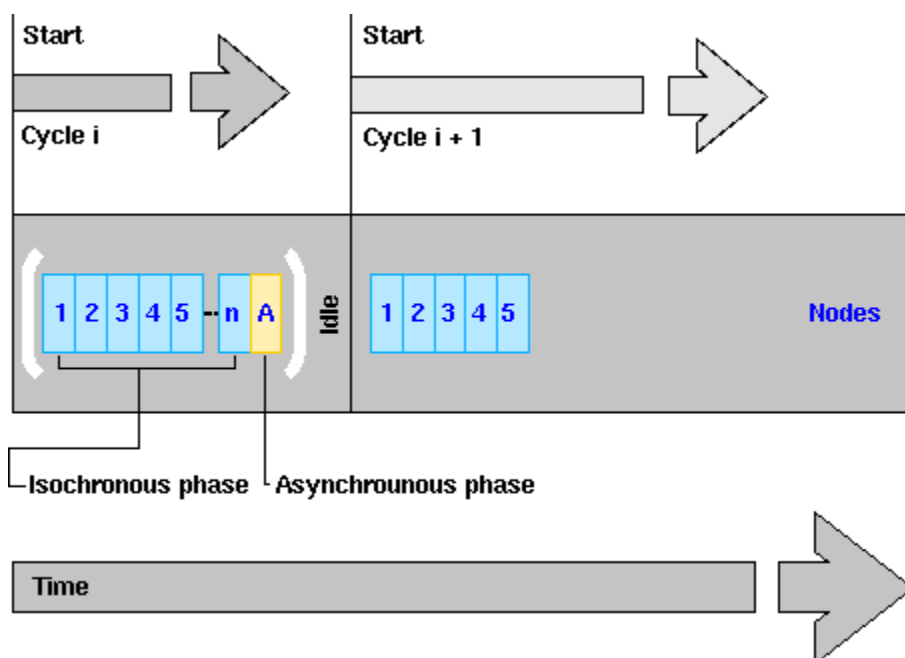


Figure 45. Time slots for nodes and the asynchronous time slot

4.2.1.6. Multiplex for Bandwidth Optimization

In addition to transferring isochronous data during each basic cycle, some nodes are also able to share transfer slots for better bandwidth utilization. For that reason, the isochronous phase can distinguish between transfer slots dedicated to particular nodes, which have to send their data in every basic cycle, and slots shared by nodes to transfer their data one after the other in different cycles. Therefore less important yet still time-critical data can be transferred in longer cycles than the basic cycle. Assigning the slots during each cycle is at the discretion of the Managing Node.

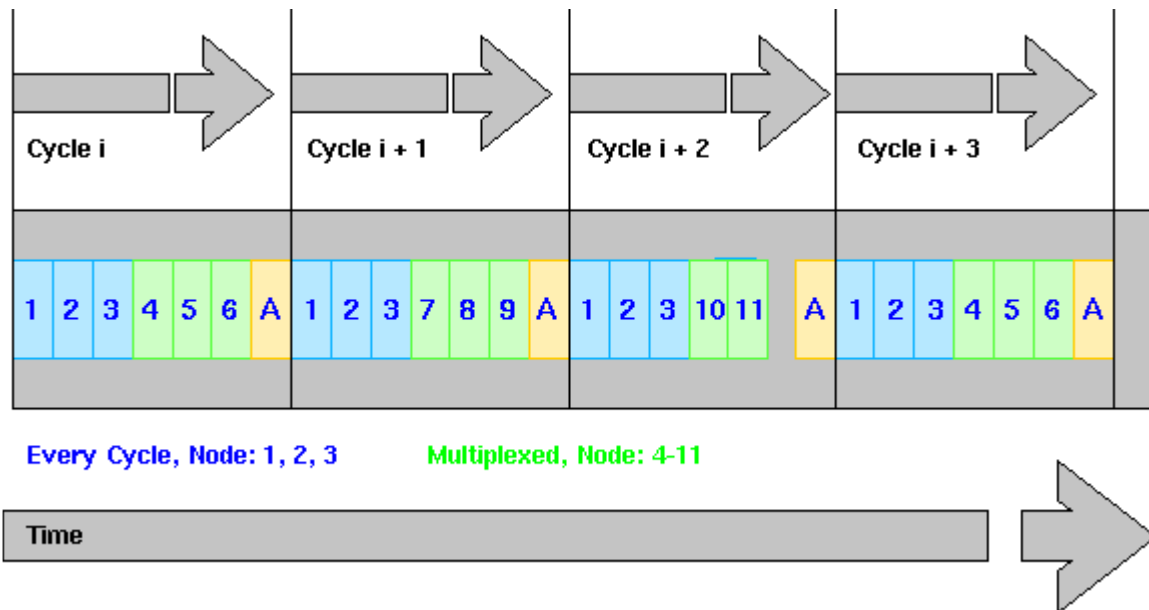


Figure 46. Time slots in EPL multiplexed mode.

4.2.1.7. Ethernet Powerlink Safety

Today, machines, plants and safety systems are stuck in a rigid scheme made up of hardware-based safety functions. The consequences of this are cost-intensive cabling and limited diagnostic options. The solution is the integration of safety relevant application data into the standard serial control protocol. Ethernet Powerlink Safety (EPLsafety). EPLsafety allows both publish/subscriber and client/server communication. Safety relevant data is transmitted via an embedded data frame inside of standard communication messages. Measures to avoid any undetected failures due to systematic or stochastic errors are an integral part of the security protocol. EPLsafety is in conformance with IEC 61508. The protocol fulfills the requirements of SIL 3. Error detection techniques have no impact on existing transport layers.

4.2.2. Profinet

PROFINET uses TCP/IP and IT standards, and is, in effect, real-time Ethernet.

The PROFINET concept features a modular structure so that users can select the cascading functions themselves.

In conjunction with PROFINET, the two perspectives PROFINET CBA and PROFINET IO exist. PROFINET CBA is suitable for the component-based communication via TCP/IP and the real-time communication for real-time requirements in modular systems engineering. Both communication options can be used in parallel.

PROFINET IO was developed for real time (RT) and isochronous real time (IRT) communication with the decentral periphery. The designations RT and IRT merely describe the real-time properties for the communication within PROFINET IO.

PROFINET CBA and PROFINET IO can communicate at the same time on the same bus system. They can be operated separately or combined so that a PROFINET IO subsystem appears as a PROFINET CBA system from a system perspective.

PROFINET is the innovative open standard for Industrial Ethernet, development by Siemens and the Profibus User Organization (PNO). With PROFINET, solutions can be implemented for factory and process automation, for safety applications, and for the entire range of drive technology right up to clock-synchronized motion control. PROFINET is standardized in IEC 61158 and IEC 61784. Profinet products are certified by the PNO user organization, guaranteeing worldwide compatibility.

PROFINET is based on Ethernet and uses TCP/IP and IT standards and complements them with specific protocols and mechanisms to archive a good Real Time performance. PROFINET enables the integration of existing Fieldbus systems like PROFIBUS, DeviceNet, and Interbus, without changes to existing devices. That means that the investments of plant operators, machine and system builders, and device manufacturers are protected.

4.2.2.1. Technology

To achieve these functions, three different protocol levels are defined:

- TCP/IP for PROFINET CBA and the commissioning of a plant with reaction times in the range of 100ms
- RT (Real-Time) protocol for PROFINET CBA and PROFINET IO applications up to 10 ms cycle times
- IRT (Isochronous Real-Time) for PROFINET IO applications in drive systems with cycles times of less than 1ms

The PROFINET protocol can be recorded and displayed using any Ethernet analysis tool. In the current versions, Wireshark/Ethereal is able to decode PROFINET message frames.

PROFINET offers scalable performance with three performance levels:

- TCP/IP: for non Real Time applications
- Real Time (RT): for Real Time transfer of time-critical process data
- Isochronous Real Time (IRT): for motion control applications

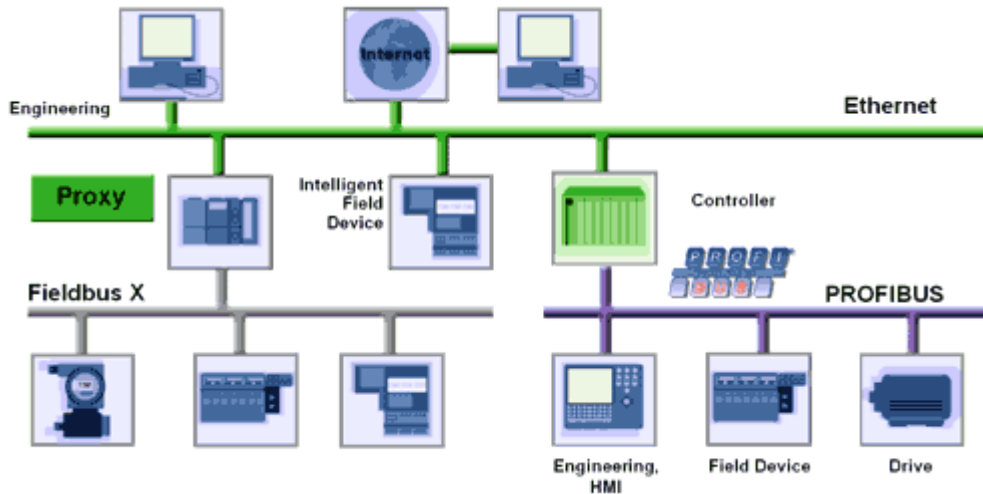


Figure 47. PROFINET Configuration

4.2.2.2. PROFINET component model (PROFINET CBA)

A PROFINET CBA system consists of various automation components. One component covers all mechanical, electrical and IT variables. The component can be generated using the standard programming tools. A component is described using a PROFINET Component Description (PCD) file in XML. A planning tool loads these descriptions and enables the logical interconnections between the individual components to be generated for implementing a plant.

The basic idea of CBA is that an entire automation system can, in many cases, be divided into autonomously operating subsystems, thereby arranging them very clearly. The design and the functions can actually end up in identical or slightly modified form in several systems. These PROFINET components are usually controlled by a manageable number of input signals. Within the component, a control program written by the user executes the required function and passes the corresponding output signals to another controller. The engineering that is associated with it is manufacturer-neutral. Communication with PROFINET CBA (without real time) is suitable for bus cycle times of approx. 50 ... 100 ms. The parallel running RT channel allows for data cycles similar to PROFINET IO (a few ms).

4.2.2.3. PROFINET and the peripherals (PROFINET IO)

Interfacing the peripherals is implemented by PROFINET IO. It defines the communication with field connected peripheral devices. Its basis is a cascading real-time concept. PROFINET IO defines the entire data exchange between controllers (devices with "master functionality") and the devices (devices with "slave functionality"), as well as parameter setting and diagnosis. PROFINET IO is designed for fast data exchange between Ethernet-based field devices and follows the provider-consumer model. Field devices in a subordinate PROFIBUS line can be integrated in the PROFINET IO system without any effort and seamlessly via an IO-Proxy (representative of a subordinate bus system). A device developer can implement PROFINET IO with any commercially available Ethernet controller. It is well-suited for

data exchange with bus cycle times of a few ms. The configuration of an IO-System has been kept nearly identical to the "look and feel" of PROFIBUS. PROFINET IO always contains the real-time concept.

A PROFINET IO system consists of the following devices:

- The IO Controller, which controls the automation task.
- The IO Device, which is a field device, monitored and controlled by an IO Controller. An IO Device may consist of several modules and sub-modules.
- The IO Supervisor is software typically based on a PC for setting parameters and diagnosing individual IO Devices.

An Application Relation (AR) is established between an IO Controller and an IO Device. These ARs are used to define Communication Relations (CR) with different characteristics for the transfer of parameters, cyclic exchange of data and handling of alarms.

The characteristics of an IO Device are described by the device manufacturer in a General Station Description (GSD) file. The language used for this purpose is the GSDML (GSD Markup Language) - an XML based language. The GSD file provides the supervision software with a basis for planning the configuration of a PROFINET IO system.

4.2.2.4. Configuration with GSD-Files

During the setup phase of the PROFINET network, the Profinet Controller must be configured with a special configuration tool such as Step7 from Siemens. The configuration process is based on electronic device data sheets (GSD-Files) which are required for each Profinet device. GSD-Files are provided by the device manufacturer and contain electronic descriptions of all relevant communication parameters of the Profinet device. Table 11 shows some basic features of Profinet.

Table 11. Profinet facts

PROFINET FACTS	
Network Type	Scalable Ethernet based advanced communication system
Topology	Very flexible with Line, Bus, Star or Tree topology depending on physical media
Installation	Switched Ethernet transmission with shielded twisted pair cables and RJ45 or M12 connectors. Alternatively Fiber Optic Media.
Data Rate	100 Mbit/s
max. n ^o . of stations	Almost unlimited
Data	Each node: up to 1.500 bytes per telegram frame Total: nearly unlimited
Network Features	Scalable communication system based on switched Fast Ethernet technology with good real time performance and advanced configuration, diagnosis and alarm handling.
User Organization	Profibus International



4.2.2.5. PROFINET and Real Time

Within PROFINET IO, process data and alarms are always transmitted in real time (RT). Real time in PROFINET is based on the definition of IEEE and IEC, which allow for only a limited time for execution of real-time services within a bus cycle. RT communication represents the basis for data exchange for PROFINET IO. Real-time data are treated with a higher priority than TCP (UDP)/IP data. RT provides the basis for real-time communication in the area of distributed periphery and for the PROFINET component model (PROFINET CBA). This type of data exchange allows bus cycle times in the range of a few hundred microseconds.

4.2.2.6. PROFINET and Isochronous Communication

Isochronous data exchange with PROFINET is defined in the isochronous real-time (IRT) concept. PROFINET IO field devices with IRT functionality have switch ports integrated in the field device. They can be based e.g. on the Ethernet controllers ERTEC 400/200. The data exchange cycles are usually in the range of a few hundred microseconds up to a few milliseconds. The difference to real-time communication is essentially the high degree of determinism, so that the start of a bus cycle is maintained with high precision. The start of a bus cycle can deviate up to 1 μ s (jitter). IRT is required, for example, for motion control applications (positioning control processes).

4.2.2.7. Profiles

Profiles are pre-defined configurations of the functions and features available from PROFINET for use in specific devices or applications. They are specified by PI working groups and published by PI. Profiles are important for openness, interoperability and interchangeability, so that the end user can be sure that similar equipment from different vendors perform in a standardised way. User choice encourages competition that drives vendors towards enhanced performance and lower costs.

There are PROFINET profiles for encoders, for example. Other profiles have been specified for motion control (PROFIdrive) and functional safety (PROFIsafe). A special profile for trains also exists.

An important profile is PROFInergy. This was requested in 2009 by the AIDA group of German automotive manufacturers (Audi, BMW, Mercedes, Porsche and VW) who wished to have a standardised way of actively managing energy usage in their plants. High energy devices and sub-systems such as robots, lasers and even paint lines are the target for this profile, which will help reduce a plant's energy costs by intelligently switching the devices into 'sleep' modes to take account of production breaks, both foreseen (e.g. weekends and shut-downs) and unforeseen (e.g. breakdowns).

PROFInergy is applicable across industry and includes monitoring services that can lead to the real time management of energy demand

4.2.2.8. Additional Highlights of the PROFINET Concept

Engineering: By supporting the Tool Calling Interface, every manufacturer of field devices can latch onto any TCI-capable software and parameterize and diagnose field devices without having to leave the program.

Proximity recognition and device replacement: All PROFINET field devices determine their neighbours. This allows replacing field devices without additional tools and prior knowledge in case of a fault. By reading this information, the system topology can be graphically represented in a very clear way.

Parameter server: Individually set data can be loaded manufacturer-neutral (e.g. via TCI) and automatically archived in a parameter server. Reloading is also done automatically when replacing a device.

Determinism: PROFINET supports deterministic data traffic, for example, for high-precision control tasks.

Redundancy: The redundancy concept defined in PROFINET significantly increases system availability.

4.2.2.9. Benefits of PROFINET

Due to the continuous further development of PROFINET, users are provided with a long-term perspective for the implementation of their automation tasks.

The system operator profits from the simple expandability of the system and high degree of availability from autonomously running subsystems.

4.3. Isochronous Real Time Applications

4.3.1. Sercos

4.3.1.1. Introduction

In the field of Industrial Control Systems, the interfacing of various control components must provide a means to coordinate the signals and commands sent between control modules. While tight coordination is desirable for discrete inputs and outputs, it is especially important in motion controls, where directing the movement of individual axis of motion must be precisely coordinated so that the motion of the entire system follows a desired path. Types of equipment requiring such coordination are for example metal cutting machine tools, metal forming equipment, assembly machinery, packaging machinery, robotics, printing machinery and material handling equipment. The **SERCOS (SERial Real-time COmmunication System)** interface is a globally standardized open digital interface for communication between industrial controls, motion devices (drives) and input output devices (I/O). It is classified as standard IEC 61491 and EN 61491. The SERCOS interface is designed to provide hard real-time, high performance communications between industrial motion controls and digital servo drives.

4.3.1.2. History

Until the early 1980's the majority of servo drive systems used to control motion in industrial machinery were based upon analog electronics. The accepted interface to control such devices was an analog voltage signal, where polarity represented the desired direction of motion, and magnitude represented the desired speed or torque. In the 1980s, drive systems and devices based on digital technology began to emerge. A new method needed to be devised to communicate with, and control, such units, as their capabilities could not be exploited with the traditional interface method used with analog drives. The earliest interfaces were either proprietary to one vendor or designed only for a single purpose, making it difficult for users of motion control systems to freely interchange motion control and drives. The members of the VDW (German Machine Tool Builders' Association) became concerned with the implications of this trend. In response to that, in 1987 the VDW formed a joint working group with the ZVEI (German Electrical and Electronics Industry Association) to develop an open interface specification that was appropriate for digital drive systems. The resulting specification, entitled "SERCOS (**S**ERIAL **R**eal-time **C**OMMUNICATION **S**ystem) interface, was released and later submitted to the IEC, which in 1995 released it as IEC 61491. After the release of the original standard, some of the original working group member companies (including ABB, AEG, AMK, Robert Bosch, Indramat, and Siemens) founded the "Interest Group SERCOS" to steward the standard. Over the history of the SERCOS interface, its capabilities have been enhanced to the point where today it is not only used for motion control systems, but also for the control of I/O on machinery, as a single machine network.

4.3.1.3. Versions

SERCOS-I was released in 1991. The transmission medium used is optical fiber. The data rates supported are 2 and 4 MBit/s, and cyclic update rates as low as 62.5 microseconds. A ring topology is used. SERCOS-I also supports a "Service Channel" which allows asynchronous communication with slaves for less time-critical data.

SERCOS-II was introduced in 1999. It expanded the data rates supported to 2, 4, 8 and 16 MBit/s.

SERCOS-III merges the hard-real-time aspects of the SERCOS interface with the Ethernet standard.

4.3.1.4. SERCOS Interface Features

Important features of the SERCOS interface include:

- Collision-free communication through the use of a time-slot mechanism.
- Highly efficient communication protocol
- Extremely low telegram jitter (specified at less than 1 microsecond, in practice as low as 35 nanoseconds).

- Highly developed standardized profiles agreed upon by multi-vendor technical working groups for dependable interoperability of devices from different manufacturers.
- Ability to control, for example, 70 axes of motion at an update of 250 microseconds for each and every drive (SERCOS-III)

4.3.2. Ethercat

The EtherCAT technology overcomes the system limitations of other Ethernet solutions: The Ethernet packet is no longer received, then interpreted and copied as process data at every connection. Instead, the Ethernet frame is processed on the fly (see below): the newly developed FMMU (Fieldbus Memory Management Unit) in each slave node reads the data addressed to it, while the telegram is forwarded to the next device. Similarly, input data is inserted while the telegram passes through. The telegrams are only delayed by a few nanoseconds.

On the master side, very inexpensive, commercially available standard network interface cards (NIC) or any on board Ethernet controller can be used as hardware interface. The common feature of these interfaces is data transfer to the PC via DMA (direct memory access), i.e. no CPU capacity is taken up for the network access.

4.3.2.1. Introduction

Typical automation networks are characterized by short data length per node, typically less than the minimum payload of an Ethernet frame. Using one frame per node per cycle therefore leads to low bandwidth utilization and thus to poor overall network performance. EtherCAT therefore takes a different approach, called processing on the fly.

4.3.2.2. Functional principle

With EtherCAT, the Ethernet packet or frame is no longer received, then interpreted and copied as process data at every node. The EtherCAT slave devices read the data addressed to them while the telegram passes through the device. Similarly, input data are inserted while the telegram passes through. The frames are only delayed by a fraction of a microsecond in each node, and many nodes - typically the entire network - can be addressed with just one frame.

4.3.2.3. Protocol

The EtherCAT protocol is optimised for process data and is transported directly within the standard IEEE 802.3 Ethernet frame using Ethertype 0x88a4. It may consist of several sub-datagrams, each serving a particular memory area of the logical process images that can be up to 4 gigabytes in size. The data sequence is independent of the physical order of the nodes in the network; addressing can be in any order. Broadcast, multicast and communication between slaves are possible and must be done by the master device. If IP routing is required, the EtherCAT protocol can be inserted into



UDP/IP datagrams. This also enables any control with Ethernet protocol stack to address EtherCAT systems.

4.3.2.4. Performance

Short cycle times can be achieved since the host microprocessors in the slave devices are not involved in the processing of the Ethernet packets to transfer the process images. All process data communication is handled in the slave controller hardware. Combined with the functional principle this makes EtherCAT a high performance distributed I/O system: Process data exchange with 1000 distributed digital I/O takes about 30 μ s, which is typical for a transfer of 125 bytes over 100Mb/s Ethernet. Data for and from 100 servo axis can be updated with up to 10 kHz. Typical network update rates are 1–30 kHz, but EtherCAT can be used with slower cycle times, too, if the DMA load is too high for the PC being used.

4.3.2.5. Topology

Line, tree or star: EtherCAT supports almost any topology. The bus or line structure known from the fieldbusses thus also becomes available for Ethernet. Particularly useful for system wiring is the combination of line and branches or stubs: The required interfaces exist on the couplers; no additional switches are required. Naturally, the classic switch-based Ethernet star topology can also be used.

Wiring flexibility is further maximized through the choice of different cables. Flexible and inexpensive standard Ethernet patch cables transfer the signals optionally in Ethernet mode (100BASE-TX) or in E-Bus (LVDS) signal representation. Plastic optical fibre (POF) can be used in special applications for longer distances. The complete bandwidth of the Ethernet network - such as different fiber optics and copper cables - can be used in combination with switches or media converters.

Fast Ethernet (100BASE-FX) or E-Bus can be selected based on distance requirements. The Fast Ethernet physics enables a cable length of 100 m between devices while the E-Bus line is intended for modular devices. The size of the network is almost unlimited since up to 65535 devices can be connected.

Using full-duplex Ethernet physical layers, the EtherCAT slave controllers close an open port automatically and return the Ethernet frame if no downstream device is detected. Slave devices may have two or more ports.

4.3.2.6. EtherCAT instead of PCI

The central PC becomes smaller and more cost effective because additional slots are not needed for interface cards since the onboard Ethernet port can be used. With increasing miniaturisation of PC components, the physical size of industrial PCs is increasingly determined by the number of required slots. The bandwidth of Fast Ethernet, together with the data width of the EtherCAT communication hardware enables new directions: interfaces that are conventionally located in the IPC are transferred to intelligent interface terminals at the EtherCAT. Apart from the

decentralised I/Os, axes and control units, complex systems such as fieldbus masters, fast serial interfaces, gateways and other communication interfaces can be addressed. Ethernet devices without restriction on protocol variants can be connected via decentralised "hub terminals". The central IPC becomes smaller and therefore more cost-effective, one Ethernet interface is sufficient for complete communication with the periphery.

4.3.2.7. Synchronization

For synchronization, a distributed clock mechanism is applied, which leads to very low jitters of significantly less than 1 μ s even if the communication cycle jitters. This is equivalent to the IEEE 1588 Precision Time Protocol standard. Therefore EtherCAT does not require any special hardware in the master device and can be implemented in software on any standard Ethernet MAC, even without a dedicated communication coprocessor.

The typical process of establishing a distributed clock is initiated by the master by sending a broadcast to all slaves to a certain address. Upon reception of this message, all slaves will latch the value of their internal clock twice, once when the message is received and once when it returns (remember EtherCAT has a ring topology). The master can then read all latched values and calculate the delay for each slave. This process can be repeated as many times as required to reduce jitter and average out values. Total delays are calculated for each slave depending on its position in the slave-ring and will be uploaded to an offset register. Finally the master issues a broadcast readwrite on the system clock, which will make the first slave the reference clock and forcing all other slaves to set their internal clock appropriately with the now known offset.

To keep the clocks synchronised after initialization, the master or slave must regularly send out the broadcast again to counter any effects of speed difference between the internal clocks of each slave. Each slave should adjust the speed of its internal clock or implement an internal correction mechanism whenever it has to adjust.

The system clock is specified as a 64 bit counter with a base unit of 1 ns starting at January 1, 2000, 0:00.

4.3.2.8. Distributed Clock

Accurate synchronization is particularly important in cases where widely distributed processes require simultaneous actions. This may be the case, for example, in applications where several servo axes carry out coordinated movements simultaneously. The most powerful approach for synchronization is the accurate alignment of distributed clocks. In contrast to fully synchronous communication, where synchronization quality suffers immediately in the event of a communication fault, distributed aligned clocks have a high degree of tolerance from possible fault-related delays within the communication system.



With EtherCAT, data exchange is completely hardware based on "mother" and "daughter" clocks. Each clock can simply and accurately determine the other clocks run-time offset because the communication utilizes a logical and full-duplex Ethernet physical ring structure. The distributed clocks are adjusted based on this value, which means that a very precise network-wide timebase with a jitter of significantly less than 1 microsecond is available.

However, high-resolution distributed clocks are not only used for synchronization. They can also provide accurate information about the local timing of the data acquisition. For example, controls frequently calculate velocities from sequentially measured positions. Particularly with very short sampling times, even a small temporal jitter in the displacement measurement leads to large step changes in velocity. With EtherCAT new, expanded data types (timestamp data type, oversampling data type) are introduced. The local time is linked to the measured value with a resolution of up to 10 ns, which is made possible by the large bandwidth offered by Ethernet. The accuracy of a velocity calculation then no longer depends on the jitter of the communication system. It is orders of magnitude better than that of measuring techniques based on jitter-free communication.

4.3.2.9. Hot Connect

The Hot Connect function enables parts of the network to be linked and decoupled or reconfigured "on the fly". Many applications require a change in I/O configuration during operation. Examples are processing centres with changing, sensor-equipped tool systems or transfer devices with intelligent, flexible work piece carriers. The protocol structure of the EtherCAT system takes account of these changing configurations.

4.3.2.10. Safety over EtherCAT

Conventionally, safety functions are realized separately from the automation network, either via hardware or using dedicated safety bus systems. Safety over EtherCAT enables safety-related communication and control communication on the same network. The safety protocol is based on the application layer of EtherCAT, without influencing the lower layers. It is certified according to IEC 61508 and meets the requirements of Safety Integrated Level (SIL) 3. The data length is variable, making the protocol equally suitable for safe I/O data and for safe drive technology. Like other EtherCAT data, the safety data can be routed without requiring safety routers or gateways. The first fully certified products featuring Safety over EtherCAT are already available. Safety over EtherCAT protocol is referred to as FSCP 12 (Functional Safety Communication Profile) in the international standard IEC 61784-3.

4.3.2.11. Openness

The EtherCAT technology is fully Ethernet-compatible and truly open. The protocol tolerates other Ethernet-based services and protocols on the same physical network - usually even with minimum loss of performance. There is no restriction on the type of Ethernet device that can be connected within the EtherCAT segment via a hub

terminal. Devices with fieldbus interface are integrated via EtherCAT fieldbus master terminals. The UDP protocol variant can be implemented on each socket interface. The EtherCAT Technology Group ensures that each interested party can implement and use this network. EtherCAT is an international standard.

4.3.2.12. Device Profiles

The device profiles describe the application parameters and the functional behaviour of the devices including the device class-specific state machines. For many device classes, fieldbus technology already offers reliable device profiles, for example for I/O devices, drives or valves. EtherCAT supports both the CANopen device profile family as well as the drive profile known as the Sercos drive profile. Since the application view does not change when migrating from CANopen or Sercos, this assists users and device manufacturers alike.

4.3.2.13. Gateways

For integration of existing fieldbus components (e.g., CANopen, DeviceNet, Profibus) into EtherCAT networks gateway devices are available. Also other Ethernet protocols can be used in conjunction with EtherCAT: The Ethernet frames are tunneled via the EtherCAT protocol, which is the standard approach for internet applications (e.g. VPN, PPPoE (DSL) etc.). The EtherCAT network is fully transparent for the Ethernet device, and the real-time characteristics are not impaired since the master dictates exactly when the tunneled transfers are to occur and how much capacity of the 100Mb/s media the tunneled protocols can use. All internet technologies can, therefore, also be used in the EtherCAT environment: integrated web server, e-mail, FTP transfer etc.

4.3.2.14. Implementation

The Master can be implemented in software on any standard Ethernet MAC. Several vendors supply code for different operating systems. There are also several open and shared source implementations. For slave devices special EtherCAT slave controller chips are required in order to perform the "processing on the fly" principle. EtherCAT slave controllers are available as code for different FPGA types and are also available as ASIC implementations.

5. Contents of The CML Box

The CML Box consists of many industrial devices. These are PLCs, switches, a NAT router, an industrial simulator and an industrial PC. In this chapter we are going to give a brief explanation of these devices.

5.1. PLC and Distributed I/Os

In this CML Box we used one PLC (Phoenix Contact ILC 150 ETH) and three distributed I/Os (IL ETH BK DI8 DO4 2TX-PAC) to illustrate a real industrial automation system.



Figure 48. PLC & Distributed I/Os

These devices use Modbus to communicate, each distributed I/O has 4 digital outputs, 8 digital inputs, an analog input and an analog output. These devices are connected to the SABO industrial simulator system.

5.2. SABO Simulator

Sabo is used for illustrating a real industrial system. It has a touch screen which lets the user change the operation of the process. The Simulator gets its inputs from a distributed I/O and its outputs are also connected to a distributed I/O, so it is possible to investigate the travel path and also the ingredient of the packets flowing through the Ethernet System.

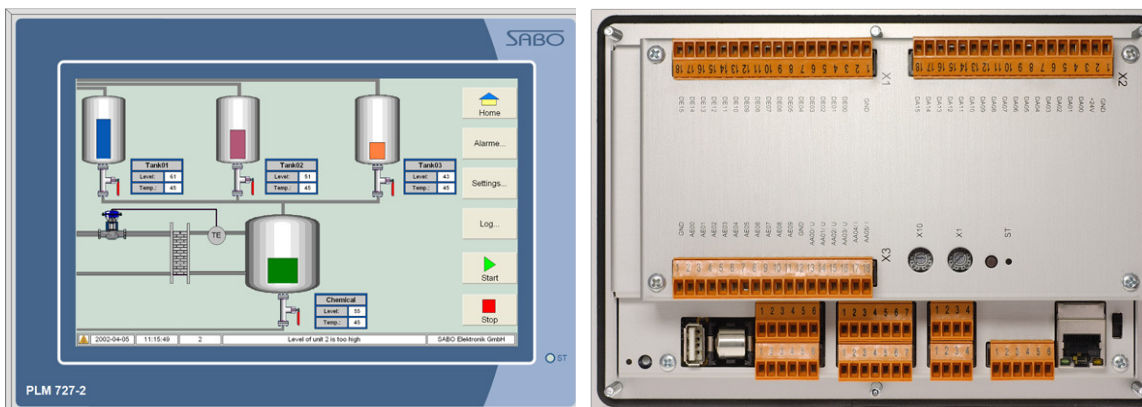


Figure 49. SABO Industrial Simulator

5.3. Manageable Switches

In CML Box 2 Moxa manageable switches are used. Manageable switches give us much more opportunities rather than a standard office switch/hub. The most characteristic features of these switches are:

- 1- In the case of a cable disconnection (on the ports which are supposed to be connected) it can send us e-mail or create an alarm event.
- 2- These kinds of switches also let us record statistical data to be viewed by the user or an administrator.
- 3- It is also possible to copy all the flow data to one external port which is called port-mirroring. With this powerful feature it is possible to capture all flow data and investigate the packets.
- 4- An internal web server is included which gives us the opportunity to change settings and view the status of the device.
- 5- This device has two supply pins. In case of power loss, it can continue working with the surge power.



Figure 50. Moxa Manageable Switch

5.4. NAT Router

Phoenix Contact NAT router is an industrial device. It has one WAN port and many LAN ports. The main concept of the device is address translating.

For example we can setup an internal (local) network with IPs of “192.168.127.1 – 192.168.127.10” so we have 10 computers over our LAN. Also we have a WAN with IPs of 10.0.0.XXX It is possible to match these IPs one by one. For example, let’s say that we set the device to translate “192.168.127.1 – 192.168.127.10” to “10.0.0.11 – 10.0.0.20” Here the most important point is that this matching is not symmetric. Suppose that we have a server on WAN with the IP “10.0.0.1” we can access it from the LAN with the IP “10.0.0.1” (same) but when the server on the WAN accesses a device on the LAN, for example the device with IP “192.168.127.1”, it uses “10.0.0.11”.



Figure 51. PxC NAT Router

5.5. Rail PC

Advantech Rail-PC is also another industrial device. It is built to work stand alone (without keyboard, mouse, monitor) This PC can operate under different conditions that cannot be accepted in an office environment. It also has 2 RJ45 Ethernet ports, many USBs and RS-232 serial communication ports. So it is possible to carry out many different configurations with this device.

Here it is planned to use a third-party remote desktop software (VNC) to access and control the device.



Figure 52. Advantech Rail PC

5.6. Wireshark

Wireshark is another third party software. It lets us capture flowing packets on an Ethernet device. So we can investigate the packets and the condition of the network. In the figure below we can see what happens when we send four “ping”s to a device. It is also possible to view each packet byte by byte with this software.

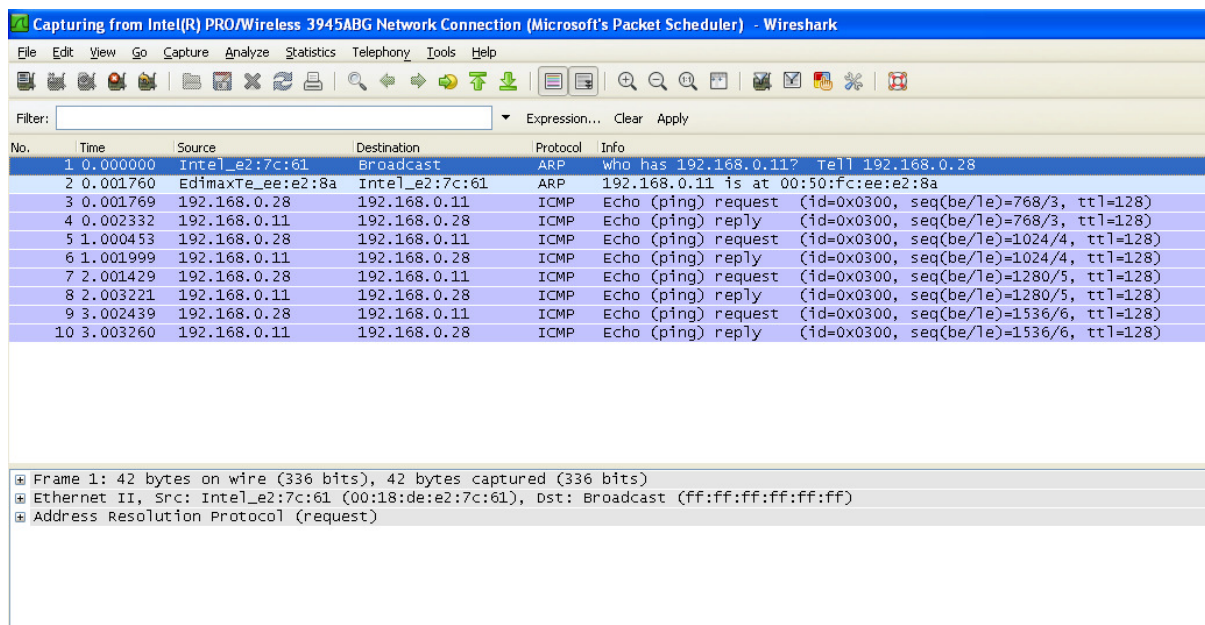


Figure 53. Wireshark

6. Cables and Topology

6.1. Unshielded Twisted Pair (UTP) Cable

Twisted pair cabling comes in two varieties: shielded and unshielded. Unshielded twisted pair (UTP) is the most popular and is generally the best option for school networks (See figure 54).

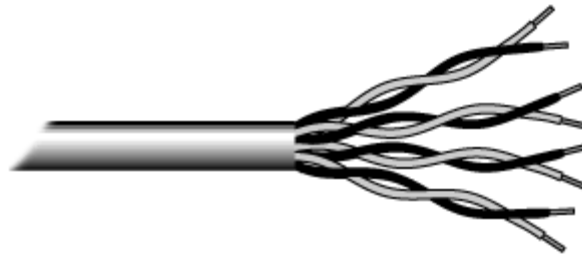


Figure 54. Unshielded twisted pair

The quality of UTP may vary from telephone-grade wire to extremely high-speed cable. The cable has four pairs of wires inside the jacket. Each pair is twisted with a different number of twists per inch to help eliminate interference from adjacent pairs and other electrical devices. The tighter the twisting is, the higher the supported transmission rate and the greater the cost per foot. The EIA/TIA (Electronic Industry Association/Telecommunication Industry Association) has established standards of UTP and rated six categories of wire (additional categories are emerging).

Table 12. Categories of Unshielded Twisted Pair

Category	Speed	Use
1	1 Mbps	Voice Only (Telephone Wire)
2	4 Mbps	LocalTalk & Telephone (Rarely used)
3	16 Mbps	10BaseT Ethernet
4	20 Mbps	Token Ring (Rarely used)
5	100 Mbps (2 pair)	100BaseT Ethernet
5e	1000 Mbps (4 pair)	Gigabit Ethernet
6	10,000 Mbps	Gigabit Ethernet

6.1.1. Unshielded Twisted Pair Connector

The standard connector for unshielded twisted pair cabling is an RJ-45 connector. This is a plastic connector that looks like a large telephone-style connector (See figure 55). A slot allows the RJ-45 to be inserted only one way. RJ stands for Registered Jack, implying that the connector follows a standard borrowed from the telephone industry. This standard designates which wire goes with each pin inside the connector.

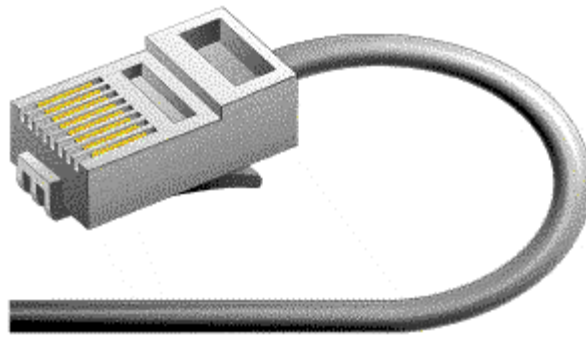


Figure 55. RJ-45 connector

6.2. Shielded Twisted Pair (STP) Cable

Although UTP cable is the least expensive cable, it may be susceptible to radio and electrical frequency interference (it should not be too close to electric motors, fluorescent lights, etc.). If you must place cable in environments with lots of potential interference, or if you must place cable in extremely sensitive environments that may be susceptible to the electrical current in the UTP, shielded twisted pair may be the solution. Shielded cables can also help to extend the maximum distance of the cables.

Shielded twisted pair cable is available in three different configurations:

1. Each pair of wires is individually shielded with foil.
2. There is a foil or braid shield inside the jacket covering all wires (as a group).
3. There is a shield around each individual pair, as well as around the entire group of wires (referred to as double shield twisted pair).

6.3. Coaxial Cable

Coaxial cabling has a single copper conductor at its center. A plastic layer provides insulation between the center conductor and a braided metal shield (See figure 56). The metal shield helps to block any outside interference.

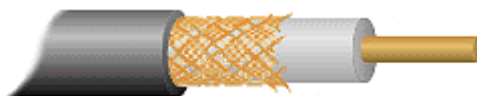


Figure 56. Coaxial cable

Although coaxial cabling is difficult to install, it is highly resistant to signal interference. In addition, it can support greater cable lengths between network devices than twisted pair cable.

Thin coaxial cable is also referred to as thinnet. 10Base2 refers to the specifications for thin coaxial cable carrying Ethernet signals. The 2 refers to the approximate maximum segment length being 200 meters. In actual fact the maximum segment length is 185 meters.

Thick coaxial cable is also referred to as thicknet. 10Base5 refers to the specifications for thick coaxial cable carrying Ethernet signals. The 5 refers to the maximum segment

length being 500 meters. Thick coaxial cable has an extra protective plastic cover that helps keep moisture away from the center conductor. This makes thick coaxial a great choice when running longer lengths in a linear bus network. One disadvantage of thick coaxial is that it does.

6.3.1. Coaxial Cable Connectors

The most common type of connector used with coaxial cables is the Bayone-Neill-Concelman (BNC) connector (See figure 57). Different types of adapters are available for BNC connectors, including a T-connector, barrel connector, and terminator. Connectors on the cable are the weakest points in any network. To help avoid problems with your network, always use the BNC connectors that crimp, rather.

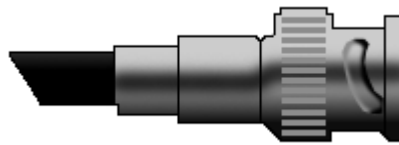


Figure 57. BNC connector

6.4. Fiber Optic Cable

Fiber optic cabling consists of a center glass core surrounded by several layers of protective materials (See figure 58). It transmits light rather than electronic signals eliminating the problem of electrical interference. This makes it ideal for certain environments that contain a large amount of electrical interference. It has also made it the standard for connecting networks between

Fiber optic cable has the ability to transmit signals over much longer distances than coaxial and twisted pair. It also has the capability to carry information at vastly greater speeds. This capacity broadens communication possibilities to include services such as video conferencing and interactive services. The cost of fiber optic cabling is comparable to copper cabling; however, it is

The center core of fiber cables is made from glass or plastic fibers (see figure 58). A plastic coating then cushions the fiber center, and kevlar fibers help to strengthen the cables and prevent breakage. The outer insulating jacket made of teflon or PVC.

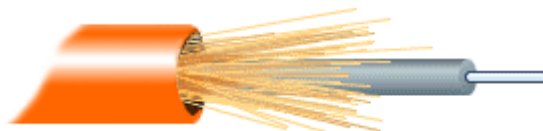


Figure 58. Fiber optic cable

There are two common types of fiber cables -- single mode and multimode. Multimode cable has a larger diameter; however, both cables provide high bandwidth at high speeds. Single mode can provide more distance, but it is more expensive.

Table 13. Ethernet Cable Summary

Specification	Cable Type
10BaseT	Unshielded Twisted Pair
10Base2	Thin Coaxial
10Base5	Thick Coaxial
100BaseT	Unshielded Twisted Pair
100BaseFX	Fiber Optic
100BaseBX	Single mode Fiber
100BaseSX	Multimode Fiber
1000BaseT	Unshielded Twisted Pair
1000BaseFX	Fiber Optic
1000BaseBX	Single mode Fiber
1000BaseSX	Multimode Fiber

6.5. What is a Topology?

The physical topology of a network refers to the configuration of cables, computers, and other peripherals. Physical topology should not be confused with logical topology which is the method used to pass information between workstations.

6.5.1. Main Types of Physical Topologies

The following sections discuss the physical topologies used in networks and other related topics.

- Linear Bus
- Star
- Tree (Expanded Star)
- Considerations When Choosing a Topology
- Summary Chart

6.5.1.1. Linear Bus

A linear bus topology consists of a main run of cable with a terminator at each end (See figure 59). All nodes (file server, workstations, and peripherals) are connected to the linear cable.

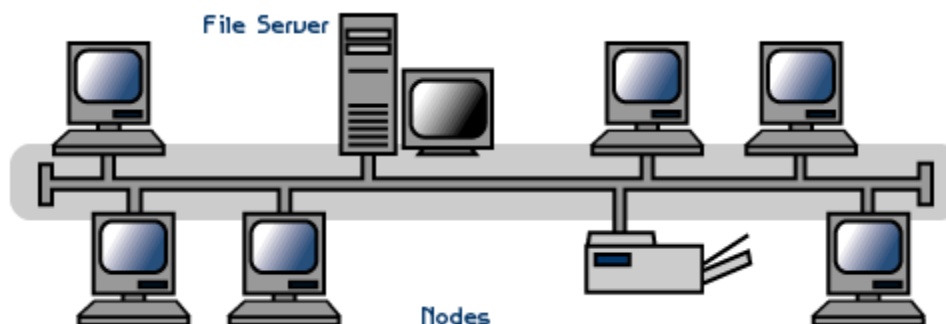


Figure 59. Linear Bus topology

Advantages of a Linear Bus Topology

- Easy to connect a computer or peripheral to a linear bus.
- Requires less cable length than a star topology.

Disadvantages of a Linear Bus Topology

- Entire network shuts down if there is a break in the main cable.
- Terminators are required at both ends of the backbone cable.
- Difficult to identify the problem if the entire network shuts down.
- Not meant to be used as a stand-alone solution in a large building.

6.5.1.2. Star

A star topology is designed with each node (file server, workstations, and peripherals) connected directly to a central network hub, switch, or concentrator (See figure 60).

Data on a star network passes through the hub, switch, or concentrator before continuing to its destination. The hub, switch, or concentrator manages and controls all functions of the network. It also acts as a repeater for the data flow. This configuration is common with twisted pair cable; however, it can also be used with coaxial cable or fiber optic cable.

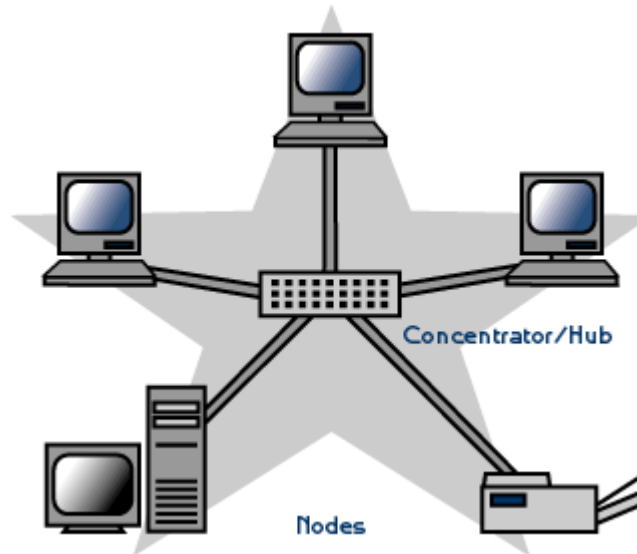


Figure 60. Star topology

Advantages of a Star Topology

- Easy to install and wire.
- No disruptions to the network when connecting or removing devices.
- Easy to detect faults and to remove parts.

Disadvantages of a Star Topology

- Requires more cable length than a linear topology.
- If the hub, switch, or concentrator fails, nodes attached are disabled.
- More expensive than linear bus topologies because of the cost of the hubs, etc.

6.5.1.3. Tree or Expanded Star

A tree topology combines characteristics of linear bus and star topologies. It consists of groups of star-configured workstations connected to a linear bus backbone cable (See figure 61). Tree topologies allow for the expansion of an existing network, and enable schools to configure a network to meet their needs.

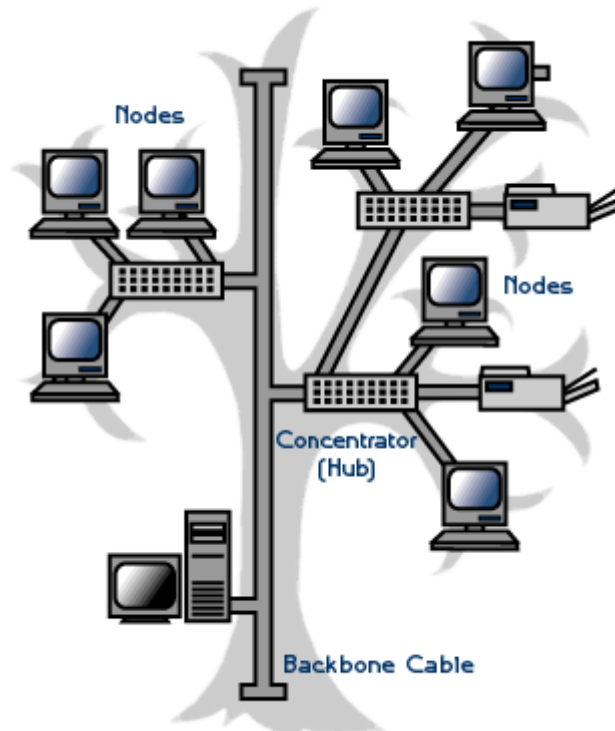


Figure 61. Tree topology

Advantages of a Tree Topology

- Point-to-point wiring for individual segments.
- Supported by several hardware and software vendors.

Disadvantages of a Tree Topology

- Overall length of each segment is limited by the type of cabling used.
- If the backbone line breaks, the entire segment goes down.
- More difficult to configure and wire than other topologies.

5-4-3 Rule

A consideration in setting up a tree topology using Ethernet protocol is the 5-4-3 rule. One aspect of the Ethernet protocol requires that a signal sent out on the network cable reach every part of the network within a specified length of time. Each concentrator or repeater that a signal goes through adds a small amount of time. This leads to the rule that between any two nodes on the network there can only be a maximum of 5 segments, connected through 4 repeaters/concentrators. In addition, only 3 of the segments may be populated (trunk) segments if they are made of coaxial cable. A populated segment is one that has one or more nodes attached to it. In Figure 61, the 5-4-3 rule is adhered to. The furthest two nodes on the network have 4 segments and 3 repeaters/concentrators between them.

This rule does not apply to other network protocols or Ethernet networks where all fiber optic cabling or a combination of a fiber backbone with UTP cabling is used. If there is a combination of fiber optic backbone and UTP cabling, the rule is simply translated to a 7-6-5 rule.

6.5.1.4. Considerations When Choosing a Topology

- **Money.** A linear bus network may be the least expensive way to install a network; you do not have to purchase concentrators.
- **Length of cable needed.** The linear bus network uses shorter lengths of cable.
- **Future growth.** With a star topology, expanding a network is easily done by adding another concentrator.
- **Cable type.** The most common cable in schools is unshielded twisted pair, which is most often used with star topologies.

6.5.1.5. Summary Chart

Physical Topology	Common Cable	Common Protocol
Linear Bus	Twisted Pair Coaxial Fiber	Ethernet
Star	Twisted Pair Fiber	Ethernet
Tree	Twisted Pair Coaxial Fiber	Ethernet

6.6. Ethernet Crossover Cable

The 10BASE-T and 100BASE-TX Ethernet standards use one wire pair for transmission in each direction. The Tx+ line from each device connects to the tip conductor, and the Tx- line is connected to the ring. This requires that the transmit pair of each device be connected to the receive pair of the device on the other end. When a terminal device is connected to a switch or hub, this crossover is done internally in the switch or hub. A standard straight through cable is used for this purpose where each pin of the connector on one end is connected to the corresponding pin on the other connector.

One terminal device may be connected directly to another without the use of a switch or hub, but in that case the crossover must be done externally in the cable or modular crossover adapter. Since 10BASE-T and 100BASE-TX use pairs 2 and 3, these two pairs must be swapped in the cable. This is a crossover cable. A crossover cable must

also be used to connect two internally crossed devices (e.g., two hubs) as the internal crossovers cancel each other out.

Because the only difference between the T568A and T568B pin/pair assignments are that pairs 2 and 3 are swapped, a crossover cable may be envisioned as a cable with one modular connector following T568A and the other T568B (see Jack crossover wiring). Such a cable will work for 10BASE-T or 100BASE-TX. Gigabit Ethernet (and an early Fast Ethernet variant, 100BASE-T4) use all four pairs and also requires the other two pairs (1 and 4) to be swapped.



[8P8C](#) modular crossover adapter





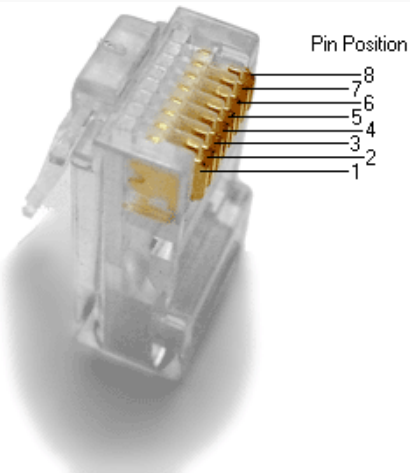














T568B wired connector



Gigabit T568B crossover cable ends



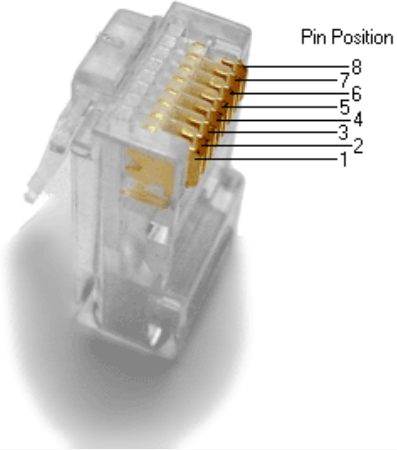


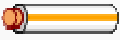











Crossover cable pinouts

Two pairs crossed, two pairs uncrossed 10BASE-T or 100BASE-TX crossover



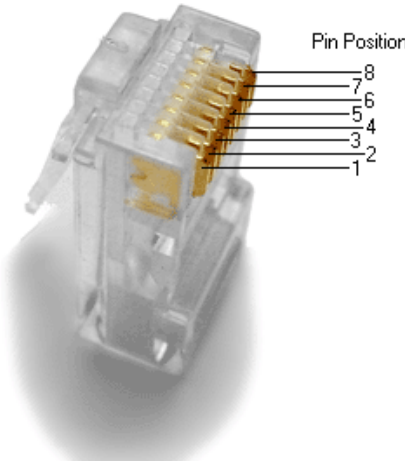



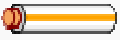










Pin	Connection 1: T568A			Connection 2: T568B			Pins on plug face
	signal	pair	color	signal	pair	color	
1	BI_DA+	3	 white/green stripe	BI_DB+	2	 white/orange stripe	
2	BI_DA-	3	 green solid	BI_DB-	2	 orange solid	
3	BI_DB+	2	 white/orange stripe	BI_DA+	3	 white/green stripe	
4		1	 blue solid		1	 blue solid	
5		1	 white/blue stripe		1	 white/blue stripe	
6	BI_DB-	2	 orange solid	BI_DA-	3	 green solid	
7		4	 white/brown stripe		4	 white/brown stripe	
8		4	 brown solid		4	 brown solid	

Certain equipment or installations, including those in which phone and/or power are mixed with data in the same cable, may require that the "non-data" pairs 1 and 4 (pins 4, 5, 7 and 8) remain un-crossed.

Gigabit T568A crossover
All four pairs crossed
10BASE-T, 100BASE-TX, 100BASE-T4 or 1000BASE-T crossover (shown as T568A)

Pin	Connection 1: T568A			Connection 2: T568A Crossed			Pins on plug face
	signal	pair	color	signal	pair	color	
1	BI_DA+	3	 white/green stripe	BI_DB+	2	 white/orange stripe	
2	BI_DA-	3	 green solid	BI_DB-	2	 orange solid	
3	BI_DB+	2	 white/orange stripe	BI_DA+	3	 white/green stripe	
4	BI_DC+	1	 blue solid	BI_DD+	4	 white/brown stripe	
5	BI_DC-	1	 white/blue stripe	BI_DD-	4	 brown solid	
6	BI_DB-	2	 orange solid	BI_DA-	3	 green solid	
7	BI_DD+	4	 white/brown stripe	BI_DC+	1	 blue solid	
8	BI_DD-	4	 brown solid	BI_DC-	1	 white/blue stripe	

Gigabit T568B crossover
All four pairs crossed
10BASE-T, 100BASE-TX, 100BASE-T4 or 1000BASE-T crossover (shown as T568B)

Pin	Connection 1: T568B			Connection 2: T568B Crossed			Pins on plug face
	signal	pair	color	signal	pair	color	
1	BI_DA+	2	 white/orange stripe	BI_DB+	3	 white/green stripe	
2	BI_DA-	2	 orange solid	BI_DB-	3	 green solid	
3	BI_DB+	3	 white/green stripe	BI_DA+	2	 white/orange stripe	
4	BI_DC+	1	 blue solid	BI_DD+	4	 white/brown stripe	
5	BI_DC-	1	 white/blue stripe	BI_DD-	4	 brown solid	
6	BI_DB-	3	 green solid	BI_DA-	2	 orange solid	
7	BI_DD+	4	 white/brown stripe	BI_DC+	1	 blue solid	
8	BI_DD-	4	 brown solid	BI_DC-	1	 white/blue stripe	



In practice, it does not matter if your Ethernet cables are wired as T568A or T568B, just so long as both ends follow the same wiring format. Typical commercially available "pre-wired" cables can follow either format depending the manufacturer. What this means is that you may discover that one manufacturer's cables are wired one way and another's the other way, yet both are "correct" and will work. In either case, T568A or T568B, a normal (un-crossed) cable will have both ends wired according to the layout in the Connection 1 column.

Although this crossover is called out in the Gigabit Ethernet standard[1], all Gigabit PHYs feature an auto-MDIX capability and are designed for compatibility with the existing 100BASE-TX crossovers. The IEEE specified Gigabit crossover is generally seen as unnecessary.

6.6.1. Automatic crossover

The automatic MDI/MDI-X configuration feature eliminates the need for crossover cables, making obsolete the uplink/normal ports and manual selector switches found on many older hubs and switches and greatly reducing installation errors. Note that although Automatic MDI/MDI-X is generally implemented, a crossover cable would still be required in the occasional situation that neither of the connected devices has the feature implemented and enabled.

Although Auto-MDIX is specified as an optional feature in the 1000BASE-T standard, in practice it is implemented on all 1000BASE-TX interfaces.

Modern switches automatically apply an internal crossover when necessary. Besides the eventually agreed upon Automatic MDI/MDI-X, this feature may also be referred to by various vendor-specific terms including: Auto uplink and trade, Universal Cable Recognition and Auto Sensing.