UNDERSTANDING DATA COMMUNICATIONS PROTOCOLS AND SOFTWARE

D R A F T   of   23 August 1988

Frank da Cruz   Christine Gianone

PREFACE

This book owes its existence to a course that we developed and taught at Columbia University in 1987 and 1988. The course, *Telecommunications Software*, was conducted in Columbia's Division of Special Programs, a certificate program for adults. The students ranged from computer novices to telecommunications managers. Some were proficient programmers, others had no programming experience at all. Few of the students had an engineering background. Finding a textbook that presented the ideas of data communication software and protocols to this mature and diverse audience -- a book that did not talk down to them, but yet was not too technical -- was a challenge. Early selections were roundly boo'd. As a result, we began to develop our own notes into something resembling a text. The result is what you hold in your hand.

The book can be used as a primary or supplementary text in courses in data communications or networking, either in academic institutions, or in professional development seminars. Since no particular technical knowledge is required beyond elementary programming ability in the C language and some high-school algebra, it should be suitable for use in high school, college, graduate school, and continuing education courses. It can also be used as a supplementary text in programming or software engineering courses.

Computer professionals who want to teach themselves or experiment with data communications programming, or who simply want to gain a real understanding of the issues of data communications, may also benefit from this book, as may programmers who want to learn about popular protocols from a software point of view. And so might data communications or programming managers who want a better understanding of what their employees are up to.

Numerous programming examples in the C language are included, and some in assembly language as well. But readers and students who are not programmers will still be able to make use of this book. Exercises are designed around the programs given here, requiring students to observe, report on, and explain their behavior. Programmers go one step further, and modify the programs or write new ones.

While existing protocols are described, none are specified or programmed in all their fullness; that would require a room full of books, preferably in loose-leaf binders for daily updating. Rather, it is our intention to familiarize readers with the fundamental principles, issues, and problems of software development and operation in general, and of data communications software in particular, ranging from PC communication programs to large-scale packet switched networks. Our central themes are the layering of levels of communication protocols, and the corresponding modularity of the software that interprets those protocols. Our goal is to develop sufficient background, tools, and criteria for evaluation of telecommunication and networking software and protocols in the ever-changing marketplace.

The computer programs were developed on the IBM PC family and the DEC VAXmate using Microsoft C and Assembler 5.0. Laboratories containing IBM PCs, ATs, PS/2s, VAXmates, or compatibles may be adapted for use with this book simply by connecting adjacent PCs with RS-232 null modem cables, so that each PC (except possibly the ones on the ends) communicates with two others, through two separate serial communication ports. This is a cheap networking laboratory indeed!

1. INTRODUCTION

Data communication boils down to two independently functioning devices, possibly of radically different design, speed, capacity, etc, trying to exchange information over a possibly hostile medium. Conventions, agreements, standards, and protocols are necessary at all levels:

- electrical
- physical
- timing
- format
- syntax
- semantics

These standards and protocols form building blocks that allow many types of communication to take place over a wide range of machines in different countries ... a message travels from one computer to the next over different types of links. Each computer knows how to talk to its neighbor at a low level, and the "end systems" know how to talk to each other, through all the intermediate computers, at a higher level.

Ultimately, in many cases, "data" is being transmitted from the keyboard of one user's computer to the screen of another's. But what is this data, and what are the computers? How does the design of the computer hardware constrain the form of the data and the design of the software that conveys it from one computer to another?

COMPUTERS, FRONT ENDS, COMMUNICATION PROCESSORS

What is a computer? It is a *tool* intended to free human beings of certain kinds of repetitive or exacting work: numerical calculations, information management, accounting, etc. Computers have had an enormous impact on society -- rooms full of file clerks and typists have been replaced by rooms full of disk drives and printers; mathemetical tables once calculated manually by armies of mathematicians; ... blah blah...

With proper programming, computers manipulate information -- often large amounts of it -- without complaint or error, without succumbing to boredom or whimsey ...

We may view a computer as a device that can accept data from an external source, store it, manipulate it in various ways, and produce results at an external destination, usually for human consumption. Furthermore, the manipulations may be specified by the user of the device by means of a "program", a series of instructions for the computer to perform. And what distinguishes a computer from most other machines is that its function can be changed simply by having it run a different program.

(Something more formal here... Definition, stored program concept, Turing machine, Babbate analytical engine, ...)

A modern computer has several aspects that are of particular concern to software:

- CPU - executes instructions, can do only one thing (instruction) at a time. The part of the CPU that executes instructions is called the instruction decoder. The CPU also has other parts.

- A repertoire of machine instructions (add, move, test, jump, etc).
- Memory - where instructions and data are accessed by CPU. Each memory location is denoted by an address.
- A program counter (PC), which contains the address of the next instruction to execute. Also called location counter.
- Registers - where data can be manipulated (arithmetically, etc). Also called accumulators.
- Virtual memory - address map between program address space & memory space.
- Memory protection, to protect co-resident processes from each other.
- Priv'd instructions (e.g. to switch processes, do i/o, halt)
- Realtime clock that provides interrupts (for scheduling)
- Interrupts, levels.
- Interrupt controller.
- Peripheral devices, controlled by the computer:
  - Input devices (like card readers), for loading instructions and data into memory.
  - Output devices (like printers), for conveying the results of computations.
  - Input/Output devices (disks, tapes), for storing information, or moving high volumes of it quickly to or from memory, or to serve as an extension of the high-speed internal memory.
- Communication devices: for communicating with a another device that is NOT controlled by the computer, e.g. person (e.g. through a terminal), or another computer.
- Device controller, to connect each device to the computer.

Let's look first at how the computer operates internally. Let's assume that a program has been loaded (somehow) into the computer's memory, at addresses 100 - 114. This program adds up a list of numbers. When it encounters a negative number, it stops adding and computes the average. The program is in a hypothetical (but typical) machine language, but written with alphabetic "mnemonic" codes for readability. The instruction format is "opcode register, address". In reality, the instructions are stored in the computer's memory as ordinary numbers, indistinguishable from data.

```
 Addr Instruction    Effect
 100  clear 1        Set register 1 to 0, increment PC (accumulator for sum).
 101  clear 2        Set register 2 to 0, increment PC (counter).
 102  lda 3, 111     Load address 111 into register 3, increment PC.
 103  load 4, (3)    Load the contents of the address in R3 into R4, incr. PC.
 104  bl 4, 109      If contents of R4 < 0, set PC to 109, else increment PC.
 105  add 1, (3)     Add contents of address in R3 to R1, increment PC.
 106  inc 2          Increment R2 (counter) and increment PC.
 107  inc 3          Increment R3 (data adress) and increment PC.
 108  jump 103       Set PC to 103.
 109  div 1, 2       Divide contents of R1 by contents of R2, increment PC.
 110  halt           Stop execution.  The average is left in R1.
 111  75             Data.
 112  18             Data.
 113  164            Data.
 114  -1             Data.
```

Thus, each instruction not only performs the indicated operation, but also specifies what happens to the program counter when the instruction has completed. This is how the instructions are executed in sequence, and also how the normal sequence can be broken.

The data and instructions are stored together, in the same memory. There is nothing about their contents or form that distinguishes them. For instance, the instruction "div 1, 2" might be identical to the data "164". The only thing that keeps them apart is that the computer has been instructed (somehow) to begin execution at statement 100, and the program is written in a way that prevents any of the data from being loaded into the CPU and executed, i.e. the program counter never points at the data (careless programmers often find their programs "executing data", with totally unpredictable results).

This is our first of many examples of the intermixing of data and control information in a single stream. This is a commonly used, and often necessary, technique in programming and in data communication. How we prevent data from being misinterpreted as control information, and vice versa, is a major issue, one we'll return to often.

DATA COMMUNICATION

Data communication is the process of getting data into and out of the computer's memory or storage through a communication device (as opposed to a peripheral device).

In most modern computers, the device controllers are little autonomous computers that have two "ends" -- one end is connected to the computer's memory, and the other to the device. Thus the device controller is an "interface" between the device and the computer. It shields the computer from the specific electrical and mechanical characteristics of the device, and vice versa.

The device controller has registers which are accessed as if they were computer memory locations. Some of these registers are for data, and others are for control purposes. The control registers include quantities called "flags", which indicate that an event, such as the arrival of data, has occurred and needs to be handled.

POLLING:

Input and output can be "polled" or "interrupt driven". In a dedicated, single-user computer, or a device like a terminal, which only does one thing at a time, the CPU can be executing a program which repeatedly looks at the device's data-ready flag while waiting for input, and then fetches the desired data item from the data register when the flag appears, and then "resets" the flag. This is called "polling", and it prevents the computer from doing any other work while it is waiting for input. Example:

```
100 load 1, 500   Load the flag register.
101 be 1, 100     If it contains zero, go back & load it again.
102 load 1, 501   Flag nonzero, so now load data.
```

One obvious problem with this approach is that if the data never arrives, the computer will be stuck in its polling loop forever. Such an input request is called "blocking": the process blocks until the requested input arrives.

One way to avoid blocking is to restrict the number of times the device will be polled before the computer gives up:

```
100 lda 2, 1000   Only look 1000 times
101 load 1, 500   Load flag register
102 bn 1, 106     If nonzero, set PC to 105
103 dec 2         Otherwise, decrement loop counter
104 bge 2, 101    If result is greater or equal to zero, loop.
105 halt          Failed to get input, halt.
106 load 1, 501   Flag went up, load data.
```

Suppose it takes 1 second to execute instructions 101-104 a thousand times. This means that we have set a 1-second "timeout" on the input. However, if we go out and buy a new model of this computer which is twice as fast, then it will time out after half a second, instead of a single second. This is a danger of "timing loops".

INTERRUPTS:

Most modern computers allow other work to progress while waiting for input. Polling is not done, and timing loops are not necessary. How is this possible? These computers contain a device called an interrupt controller. This device is actually a separate, special-purpose computer which constantly polls the flag registers of all the device controllers. Whenever it sees a flag "go up", it saves the value of the program counter in a special place, and then looks in a special table, called the "interrupt vector", at the entry associated with the device, and takes the address stored there and puts it in the program counter. The address in the interrupt vector is the location of an "interrupt service routine".

Here's a simple example, in which the computer counts up until the user types a character:

```
100  clear 1      Set R1 to zero
101  inc 1        Increment R1
102  jump 101     Do it again and again and ...
103  halt

210  348          Interrupt vector
211  785          ...
212  103          Entry for data available at terminal
213  442
...
```

As soon as the user types a character, the program halts, with the count left in R1. In a more realistic example, the computer doesn't halt, but copies the character from the data register to a place where the program can use it. For instance, terminals are typically connected to computers through a device called a UART (Universal Asynchronous Receiver/Transmitter). The UART has a communication cable connector on one end, and registers and flags on the other. When a character arrives, the data-ready flag goes up. The computer must copy the data character out of the data register and "push down" the flag before the next character arrives and overwrites the data register.

On a system with many such devices, flags are going up all the time. The machine is jumping around frantically, responding to the data-ready interrupts, very similar to how you play the carnival game Whack-A-Mole.

FRONT ENDS:

Even when i/o is interrupt driven, a computer that does a lot of communication will find itself spending large amounts of time handling interrupts. Every time a character arrives at a communication port, the computer must stop what it's doing, get and dispose of the character, and then put things back the way they were (more about this next time).

Some computers come with "front end communication processors" to relieve the main computer of much of this tedium. These front ends handle the interrupts, collect the data, and then neatly deposit it directly in the computer's memory. This allows the computer to devote its resources to "real work".

ENCODING -- TRANSMISSION -- DECODING OF DATA

We've shown how a computer stores instructions, addresses, and data all in the same memory space. But how is the memory space organized, and how are the contents of the memory interpreted? First, the obligatory digression on...

NUMBER SYSTEMS:

A string of m digits a(m-1), a(m-2), ..., a(1), a(0) in base N represents an m-digit number, the sum of a(i) * N**i for i = 0 thru m-1. The digits for base N range from 0 to N-1, where each digit is a single symbol. If N is greater than 10 (decimal), then usually alphabetic or other characters are used.

The DECIMAL (base 10) system we use in everyday life uses the digits 0-9.

```
A decimal number 1234 = 1 x 10^3 + 2 x 10^2 + 3 x 10^1 + 4 x 10^0
(powers of 10)        =  1000    +  200     +   30    +   4       = 1234
1, 10, 100, 1000, ...
```

In a computer, numbers (integers) are represented in base 2, using the digits 0 and 1. This is called the BINARY system.

```
A binary number  1011 = 1 x 2^3  + 0 x 2^2  + 1 x 2^1  + 1 x 2^0
(powers of 2)         =   8      +   0      +   2      +   1      = 11
1, 2, 4, 8, 16, ...
```

OCTAL (base 8) uses 0-7. An octal digit corresponds to exactly 3 bits.

```
An octal number 2573 = 2 x 8^3 + 5 x 8^2 + 7 x 8^1 + 3 x 8^0
(powers of 8)        =  1024   +  320    +   56   +   24          = 1424
```

HEXADECIMAL (base 16) uses 0-9 and A-F. A "hex" digit corresponds to exactly 4 bits, e.g. 9F hex = 1001 1111 binary. A decimal number doesn't.

```
A hex number 1E9A = 1 x 16^3 + 14 x 16^2 + 9 x 16^1 + 10 x 16^0
(powers of 16)    =  4096    +   3584    +  144     +   10        = 7834
1, 16, 256, 4096
```

How many different numbers be represented in n decimal digits? 10^n. How many different numbers be represented in n binary digits? 2^n.

INTERNAL REPRESENTATION OF DATA

Computers store all data as seqeunces of binary digits, or "bits", i.e. zeros and ones. The computer's memory is organized into fixed-size chunks called "bytes" and "words". A byte is typically 8 bits long, and a word is usually 16 or 32 bits. A byte is typically used to hold a character, and a word is typically used to hold an instruction, an address, or a number. A register is usually word-size.

Usually, the smallest unit of data in the computer's memory that can be directly addressed or manipulated is a byte. The finer the granularity of address references, the longer must be the addresses. Since the word length of the machine determines the amount of circuitry that most go into all of its data paths and registers, it is not practical to have very long words. Therefore, to get the most out of an address, it is used to rererence a byte or a word, rather than an individual bit.

The other side of the coin is that the more granular the addresses, the more instructions are required to extract bytes or other fields from within words. A common design is a 32-bit word length, with byte addressing, allowing 2 to the 31st power bytes to be addressed. That's about 2.15 billion bytes.

CHARACTERS:

People have been storing and communicating information for thousands of years. Computers are only doing what people have always done, but in a more formalized, mechanical way.

In spoken language, ideas are expressed in "real time" as sequences of sounds, which compose words, phrases, and so forth, strung together according to commonly accepted rules. Thus speech symbolically represents ideas.

Information may be stored in a graphical way for reference at a later time, as pictures or hieroglyphs (which substitute for speech by representing ideas directly), or as letters, digits, and punctuation (which are a "second-level" symbolism: symbols that stand for symbols that stand for ideas).

Pictographic symbolism is more widely comprehensible than alphabetic writing. Pictures can be understood by people who speak many different languages. Even when highly stylized, such as Chinese ideograms, they can form the basis of communication between people who have no spoken language in common. Alphabetic or numeric writing, on the other hand, is specific to a particular language or number system.

When machines were first invented that could store or process information, a method was needed to encode symbols into a form the machine could store and manipulate, because these early machines could not be expected to read documents, let alone pictures, directly. The first method devised for storing data to be read by machine involved punching holes into cards. The Jacquard loom (early 1700s) and the Hollerith tabulating machine (1890 census) were two landmark devices.

Since these machines were invented in Europe and North America, the machine symbols were chosen to correspond to alphabetic or numeric symbols, plus any additional symbols necessary for control of the machine itself.

Thus a third level of symbolism was introduced: a machine-specific representation of language-specific symbols for ideas. These three levels of symbolism persist into the present day.

CHARACTER SETS:

Data communication, as we know it today, began with telegraphy in 1837: Morse code, in which the symbols are assigned to letters based on their frequency of occurrence in typical English text (ETAIONSHRDLU...), the most common letters having the shortest symbols (Wheel of Fortune).

```
 E .   A . _   R . _ .  L . _ . .   1 . _ _ _ _
 T _   I . .   S . . .  X _ . . _   2 . . _ _ _
```

Dit (.) = 1 time unit, Dah (_) = 3, Interchar space = 3, Interword space = 5. Letters have 1 - 4 symbols, digits have 5, punc & ctrl have 6.

Morse code is well suited for human transmission & reception -- it minimizes the code length for typical messages, and therefore maximizes the transmission efficiency. And it also has the advantage of extensibility -- additional code words can be added with 7 bits, 8 bits, 9 bits, etc. (not that this has been done)...

Although Morse code is a good encoding for transmission, it is not well suited for storage in computers, which have fixed-length bytes that can contain any pattern of 0's and 1's.

How can we design a character set to fit into fixed-length bytes? First, note the length of the byte, in bits, call it n. Then, you can have 2^n symbols in your alphabet. Make a list of the numbers 0 through 2^n - 1, and pick a symbol to go with each number. Then build keyboard and printing devices that honor this code. For instance, if you push the "A" key, it sends a byte with value 65 to the computer. If the computer sends that same byte to a printer, the letter "A" appears.

What are some criteria to be used in designing a code? (ask the class)

- It should include all the symbols you'll ever need, because once you've built all these keyboards and printers, it'll be tough to add new symbols. (you've created a standard!)
- The codes should be assigned in alphabetical and numeric order, so that the computer can easily sort records composed of these characters, and algorithms for converting between numeric character strings to internal numbers can be simple. (show a sample program for converting numeric strings to internal numbers...)
- There should be some special characters used for control, rather than data.

How does Morse code fit these criteria?

Here are some real computer codes:

12-symbol Hollerith code (1890) - 10 digits and 2 special symbols.

A card code, one punch per row. Rows 11 and 12 for control. Around 1932, this code was expanded to include letters, -, *, and &.

5-bit Baudot (CCITT #2, 1931)

Uppercase letters, digits, punctuation marks. But 2^5 = 32 is less than 26 UC letters + 10 digits, so special characters are used to indicate character set shifts: 11011 = Figure Shift (digits and punctuation), 11111 = Letter Shift. Baudot code was intended for use with Teletype machines, which were to replace Morse-code telegraphy. Teletypes had the advantage that they "transcribed" the message automatically (by printing it on paper), and could also store the message for later retransmission on punched paper tape.

Aside from its limited repertoire of characters, Baudot has certain other unfortunate characteristics: letter codes are not in alphabetical order, numeric codes are not in numeric order, etc., and the same code word can mean two different things, depending on the shift state (which means you can't tell what a character is if you look at it out of context). This makes Baudot encoding poorly suited for computer use.

6-bit IBM BCD or BCDIC (2^6 = 64) (card code)

IBM. Evolved from Hollerith code, originally a card code, eventually also an internal code. As a card code, BCD allowed multiple punches per column. BCD went through several versions:

V1	late 1950s: 48 graphics, punches and numeric codes defined for use on early IBM computers.
V2	Addition of European character substitutions for @#$, and addition of ()+=' for Fortran and math, subst for %@#$ and box.
V3	1962: 6 bits, 64 graphics, allowing math and commercial symbols to coexist.

Despite the problems of each version, codes for alphabetic and numeric characters were in natural order, allowing easy translation and sorting.

7-bit ASCII (first version 1963, present version 1977) (2^7 = 128)

The result of standards committee work. The aim was to design a standard code to be used by all computers worldwide for information interchange. In order to encompass the requirements of commercial, scientific, and military computing, plus AT&T and Western Union requirements, plus the need for control characters, it was decided that more than 64 characters would be needed. The CCITT code extension technique of shifting was avoided because of the effects of line noise on shift characters. A 7-bit code was chosen, which allowed for lowercase as well as uppercase characters. An 8-bit code was not chosen for reasons of economy.

8-bit EBCDIC (1964) (2^8 = 256)

IBM. Extension of 6-bit BCD to 8 bits, allowing for lowercase letters, etc., math symbols, commercial symbols, plus space for future expansion.

Unfortunately, all of these sets coexist. Once a character set becomes popular, it never goes away. For instance, we now have:

ASCII, used in most terminal communication, dialups, DEC, IBM PC, etc.

EBCDIC, used in IBM mainframes.

Baudot, used in TDD (Telecommunication Devices for the Deaf).

CONTROL VS GRAPHIC CHARACTERS:

The character sets listed above include codes for letters, digits, space, and some punctuation characters. These are known as graphic characters: they cause ink to appear on the page, or phosphor to light up on the screen (with the exception of space, which also counts as a graphic).

Character set designers realized early on that special codes would be necessary to allow the terminal user to control the computer, or the computer to control the Teletype or printer. Therefore, most computer character sets include a set of "control characters", like:
Carriage Return (CR): return the carriage (or cursor) to the left margin.
Line Feed (LF): Move the print head (or cursor) down one line.
Form Feed (FF): Advance paper to top of form (next page), or clear screen.
Horizontal Tab (HT): Move print head or cursor to next tab stop.
Bell (BEL): Sound the terminal's bell (or buzzer, or beeper).
and many more. These control characters are inserted in the appropriate places in the stream of graphic characters to cause the desired action. The ones listed above are fairly standardized -- i.e. they can be expected to cause the same action on any terminal or printer. Others are not. Their effects will vary according to the actual device.

The following program demonstrates how the same data in the computer may be interpreted as a number or a character:

```
#include <stdio.h>
main() {
    int i;
    for (i = 0; i < 256; i++ )
        printf("%3d  %c\n",i);
}
```

If you run this program, you will see two columns whizz past you on the screen -- the left column showing the numeric value, and the right the corresponding ASCII character. But if you want to see them all on the screen at the same time, you can try to modify the program to display 8 columns of 16 rows each:

```
#include <stdio.h>
main() {
    int i,j;
    for (i = 0; i < 16; i++ ) {
        for (j = i; j < i+113; j+=16)
            printf("%3d  %c | ",j,j);
        printf("\n");
    }
}
```

Try running this program. Why don't the columns line up?

CHARACTER SET TRANSLATION:

How can data from one set be translated to another? Easy enough if the sets were all of equal length: simply make a table, whose addresses correspond to the numeric character values in one set, with the contents of each address being the value of that character in the other set.

But the lengths are not equal. E.g. EBCDIC has twice as many characters as ASCII, therefore it is impossible to translate arbitrary data from EBCDIC to ASCII, and expect to be able to translate from ASCII back into EBCDIC. In general, a translation from a longer code to a shorter one is not "invertible".

A major difficulty in data communications involves translation between different character sets. In practice, ASCII/EBCDIC tables have been designed, but they are not... (fill in)

EXTENDED CHARACTER SETS

How do we accommodate "foreign" alphabets -- Hebrew, Arabic, Cyrillic, Chinese, Japanese (Hiragana, Katakana, Kanji), Hindi, etc etc.?

For languages like German, Norwegian, and Swedish, a commonly used technique is to appropriate some of the less-frequently used US ASCII or EBCDIC graphic characters, and assign different graphics to them, e.g.:

USASCII	Germany	Norway/Denmark	Sweden/Finland
[	Umlaut A	A/E	Umlaut A
\	Umlaut O	Slash O	Umlaut O
]	Umlaut U	Circle A	Circle A
{	Umlaut a	a/e	Umlaut a
|	Umlaut o	Slash o	Umlaut o
}	Umlaut u	Circle a	Circle a

But this makes the USASCII bracket, slash, and bar characters unavailable, which is, at best, an inconvenience for programmers (imagine what a C language program looks like without `{}[]\|`). And it doesn't really solve the problem, even for these languages -- there are still important characters not represented, like German ess-zet (double S).

What about languages with totally different alphabets? What if the number of characters in an alphabet exceeds 2^8? What if text must be displayed in a mixture of alphabets?

Computer architecture is already pretty well fixed -- 8-bit bytes, etc., so can't start inventing 9-bit, 10-bit, ... codes. Current schemes propose "alphabet shifts", as in Baudot. E.g. 11111111 = alphabet shift, next byte tells which alphabet: 0 = Roman, 00100110 = Greek, 00100111 = Russian, etc.

This scheme is reminiscent of the old Baudot shift code extension scheme, and has the same drawbacks, e.g. what happens if the shift character, or alphabet code gets corrupted?

And this doesn't even begin to address the question of whether writing in a particular language goes from left to right, or vice versa (or up and down).

The nice thing about standards is that we have so many to choose from! Some current character-code standards include:

7-bit ASCII is ANSI standard X3.4 and ISO 646 and CCITT T.50.
ANSI X3.32 specifies graphic renditions for control characters.
ANSI X3.41 and ISO 2022 give 8-bit code extension techniques for ASCII.
ANSI X3.134.1 & ISO 4873 specify an 8-bit code for information interchange.
ANSI X3.134.2 specifies an 8-bit ASCII multilingual character set.
ISO 6937 specifies coded character sets for text communication.
ISO 8859/1 = IBM "code page 500", a superset of ANSI X3.26 (1980)

The fact that there are so many competing standards, plus nonstandard vendor-specific approaches (like the IBM PC character set), indicates that it will be a long time before terminals and PCs are capable of operating in a multilingual world.

TERMINALS

Now that we have settled on schemes for representing and storing characters, how do we get them in & out of computers?

Terminals are devices that let people communicate with computers. Terminals are just special-purpose computers connected to a keyboard, a screen, and a communication port. The keyboard allows you to type characters which are sent out the communication port to the computer. The screen allows you to see the messages you type and the respones the computer sends back to you. Terminals are the basis for much of today's data communication.

Each character you type is sent out the communication port as a 7-bit or 8-bit numeric code in a particular character set, usually ASCII. Each code that arrives at the communication port causes the corresponding character to be displayed on the screen. If the computer does not echo the characters you type, then the terminal also displays these on the screen.

(DRAW PICTURE)

A key issue is unpredictability. The computer does not know in advance when person will hit a key (fingers!). The terminal does not know when computer will send characters. Both could happen at same time. This is why communication lines tend to be interrupt-driven on computers. In terminals, the processor constantly polls the keyboard and the port (it can afford to do this, because, unlike a general-purpose computer, a terminal has nothing else to do).

Modern video display terminals (VDTs) have many controls & options, particular to each VDT (e.g. VT100). These are used by software on the host computer to allow full-screen applications, to produce special effects, and so forth:

- Communication settings (speed, parity, etc: covered later)
- Local echo, remote echo
- Loadable or switchable character sets (possibly programmable) function keys
- Escape sequences for screen control, special effects (blink, reverse, underscore, color, curor positioning, screen partitioning, editing, etc)
- Graphics (needs special language, like Tek 4010)

Escape sequences are introduced by the ASCII control character "escape" (ESC), which tells the VT100 to interpret the subsequent characters as a command, rather than data.

Escape sequence examples for VT100:

```
ESC [ 0 J      Clear screen
ESC [ 3 A      Cursor up 3 lines
ESC [ 0 K      Erase from cursor to end of line
ESC [ 4;30 f   Position cursor at row 4, column 30
ESC [ 7 m      Reverse video
```

So the VT100 CPU does not simply copy incoming characters to the screen, but looks for escape sequences. When found, the VT100 treats them as commands and takes the corresponding actions. (This is another instance of mixing control information with data.)

The list of VT100 (or 200, or 300) escape sequences is quite long. In fact, the manual the for new DEC VT320 terminal is about an inch thick. This should give you an idea of what's in store for someone who wants to write a PC program that emulates such a terminal.

We will discuss terminal emulation in more detail in a future meeting.

SOFTWARE

So far, we've looked at several of the fundamental components of computing and data commmunication: computers, front ends, interrupts, number systems, character sets, and terminals.

Now, we'll look at software: the glue that holds all of these elements together. "Software" is a word originally coined to distinguish computer programs from the physical computer and its pieces, i.e. from the "hardware". A program is a set of instructions that is loaded into the computer's memory for execution. A program is not a permanent part of a computer. This is what distinguishes computers from most other kinds of machinery -- by changing their programs, you can make them do entirely different tasks: payroll, weather prediction, word processing, scientific calculation, ..., and the task that will concern us most, data communication. It is because computer programs can change that they are called "soft".

It is possible for a computer to have "read-only memory" (ROM), in which a program is permanently stored. For instance, most computers have a "bootstrap ROM", which contains the program the computer executes when it is first turned on. This program looks on the disk for the operating system, loads it into memory, and starts it. If it were not for the bootstrap ROM, you would have to enter this program into the computer's memory yourself every time you turned the computer on. These permanent programs are sometimes called "firmware" (hard software?). But firmware is really software, developed the same way, but then etched into memory once it is fully operational and (one hopes) debugged.

PROGRAMMING:

Programming is the process of creating software. It is really a form of writing, in which the writer specifies a task to be done and exactly how to do it. Very similar to writing a cookbook, or assembly instructions for a bicycle, but somewhat more formalized, with more steps.

```
+-------------------+
| Programmer        | (You :-)
+-------------------+
| Documentation     | Manual for programming language, OS
+-------------------+
| User Program      | The program you write
+-------------------+
| Libraries         | e.g. IMSL in Fortran, stdio in C, or runtime system
+-------------------+
| OS Services       | Device drivers, etc
+-------------------+
| Hardware Services | Machine instructions, interrupts, physical i/o, etc.
+-------------------+
```

In addition to writing the program, the programmer must also write documentation for the program, so that the user knows how to use it. User manuals for typical popular programs (word processors, database managers) can be quite thick. Documentation should also be written for future maintainers of the program, since we all know that professional programmers rarely stay in the same job for more than a year. In fact, the program itself should be considered the ultimate form of documentation about itself, and programmers should always be encouraged to write their program as much for future human readers as for the computer.

MACHINE LANGUAGE:

All programs, no matter what language they are written in, must ultimately be translated into machine language before they can be executed. Machine language is the internal language of the computer, typically stored one instruction per computer word, with an instruction consisting of an operation code, specifying which instruction to execute, and one or more operands, such as registers or memory addresses.

A list of such instructions constitutes a program. This program specifies, in minute, exact detail, what the machine is to. The machine does exactly as it is told, and has absolutely no idea what the programmer's intentions may have been. It is up to the programmer to code the program correctly.

Example (IBM PC machine language, addresses shown in hex):

```
Address Contents (binary)                                  Interpretation
 0000    0000010011010010                                   Data = 1234
 0002    0001000111010111                                   Data = 4567
 000A    000000000000101010001011000101100000000000000000  MOV DX, 0000
 000E    00000011000101100000000000000010                   ADD DX, 0002
```

Note, the IBM PC has variable length instructions.

ASSEMBLY LANGUAGE:

Back in the old days, machine language programs were actually entered into the computer by hand, a bit at a time, using little switches. The programmer had to have know the bit patterns of the operation codes, the exact instruction format, etc.

Furthermore, since instructions may contain the addresses of data or other instructions, these addresses had to be known beforehand, and if an instruction needed to be inserted or deleted, then many other instructions would have to be adjusted to reflect the changes of address.

Clearly, programming in machine language is tedious and error prone.

Assembly language allows substitution of textual symbols for machine instructions and addresses. This way, the programmer does not have to memorize bit patterns for the opcodes, and the computer can do the bookkeeping required for address resolution.

Generally one assembler statement corresponds to one machine instruction. An assembler program translates symbolic assembly language source into binary machine code. It looks up operation codes in a table, and substitutes the corresponding numeric values, e.g.

```
CLC = 11111000 (Clear Carry Flag)
CLI = 11111010 (Clear Interrupt Flag)
LEA = 10001101 (Load Effective Address)
NOP = 10010000 (No Operation)
```

Address resolution is a bit more difficult, since addresses can be referred to before they are defined. For this reason, assembly generally proceeds in two passes. Pass 1 translates the opcodes and makes a table of labels and their corresponding offsets (among other things), leaving "blanks" in the address fields of any instructions that refer to addresses that are defined further down, e.g.

```
Assembly Code        Pass 1              Pass 2
 A: JMP B       100/  11101011 (B)        11101011 11001000 (= 200)
    :
 B: CLI         200/  11111010
```

In pass 2, after the values of all symbolic addresses are known, the assembler fills in all references to them.

The original assembler was probably written in machine language, and then recoded into assembly language, assembled by the machine language version, which could then be discarded forever. Then, the assembler can be used to assemble itself, as new features are added. This process is called "bootstrapping".

Assembly language is used for various reasons:

- On many systems, it's the only programming language that is supplied free (the IBM PC is a notable exception).
- It provides access to data manipulation functions that may be unavailable in a particular high-level language, especially bit manipulations (test and/or set, logical AND, OR, and XOR), unsigned arithmetic, shifting, etc.
- It provides direct access to specific addresses, allowing programs to get at device registers and interrupt vectors, allowing direct control of i/o .
- Hand-coded assembly routines can sometimes be much more efficient than the code generated by high-level language compilers.

However, assembly language is not at all transportable between different kinds of machines. It is also comparatively tedious to code, since it deals in minutiae, and once coded it is hard to read.

ACCESS TO OS SERVICES FROM ASSEMBLY LANGUAGE:

What happens when a user program attempts to execute an instruction that is not known to the machine? Most machines handle this problem by generating an "illegal instruction" interrupt. Control immediately passes to the OS's illegal instruction interrupt handler, which normally terminates execution of the program and displays some kind of error message.

But selected operating system services, such as opening files or doing input and output, must be made available to user programs. But many computer systems are organized so that the operating system is in a separate address space from the user program. This prevents the user program from writing over and destroying the operating system. But it also prevents the user program from "calling" subroutines in the operating system, because the user program can't "see" them. Access is often provided through the illegal instruction interrupt. A special "illegal instruction" is defined to cause this interrupt, and when the system's illegal instruction handler gets control, it checks if the opcode is this special one, e.g. (on the PC)

```
INT = 11001101
```

and if it is, then checks the effective address, and treats it, plus any data in the registers, as a "function code" and arguments, e.g.

```
MOV DX, OFFSET STRING
MOV AH, 09H
INT 21H
```

is a "DOS call". Register AH contains the function code "09H" which means "print a string", and DX contains the address of the string.

As an aside, MS-DOS does allow user programs to write into the operating system, so care must be taken to avoid this. But in some cases, it's desirable. In fact, this is how many PC utilities work. The method generally used is to substitute the address of some user-written code into the interrupt vector for a particular interrupt. Then, whenever the interrupt occurs, the user code, rather than the normal operating system handler, is executed. For instance, many data communication programs work this way, because the MS-DOS communication device handlers are generally quite slow and limited in functionality. When such a program exits, care must be taken to restore the interrupt vector to its previous state.

System programs are the ones that need to rely most heavily on the underlying OS and/or hardware, and must have access to the widest possible range of system facilities. Data communications programs are good examples.

Often access to these very specific and/or low-level functions can be achieved only from assembly language. So high-level languages that need to do this must call upon assembly-coded functions.

HIGH-LEVEL LANGUAGES (PROCEDURAL):

High level languages allow procedures to be specified in generic constructs independent of the underlying machine's architecture and instruction set. One statement in a high level language may correspond to many machine instructions. High level languages include:

- Variables
- Data types (integer, real, string, ...)
- Data structures (arrays, structures, lists, ...)
- Expression evaluation
- Assignment of value (expression) to variable
- Blocks & scoping of variables
- Arithmetic and other operators (+, -, /, *, etc), with precedence
- Arbitrarily complex arithmetic expressions
- Relational operators (<, >, =, etc) and expressions
- Data type conversions in expressions or assignments
- Control structures (IF-THEN-ELSE, FOR, WHILE, etc)
- Procedures (functions, subroutines), parameters passed by reference, value
- Printing and formatting functions (numbers to strings, strings in fields...)
- Comments
- Only limited access to OS services (OPEN, CLOSE, READ, WRITE, etc)

There are relatively few popular high-level languages: C, Cobol, Basic, Pascal, LISP, Fortran, PL/I, Ada, etc. The definition of a high level language tends to be fairly uniform across machines and operating systems so that, compared to assembly language, it is painless to move a high-level language program from one machine to a different one.

Most languages provide access to OS services through functions like open, close, read, write, etc, which are calls to the file system.

"VERY HIGH-LEVEL LANGUAGES"

Tend to be non-procedural. For instance, a SCRIBE document "program" describes the general style of a typeset document, freeing the "programmer" from specifying the procedure -- "make a CACM article", "this is the title", "this is a footnote", etc. (SCRIBE knows exact format of columns, footnotes, bibliography, etc, for each document style). Another example: database query languages -- "tell me all the left-handed males older than 35 who weigh less than the average female younger than 35".

THE PROGRAM DEVELOPMENT CYCLE

Source programs are created by the programmer using a keyboard (terminal, PC, or even keypunch), often in conjunction with a text editor or word processing program. The result is one or more text files, which are readable both by people (e.g. on the screen, or printed on paper) and by the computer's language translator (compiler or assembler). These files are called "source files".

A program may be built from one or more source files, or "modules", written in assembly language or a high-level language. The assembler or compiler produces an intermediate form of machine code, called "relocatable", or "object", code, which must be "linked" and "loaded" before it can execute.

LINKING AND LOADING:

Each module's internal addresses must be adjusted to reflect the module's position relative to the beginning of the program, after all the modules have been concatenated and linked together. This is done by a linker program.

Linking also resolves cross references, for instance when Module B refers to a symbol defined in Module A. During the link process, the linker makes a table of such references on the first pass, and fills them in on the second, much as the assembler resolves addresses within a module.

When an executable program is loaded from a disk file into memory, some last-minute fixups may be required, like loading base or segment registers, to reflect the actual physical address where the program has been loaded. Once the fixups are applied, the operating system locates the program's starting address and transfers control to it. E.g. on IBM PC or 370. Loader finds free memory, puts program in it, then sets base register.

Often, the linking and loading functions are combined into the same program, e.g. MS-DOS LINK. The result is a file that contains an executable program -- approximately identical to the memory image of the program.

LIBRARIES:

Languages often have libraries of commonly used functions, which can be included with user programs, like the Unix stdio library (printf, etc), or the Fortran IMSL library (math functions). These are generally in the form of separate, precompiled modules (i.e. object modules) that are concatenated together into a single file. Libraries can be either monolithic (the whole library gets loaded with your program) or indexed (the required modules are extracted and loaded with your program).

On some systems, libraries can take the form of "runtime systems", which are combined with your program at runtime, so that the code does not take up space on disk. The pitfall here is that when the runtime system changes, the programs that use it might stop working.

THE PROGRAM DEVELOPMENT CYCLE

The program is created by typing it into a text editor. Then it is compiled or assembled, then linked and loaded, and then the executable program is invoked. Errors can occur during compilation, linking, or execution, and when they do, the program can be debugged.

```
     types        compiles         links              debugs
User -----> source --------> object\
            source --------> object >-------> executable -------> (back to
            source --------> object/                                source)

            headers          libraries        runtime system
```

An executable program, once loaded into memory, may be debugged at machine-code, assembly, or source level if the tools are available (often, they are not). Such tools allow the program to be single-stepped, variables examined and changed, etc. When the bug is located, the fix is applied in the editor to the source code, and the process repeats until the program is complete.

Program development may be "top-down", or "structured" -- write gross outline, then fill in successive levels of detail -- or "bottom-up", usually when the whole program depends on some low level function -- write and debug the low-level function, then add on upper layers that use it. The bottom-up style is common in data communications programming, where low-level communication must be established before high-level operations can take place.

COMPILERS VS INTERPRETERS:

A compiler or assembler produces immediately-executable machine code (after linking & loading). Advantages: efficient execution, etc. Disadvantages: tedious development cycle.

Interpreter (like BASIC, LISP): statements are "compiled" each time they are encountered. The program development cycle described above is entirely sidestepped. Appearance of execution directly from source code. Advantages: easy program development, debugging. Programs may be stopped, variables examined, modified (in some cases), continued or restarted. Disadvantage: slow execution.

OPERATING SYSTEMS

An operating system is a special piece of software, providing the environment in which all other software runs. It presents a "virtual machine" to the user, hiding the details of the particular physical machine. This allows programs to be portable, generic, rather than specific to a particular machine, memory, and device configuration.

The OS is the basic control program of the computer. Its principal job is to allocate and protect the resources of the computer.

An OS is a good example of layered, modular software. Each piece has well defined functions and well defined interface to other pieces.

We will look at a full-fledged multiprocessing, interactive, timesharing, multiuser OS. Microcomputer OS's lack many of these functions, but there is increasing pressure to add them in (e.g. in OS/2) -- the microcomputer OS's are gradually growing into the "old" multiprocessing model.

```
LAYERS OF AN OS:        ***** DRAW THIS A PIECE AT A TIME *****

7     User :-)  (and manual?)
    +-----------------------+
6   | Shell or User Program |  User context
    +-----------------------+---------------
5   | Scheduler             |  Supervisor
    +-----------------------+  Context
4   | File Access           |
    +-----------------------+
3   | Input/Output          |
    +-----------------------+
2   | Memory Management     | /Wait & Signal
    +-----------------------+/
1   | Kernel                |--First Level Interrupt Handler
    +-----------------------+\
0   | CPU, memory, etc.     | \Dispatcher
    +-----------------------+
     (LEVEL 0, PICTURE FROM LAST TIME)
```

What is all the stuff below the user program? Why can't you put a Lotus or Kermit diskette into the disk drive, turn on the machine, and run the program directly, without an OS?

Just as functions that are commonly used by programs in a certain language are gathered into a library or runtime system, the functions that are common to all programs that run on a computer are gathered into the operating system. Therefore, all programs can use these functions in a common, consistent way, and each programmer does not have to code these functions into each program. Given the variation in computer equipment, this would not even be possible, since the program would have to know what slot a board is in, which board it is, etc etc.

And finally, there are functions the user program simply cannot be trusted with -- disk and file management, etc. And of course, on multiuser systems, the operating system must schedule users' access to the system's resources, and must protect the user processes (and files) from one another.

The interactive user only sees the shell, or "user interface" (e.g. COMMAND.COM on the PC). The programmer may also be allowed certain restricted entrees into the OS.

BASIC NOTIONS

Process: "An address space in execution", a runnable (or running) program.

Interrupt: An asynchronous (unpredictable) event, noticed by hardware (interrupt controller), which transfers control to a specified program (interrupt handler) for each kind of interrupt. The addresses of the programs are kept in a table (interrupt vector) whose location is known to the hardware.

Priv'd instructions, executable only from "supervisor mode", used for:

- memory management (OS can R/W any memory, user can only R/W own).

- enable/disable interrupts
- switching processes
- i/o
- halt

SVC - SuperVisor Call. The mechanism that lets user programs invoke functions of the operating system. Usually invoked via an interrupt.

SYSTEM KERNEL (OR NUCLEUS) - OS LAYER 1

Most critical and machine-dependent piece of software in system. Usually written in assembler (exceptions: Unix, Burroughs/Algol). Should be as simple as possible, for efficiency and correctness. Provides environment in which processes can exist. The kernel consists of:

FIRST-LEVEL INTERRUPT HANDLER:

Determines source of interrupt, saves context (process info and PC) of current process, dispatches to appropriate handler via the dispatcher. Interrupts may come from external devices, internal conditions (like arithmetic over- or underflow), system calls (typically invoked via "illegal instructions"), or the system clock.

Most systems provide an "interrupt vector", a table of interrupt handlers, one for each kind of interrupt. Some systems further classify interrupts by priority, so that a lower-priority interrupt (e.g. from a communication line) cannot interrupt a higher-priority one (e.g. from disk). (On some machines, like many PCs, this can be all done in hardware.)

DISPATCHER:

Low-level scheduler, entered upon any interrupt (invoked by 1st-level interrupt handler), starts interrupt routine, returns to process if still most suitable to run, otherwise saves environment of process, retrieves environment of new process, and transfers control to it. Next process is selected by high-level scheduler from the process queue.

INTERPROCESS COMMUNICATION PRIMITIVES:

(skipping all this...)

INPUT/OUTPUT (LAYER 3)

Device drivers, buffer management, "access methods"...

A device driver is a portion of the operating system software that embodies specific knowledge of the control mechanisms for a particular device. It provides a uniform "interface" between the device and higher-level software, allowing the higher-level software to issue input/output requests in a standard format, like "read sector abc and put the data at address xyz", without specific knowledge of the hardware, slot, registers, etc.

Device driver generally operates at interrupt level -- gets data from device, puts it in buffer, clears the interrupt condition, and dismisses the interrupt, allowing the previous program (or the scheduler) to resume control.

FILE SYSTEM (LAYER 4)

Uniform set of operations on files, independent of device. Online disk storage - data always accessible, allowing users to share programs, libraries, data. Directory provides a catalog of available files.

File system allows PROGRAMS (not users) to invoke following functions:

- file creation & deletion
- access to files for reading & writing
- automatic management of disk space
- reference to files by symbolic names
- failure protection (redundancy, robustness, backup mechanism)
- sharing and protection
- supervisor calls for file i/o at "logical" level: open/close, buffered read/write character, record, line
- Other devices on which data resides (tape, networks) can be treated as files.

Directory structure: (DRAW PICTURE)

- Storage allocation table
- Directory blocks: name, disk address, protection, size, date, write-bit, use count, author, account, etc etc...
- Flat file systems, 1-level, 2-level, multilevel systems.

File operations: opening & closing, file descriptors, reading & writing:

- Open:
    - find actual device look up name in directory check access (protection, multiple access, etc) create file descriptor (fd): (LOOK IN FILE.H...) tell how this is
        - file name, is similar to pointer to device descriptor/driver network operation... length of file location of start of file, location of next character (or block), mode of access (read, write, append).

  Read/write: operations refer to fd. Buffering, etc, is invisible. (show connection to device driver)

- Close: flush pending output, fix write bit & use count, release fd.

Usually the user program sees only the file system. The file system, in turn, calls on the device driver:

```
USER       FILE SYSTEM            DEVICE DRIVER
 read()     Get char from buffer    Get sector or char if buffer empty
 write()    Put char in buffer      Put sector or chars if buffer full
```

RESOURCE ALLOCATION & SCHEDULING (LAYER 5):

(skipping all this...)

SIMPLE INTERRUPT-DRIVEN I/O EXAMPLE:

Let's look at a simple example of how the OS handles a simple terminal interaction. User program "X" is running on the computer. The program counter (PC) indicates the next instruction in X to be executed.

Suddenly, a user types a character at a terminal.

The character arrives at the UART, which puts it in its holding register and raises its "input ready" flag.

The interrupt controller notices the flag and:

- Gets the address of the UART's device handler (DH) from the interrupt vector
- Saves the current PC
- Puts the DH's address into the PC, so the next instruction starts the DH

The DH copies the character from the UART to an internal buffer.

The DH pushes the UART's flag down, so the UART knows the character was read, and the interrupt controller knows the interrupt was handled.

The DH "dismisses" the interrupt. The saved PC is restored, and program X is continued where it left off.

When the user program wants to read a character from the terminal, it gets the next unread character from the internal buffer.

A SLIGHTLY MORE COMPLICATED ILLUSTRATION OF OS SERVICES (many steps skipped)

Suppose a user of a timesharing system wants to copy some text from her terminal into a file. The user first issues a command, like

"COPY TTY: FOO.BAR" (DEC operating systems), or "cat > foo.bar" (UNIX)

What really happens? Let's look at what goes on just after user enters this command.

The shell calls the OS file service to create a new, empty file FOO.BAR, and open it for write access.

. File service attempts to create a new file called FOO.BAR.

... But to do this, file service must call upon the disk device driver to read the directory blocks.

..... But to read directory blocks, the disk device driver must know the details of disk operation, how to feed read-block commands to the disk controller, etc. The device driver then waits for a completion interrupt from the disk controller, and then returns a pointer to the block to the file service.

... The file service continues to read directory blocks until it has found FOO.BAR, or it has read all the directory blocks and did not find FOO.BAR.

... If FOO.BAR existed previously, file service must free all the data blocks previously occupied by FOO.BAR, by setting FOO.BAR's length to zero, and marking the old blocks free in the storage allocation table. If FOO.BAR did not exist previously, a new directory entry for an empty file FOO.BAR must be created.

..... Disk service must be called to update the appropriate directory and storage allocation disk blocks.

. Assuming all went well (no i/o errors, etc), file service returns a file descriptor (fd) for the new file for use by the shell.

2. The shell issues an input request to the terminal for one character, and waits for the character to be typed.

. The input request activates the terminal device driver, which sets up an IORB for the terminal, and waits for completion.

. User's program waits (is blocked).

3. User gets a phone call and doesn't type any characters for a while. Other users are making demands upon the computer. As this occurs, the scheduler begins to activate the other users' processes.

. The scheduler must call upon the low-level dispatcher in the nucleus in order to activate other users' processes.

4. Eventually, so many other active processes want to use the computer that our user must be moved to secondary storage.

. The scheduler calls upon the memory manager to determine which process to replace. It chooses our user's process because it is in a wait state.

. It then calls upon the disk driver to transfer our user's memory pages to disk.

. Then it calls upon the dispatcher to activate the next process.

5. Our user gets off the phone and types a character.

. The character causes an interrupt, which is caught by the terminal device driver, which in turn signals completion to the user's process.

. However, the user's process was not in memory. But the signal caused the wait state to be cleared, so the process is eligible to run, so the scheduler calls calls upon the memory manager to find space for the user's process pages in memory. But since there is no free space, the memory manager must choose another process to "swap out", and calls upon the disk service to do so. Then it calls upon the disk service to read our user's process pages into the newly freed memory.

. The character which the user typed is now copied into the file system's terminal input buffer.

6. The shell now reads the char from the buffer and writes it to the disk file FOO.BAR.

. The OS file service saves the character in a disk block buffer.

7. The process repeats until the file service's disk block buffer is full. At that point the file service calls upon the disk driver to read the disk storage allocation table. From this table, a free block is selected, marked as used, and the storage allocation table and the new data block are written to disk.

8. The process repeats, character-by-character, block-by-block, until and end-of-file is (somehow) signalled from the terminal. The shell calls upon the file service to close the file. File service calls upon disk service to write out the last data block, then it sets the file creation date and file length in the directory block and calls upon disk service to write that out too.

2. COMPUTER NETWORKS

There are two kinds of data communication: point-to-point, and networks. We have more than 100 years experience with point-to-point communication: telegraph, teletype, terminal-to-computer, etc.

```
Point-to-point                         Network
  A--------B              A-----------B       J--------K
                          | \       / |       |        |
                          |   \   /   |       |        |
                          |     C     |       H--------I
                          |   /   \   |       |
                          | /       \ |       |
                          D           E--------F--------G
```

Point-to-point communication is still the predominant mode of communication. And the lessons we've learned from it have strongly influenced the design of networks.

TERMINAL-TO-HOST COMMUNICATION

Much of today's data communication follows the terminal-to-computer model. We still have a large number of real terminals connected to computers. What are some of the issues in terminal-host communication?

The terminal provides a certain set of functions --

- Generic functions:
  - Display and transmission of ASCII characters
  - Response to CR, LF, FF, HT
- Response to terminal-specific escape sequences (no universal standard)

And the host computer provides its own set, through its "console driver" --

- Reception and transmission of ASCII characters
- Response to host-specific character sequences:
  - Commands (shell)
  - Control characters (interrupt/cancel, editing, status inquiry, etc)
- Knowledge of how to control the screens of specific terminals.

Control characters (like Ctrl-Y or Ctrl-C) often cause special actions and are therefore not treated as data.

When the host wishes to control the appearance of the terminal's screen, then the terminal and the host must agree about what escape sequences to use. This means the host must know the terminal type. But how does it know?

1. User can tell it (but what if user doesn't know either?)
2. It can be entered into system tables (terminal on port 23 is a VT100) (but what if this changes?) (and what about dialups?)
3. Terminal can include a "what are you?" inquiry mechanism (but not all

terminals do).

If the host thinks it's controlling a different kind of terminal than the user really has, the result will be incomprehensible, fractured screen displays.

Hosts or applications may keep tables of "generic" terminal functions for each particular terminal (clear screen, clear line, etc). Many ways to do this -- in the shell and in each application, in the console driver with calls available to user programs, or in libraries like Unix "termcap" and "curses".

Terminal	Clear Screen	Clear to end of line	Highlight
VT52	ESC H ESC J	ESC K	(none)
VT100	ESC J ESC [H ESC [J	ESC [K	ESC [7 m
C-100	Ctrl-L	ESC Ctrl-U	ESC G

Terminals are intended purely as "interfaces" between a human being and a computer. The terminal reacts only to the user's keystrokes or characters arriving from the host computer, and takes no actions "on its own". For its own part, the host might make certain assumptions about terminal connections: e.g. that input will be relatively slow, because it's coming from people's fingers (the fastest typist can only do about 120 wpm = 600 cpm = 6000 bpm = 100 bps), but that output can be voluminous and fast. Sometimes the console drivers and associated buffers are designed on this assumption.

PC-TO-HOST COMMUNICATION

Increasingly, people use PCs in place of terminals, because, as general-purpose computers with built-in keyboard and screen, they can be programmed to emulate terminals, in addition to all their other functions.

Not only can a PC emulate all the functions of a particular terminal (or several different terminals), but it can surpass the terminal's capabilities in various ways:

- keys can be redefined
- screen memory can hold previous screens for rollback
- screens can be copied to the PC's disk ("raw download")
- PC disk data can be copied to the communication port ("raw upload")
- various translations can be done (e.g. European character sets)

But terminal emulation does not provide an error-free link between the host and the mainframe, any more than a terminal-host link does. Thus data transfer is necessarily a risky business: data can be lost in transmission or corrupted by interference, undetectable by the receiver.

Data can be lost not only because of transmission problems, but also because the host computer simply was not designed to receive characters in a continuous stream from a terminal.

If non-error checked data transfer can be done by a terminal emulation program, then how can error-free data transfer be accomplished? The problem is that a terminal emulation program is self-contained, and talks to the host as if it were a terminal. No special programming of the host is required. But terminals have no provision for error recovery, so the host has no way of dealing with a terminal on this level.

To transfer data reliably, between two independently functioning devices of possibly differing speeds and other characteristics, over a possibly hostile communication medium, some level of cooperation is needed between the two computers which is higher than the terminal-to-host model: a set of rules, procedures, and formats by which the two computers exchange information to achieve a desired objective. In other words, a "protocol".

An error correcting protocol requires cooperating processes (programs) on each end of the connection -- one on the PC, one on the host -- to format, encode, and decode special messages and to request retransmission of damaged messages. These messages contain the data to be transferred, plus error-detection and sequencing information, so the information arrives intact and in order.

But there is a particular problem on the host end. Since the PC is connected to the host through its console driver, it is not possible to send it arbitrary data characters. For instance, if the data contains a Ctrl-Y, then sending it to a VAX/VMS host could cancel the operation unexpectedly, resulting in a lost or incomplete file. The ability to send arbitrary data across a communication link is called TRANSPARENCY, and the connected from a terminal or PC to a timesharing host is generally NOT transparent.

Text files are a special (and common) case, i.e. files that contain only printable characters, CR, LF, FF, and HT. Most host computers accept all of these characters as data.

An important part of a PC-host file transfer protocol is to overcome these transparency problems.

Once a PC-host protocol has been designed, it can also be adapted to operate PC-to-PC and host-to-host.

Two such protocols are Xmodem and Kermit, which we will look at in the coming weeks.

Terminal-to-host connections were the predominant mode of communication for many years. The first host-host connections were built upon them, with one host tricking the other into believing it was a terminal. But terminal-to-host connections have certain intrinsic limitations:

- They operate only point-to-point, and so can connect only two computers.
- Only one user can use a particular point-to-point link at a time.
- They operate at relatively low speeds.
- Connections must be made "manually" each time they are to be used, e.g. setting communication parameters, dialing up, logging in, etc.
- They are error-prone unless a special error-correcting program is run on each end (each using the same protocol), and such a program is not always available.
- Connections are not transparent to all patterns of data.

NETWORKS

A network is a physical arrangement, used in conjunction with a set of rules, formats, and procedures, that allows two or more independently functioning computers to transmit data -- characters, messages, files, records, commands, transactions, etc -- to each other in a useful form, usually over special dedicated communication devices and media.

Standards and protocols must exist at many levels to handle problems like error detection and correction, elimination of lost or redundant data, allowance for multiple users to share the same communication channel, data conversion, etc. These protocols are usually implemented in software.

Networks alleviate the restrictions on terminal-to-host connections:

- They can connect more than two computers together.
- Many users can share the same communication medium.
- They operate at relatively high speeds.
- Physical connections are usually dedicated, always ready to be used.
- They include error detection and correction for all applications.
- A wider variety of services may be available.
- Connections are generally transparent to all bit patterns.

Although data communication and networks are the topics of this course, it should be noted that they can be great wasters of time and money. Why do you, or does your organization, need data communication? How often must data be transferred, and how much of it? What resources really need to be shared?

Data communication professionals should always be sensitive to the costs and benefits of any given approach. Even though a network is a very hot, prestigious commodity, the cost and pain of network installation is not always worth it. Always consider the alternatives: terminal-to-host communication, file transfer protocols like Kermit and Xmodem, exchange of magnetic media (e.g. walking down the hall with a floppy disk, mailing a tape), phone calls, and even face-to-face human interaction.

For example, let’s compare networks with terminal-to-host connections:

- Networks are more expensive, requiring special, often proprietary, hardware and software.
- It is not always possible to put computers made by different companies on the same network.
- Networks sometimes have stringent distance limitations.
- Networks are more complicated to install, maintain, and administer.
- Functions specific to the terminal:
- There is always the need to communicate with some computer that is not on the network.

But now let's look on the positive side. Where terminal-to-host communication programs generally offer only terminal emulation and possibly file transfer, networks generally offer a wider range of services. These might include:

- Virtual terminal service. Similar to terminal emulation, except the host machine does not need to know what kind of physical terminal the user has, what its baud rate is, its character set, its control sequences, etc, and because the session is error-free and synchronized.
- File transfer. May be invoked explicitly by running a program (similar to Kermit or Xmodem), or in some cases implicitly, simply by referring to a remote file by using special syntax in its name (e.g. in DECnet, "COPY NODEA::FOO.BAR NEW.BAR").
- Electronic mail. Really a special case of file transfer, but the file is a message to be delivered to particular users on local or remote systems for later reading. Electronic mail systems allow messages to be replied to, forwarded, etc.
- Remote file access. Treating remote files as if they were local, transparently to all applications software (e.g. "EDIT NODEA::FOO.BAR"). Typically implemented within the operating system's file services -- open, close, read, and write functions.
- Distributed file systems. Files are located on many machines and are accessed as if they were local, transparently not only to the application software, but also to the user (no special syntax required in filename).
- Shared devices. Printers, disks, tapes, dialout modems, etc, can be shared by multiple machines on the same network. The network gives the illusion that the device is local, transparently to application software.
- Shared databases. Users on different machines can query, perhaps even update, the same database. Transaction processing (like airplane reservations) are a good example. Of particular use in a network is a "name server" which is often queried implicitly when a user refers to the name of a remote system or user.

The fancier services are possible because the network support is integrated into the operating system. Thus disk and printer drivers, even portions of the file system, may be using the network rather than the normal local devices.

NETWORK TOPOLOGIES

To connect a computer to a network, you need:

- A physical interface between the computer and the communication medium, generally a circuit board that plugs into the computer, and has a connector for the network cable.
- Software that controls the physical interface (a device driver).
- Software that implements the same network protocols as the other computers on the network
- And you must be in physical proximity to the network medium.

The physical arrangement of the network is called its "topology". There are several basic kinds:

Star all nodes connected directly to a central hub (like terminal network). Bus
all nodes connected to a single, shared cable (like Ethernet). Ring
all nodes connected to a circular wire (a loop, like IBM Token Ring). Mesh
nodes connected to each other in arbitrary ways (most wide-area nets).

In all but the mesh arrangement, it is a fairly straightforward task to get a message from one node to another.

Particular technologies are not necessarily tied to particular topologies. For instance, Ethernet on coax cable is a bus, but there can be star-wired Ethernet based on twisted pair or optical fiber. In some ways, the choice of technologies is the crucial one, because this dictates what boards you will plug into your computers, and what software you will run to interact with these boards. It is often possible, at a later date, to replace the network cabling with something entirely new, e.g. coaxial cable with fiber, leaving a large investment in controller boards and software untouched.

Data networks are divided into two major kinds: local area networks (LANs), and wide area networks (LANs).

LOCAL AREA NETWORKS (LANS)

Local Area Networks are usually found within a building or campus, and have bus, ring, or star topology. Since wiring is on-premisis and limited distance, special (expensive) media can be used -- coax cable or optical fiber -- for high bandwidth (3-100 Mbps)

> "bandwidth" -- information transferred per unit time, e.g. bits per sec

and good noise immunity.

Typical services on PC-based LANs include shared disks, shared printers, modem pools, shared databases, even remote procedure calls (programs with subroutines running on separate computers). These applications require very fast, relatively noise-free connections, and LAN software is generally designed on this assumption. When larger hosts are involved in a local net, wide-area functions like terminal service (remote login), file transfer, and mail may also be supported.

```
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
     |     |     |     |     |     |     |     |     |
     A     B     C     D     E     F     G     H     I
```

In some local area networks, all nodes share the same transmission medium. How can they do this without interfering with each other? There are two basic methods:

<u>Frequency Division Multiplexing (FDM)</u>

In "broadband" networks, the cable's bandwidth is divided into many channels, of different frequencies, just like CATV cable can carry many channels simultaneously. But TV is a receive-only device. In a network, each device has separate send and receive frequencies. Thus, each channel is really two channels, and the network has a "head end" to shift between the sending and receiving frequencies of each channel. To talk to devices on other channels, special frequency converters are required for each pair of channels to be interconnected.

Broadband networks offer high overall bandwidth (in the 300 MHz range), over exactly the same medium that cable TV uses. But the bandwidth is chopped into relatively slow channels, about 6 MHz each, and those are often further subdivided. Management of a broadband network is complicated, involving frequent "sweeps" and tuning, channel management and assignment, installation of more and more frequency converters, etc. The trend seems to be away from broadband networks.

Time Division Multiplexing (TDM)

In "baseband" networks are the opposite of broadband. The overall bandwidth is lower -- typically in the 5MHz to 20MHz range. But each station gets to use the entire bandwidth, rather than a small channel. This means that all stations use the same frequency. Therefore they must transmit at different times. This is called time division multiplexing.

In the simplest TDM scheme, every station gets a fixed-size time slot. But this is wasteful. The two most common TDM schemes used in LANs today are those associated with Token Ring and Ethernet.

Token ring networks are espoused by IBM. Current IBM Token Rings run at 4Mbps. In these networks, stations are connected in a ring, or "daisy chain". A "token" is circulated around the ring; only the node that is in possession of the token may transmit (for a limited amount of time). When transmission is done (or if there is nothing to transmit), the token is passed to the next node. A drawback of a true ring is that failure of any particular node (including turning it off) will break the ring. For this reason, real rings are wired in star fashion, with a passive wire center which automatically closes a relay across non-functioning nodes.

Ethernet is a more widespread LAN technology, having been adopted by dozens of major companies. Ethernet requires special coaxial cable (not CATV cable), and runs at 10Mbps. Ethernet is a "broadcast" network with no central control -- each node may broadcast at any time. All nodes receive all messages simultaneously. Any node may broadcast when the medium is not in use. But what happens when two nodes decide to broadcast at the same time? The messages interfere with each other, and both become corrupted. A typical medium access mechanism (the one used in Ethernet) is called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). This means: before sending, check the medium for a carrier signal. If it's there, wait till it goes away before transmitting. Then, if during transmission, interference is detected (listen while sending), stop immediately, and "back off" a random amount of time, and then repeat the process.

Token ring advocates claim the benefit that the network can never become swamped, and will therefore always behave predictably. This is important in realtime applications, like robot control on the factory floor, where actions have to happen at the right time. Ethernet advocates claim that Ethernets rarely saturate, and if they do, they can be divided into smaller Ethernets at little performance penalty. We'll see how to do this in a couple weeks.

TDM LANs like Ethernet and Token Ring have a particular characteristic that frightens some people away. Since all messages go to all nodes, there is nothing to prevent node A from reading node B's messages. Although network interfaces are generally configured not to accept other people's messages, they can also be configured to read all messages. In fact, this is a desirable feature to help the network manager track down problems. But since "unauthorized" people may be able to configure their boards this way (this is called "permiscuous mode"), these networks are normally not used in situations where security is an issue, at least not without a relatively foolproof encryption scheme.

WIDE AREA NETWORKS (WANS)

Wide Area Networks (also called "long haul" networks): Bigger than a building or a campus. Generally use communication media available only from an external organization, or "common carrier" -- dedicated leased phone lines or satellite transmission, so the medium is usually not under direct control of the owner of the computers. Bandwidth is in the 9.6Kb-1.54Mb bps range, compared to 4-10 Mbps for typical LANs, and the connections are more susceptible to interference that LAN connections.

Typical services include interactive terminal sessions, file transfer, electronic mail, remote job entry, etc. Topology is typically mesh. Well known examples include Telenet, Tymnet, Arpanet, Bitnet.

```
    D ----- E ---- I         K   M --- P ---- Q
   / \     /                 |  /       \
  /   \   /                  | /         \
 /     \ /                   |/           R
A ----- B ---- F --- H ----- J ---- L
 \     /       |   /         |      |
  \   /        |  /          |      |
   \ /         | /           N ---- O
    C -------- G
```

In wide area networks, when there is a path between two nodes, it is for their exclusive use -- they have control over it, much like a connection between a PC and a computer. The two nodes don't have to worry about sharing the physical connection. This type of network is sometimes called a "point-to-point" network, because it is composed of many point-to-point connections.

Since the communication media used in wide-area networks are subject to interference and are relatively slow, the communication software must be designed to detect and correct errors efficiently and to make the most efficient use of the medium, especially when satellites are involved.

NETWORKS OF NETWORKS

In practice, many of today's networks consist of multiple interconnected networks. For instance, at Columbia University there are many departmental LANs, all connected to a campus-wide "backbone" Ethernet cable. Some of the big computers on this backbone serve as "gateways" to wide area networks like BITNET, CSNET, CCNET, etc etc., which in turn get to hosts at other universities that are gateways into their own local nets. So it is possible for a user of a PC on a token ring at Columbia (with appropriate software) copy a file from the disk of a PC on a local Ethernet at Stanford.

BASIC DATA TRANSFER

The most fundamental task of a network is to deliver data correctly. But since the data can be corrupted in transit, e.g. by electrical interference, the receiver must have a way of knowing that what it got is what was sent. The method most commonly used is this:

1. Break data up into chunks of manageable size, like 100 or 1000 bytes.

2. Add up the numeric values of all the bytes (or use some other arithmetic combination of their values). Append the result to the data itself.

3. Uniquely mark the beginning and end of the data.

4. Send it.

The result is a delimited, error-checked piece of data, called a "packet" or a "frame" or a "block". The receiver knows exactly where it begins and ends, and can differentiate the data from the block check. The receiver makes the same calculation on the data that the sender did, and checks for agreement. If the block checks disagree, the frame is known to be in error, and retransmission can be requested.

ROUTING

In a point-to-point network, how do messages find their way through through the maze to their final destination? For a small number of nodes, you could connect each node to all others:

(Draw picture for 1, 2, 3, 4, 5...)

This is impractical for a large number of nodes (combinatorially speaking, this is sampling without replacement, n things taken 2 at a time, or

binomial coefficient of n and 2 = n!/((n-2)! x 2!) = n(n+1)/2

For large n, this number is ridiculous (n = 10, need 45 wires; for 100 need 4950, for 1000 need 499500), hence a "fully connected network" is usually impractical.

Not connecting each node to every other node means that some kind of "routing" of data has to occur when two nodes that are not adjacent must communicate; if a node receives a message that is not for itself, it relays it to (or in the general direction of) the intended recipient.

MESSAGE SWITCHING VS PACKET SWITCHING

What is the data that we are transmitting through the network? Often, it is some kind of message, or file. There are two major ways to transmit data through a routing network: in whole messages (files, electronic mail messages, etc), or in smaller pieces.

Message switching means that a message -- a logically complete piece of data -- is sent as a whole from one node to the next. Each node stores the whole message until it has been successfully transmitted to the next. This is called "store and forward" or "message switching". Early message-switching systems were based on teletype machines and torn paper tape.

Message switching is akin to batch scheduling on a mainframe computer; each user's "job" must wait in a queue for its turn, then it gets the whole CPU for as long as it takes for the job to complete.

The major disadvantage of message switching is that a user's message must wait in a queue to be sent. A very short message may have to wait hours for a long message to complete. Therefore, response cannot be in "real time", and interactive network applications (like terminal connection) aren't possible.

BITNET and USENET are the primary examples of message-switching networks.

Breaking user's messages up into little chunks, each chunk tagged with source/destination/user identification, allows many users to share the transmission medium by "interleaving" these chunks, very much like many users may share the single CPU of a timesharing computer; for instance, long files may be sent at the same time a user is conducting a two-way character-oriented dialog. The little chunks are called PACKETS, and this technique is called PACKET SWITCHING.

Arpanet and the public networks (Tymnet, Telenet, Datapac, etc), along with proprietary networks like DECnet are examples of packet-switched networks.

THE LAYERED MODEL

We've discussed, in a rather unstructured way, some properties and problems and problems of networks. In fact, this kind of haphazard approach was the basis for many early networking efforts. After many years of effort, millions of lines of software coding, some valuable lessons were learned.

(blah blah)...

Potential problems in getting a message from one computer to another within a network can be resolved by formulating a set of rules, procedures, and transmission formats specially suited to the particular problem -- in other words, a protocol. If we have formulated the problems well enough, we can devise protocols that are relatively independent of each other. The protocols for each problem can be combined together to form a hierarchy, in which each protocol depends upon the one "beneath" it to do its job. Each protocol in this hierarchy is called a "layer".

For example -- I go to the showroom to buy a car. The protocol for buying a car is to negotiate with the salesperson, and then give her some money. I don't care how she got the car. She calls upon the next lower layer to do that: the trucker. The trucker, in turn, picks up cars from the factory, without caring how they were made. At the factory, they put together the raw materials to form a car. The raw materials, in turn, are delivered by planes, boats, and trains, from mines, quarries, and oil wells. Each layer does its own job, unconcerned with what the other layers do, but caring very much how it "interfaces" with the adjacent layers (getting paid, etc).

There are many different network designs, and there different ideas about how many layers there should be, and what each should do. After break, and for the rest of the course, we'll be looking at a model which is almost universally used for describing any network, even though very few real networks today follow this model.

THE ISO OPEN SYSTEM INTERCONNECTION REFERENCE MODEL

Packet switched networking began in the late 1960s with research efforts like the DoD-sponsored ARPAnet, and corporate networking efforts like IBM's SNA and DEC's DNA (DECnet), both appearing on the market initially around 1974 after some years of research and development. By 1978, feelings ran high that incompatible networking techniques were creating barriers between systems of different manufacturers, and the International Organization for Standardization (ISO) formed a Technical Committee (TC) charged with developing a "reference model" (RM) for "open systems interconnection" (OSI).

The ISO is a worldwide voluntary federation of national standards institutes established in 1947. Currently, about 88 countries participate in about 160 technical committees and some 1900 working groups and subcommittees, covering everything from screw threads to nuclear energy. The USA member body of ISO is ANSI, the American National Standards Institute.

OSI Technical Committee 97, formed in 1961, deals with computer technology and office equipment. Two subcommittees (SC) deal with data communications: SC6 for telecommunications and information exchange between systems, and SC21 for OSI. By 1980, within 2 years of its formation, SC21 had formulated ISO Draft International Standard (ISO/DIS) 7498, "Information Processing Systems Interconnection - Basic Reference Model". It has since been adopted by the CCITT (International Telegraph and Telephone Consultative Committee) as Recommendation X.200 (CCITT Recommendations are tantamount to standards).

The ISO Open System Interconnection Reference model (sometimes called ISORM) specifies a 7-layer communications architecture allowing "open systems" to communicate. The model is general in nature, and does not specify any particular protocol or technology at any layer. It only specifies the functions of each layer and the interface between each layer and the adjacent layers on the same system.

In Europe, ISORM has already taken firm root. Most networks are based on X.25, an ISORM-conformant networking architecture (X.25 will be explained later). In the USA, X.25 is used in popular "public" networks like Telenet and Tymnet (and Datapac in Canada), and proponents of other architectures including vendors like IBM (SNA) and DEC (DNA), as well as government agencies like DoD's ARPA (TCP/IP), have all announced conformance with the ISORM as a goal in future networking development. These architectures will gradually be supplanted by "open" ISO architecture, allowing diverse heterogeneous systems to interconnect over a potentially vast worldwide network.

Support for ISO/OSI architecture will probably become mandatory in all networked computers purchased by the US Government, under the Government OSI Procurement specification (GOSIP). General Motors Manufacturing Automation Protocol (MAP) and Boeing Technical and Office Protocols (TOP) are both OSI-based, and are receiving much attention in the press and from computer vendors like IBM and DEC.

There are also do-it-yourself networks (that arose in the absence of standards bodies or corporate decrees) like FIDO, UUCP, BITNET, CSnet/Phonenet, each with its own special story...

WHAT IS A LAYER?

In theory, a layer is a protocol for doing a particular, well-defined, circumscribed function, along with the hardware and/or software that interprets (implements) the protocol. In fact, the functions of the ISO layers tend to overlap; the description of each layer reads very much like the descriptions of all the others. By searching for the minor differences, you can extract the essence of each layer.

Whatever its function might be, an OSI layer is an "entity" that:

1. communicates with its peer layer on another system,

2. provides services to its superior layer on the same system, and

3. calls upon the services of its inferior layer on the same system.

In OSI terminology, a particular layer, N, is embodied by an "(N)-entity". The layer immediately above it is an (N+1)-entity (a "service user" of the (N)-layer), and the layer below it is an (N-1)-entity (a "service provider" to the (N)-layer).

```
  SYSTEM A                                    SYSTEM B
+-------------+                             +-------------+
| Layer (N+1) |                             | Layer (N+1) |
|   ^         |                             |   ^         |
+---|---------+                             +---|---------+
|   v         |  Peer-to-peer communication |   v         |
| Layer (N)   |<--------------------------->| Layer (N)   |
|   ^         |                             |   ^         |
+---|---------+                             +---|---------+
|   v         |                             |   v         |
| Layer (N-1) |                             | Layer (N-1) |
+-------------+                             +-------------+
```

Programmers can think of a layer as a procedure (subroutine) that is called by its upper layer with some data and other parameters, that sends a message to its peer layer on another system by calling another procedure (i.e. the next lower layer), and returns some kind of data or status code.

```
                 +-----------+
(N+1)-Layer      | (N+1)-PDU | (N+1)-layer delivers its PDU to the (N)-layer
                 +-----+-----+
                       |
                       |
(N)-Layer              V
+---------+      +-----------+
| (N)-PCI |----->|  (N)-SDU  | (N)-layer adds its own protocol information...
+---------+      +-----+-----+
                       |
                       V
       +---------+-----------+
       | (N)-PCI |  (N)-SDU  | to form an (N)-PDU, which it delivers...
       +---------+-----------+
                       |
                       |
(N-1) Layer            V
       +---------------------+
       |      (N-1)-SDU      | ...to the (N-1)-layer, which sees it as an SDU
       +---------------------+
```

A message sent from peer to peer is called a protocol data unit (PDU). It consists of the block of data (called a Service Data Unit, or SDU) provided by the (N+1)-entity, to which the current (N)-layer adds some Protocol Control Information (PCI). In order to transmit its PDU, the (N)-layer passes it to the (N-1)-layer. The process repeats until the lowest layer is reached; layer N's PDU becomes layer N-1's SDU:

(picture here)

Each layer normally communicates with its peer in three phases: connection establishment, data transfer, and connection release. Each layer is allowed to know the interface only to its adjacent layers. Thus, each layer hides the details of non-adjacent layers from its adjacent layers; it is an "interface" in the true sense.

Most layers provide the following services:

- connection establishment
- connection release
- normal data exchange

- expedited data exchange
- synchronization
- error detection
- error recovery
- exception reporting

Each layer has its own address, its own protocol, its own formats.

CONNECTION-ORIENTED VS CONNECTIONLESS

Most OSI protocols are connection-oriented. That means, they proceed through three phases: connection establishment, data transfer, and connection release. When a connection is opened, a database is created for the connection, containing information like the address of the peer, the current sequence number, the flow-control status, and a retransmission buffer for the most recent (or several most recent) messages. For the rest of the connection, this database is referred to very efficiently using shorthand "pointers", much like the file numbers that programmers use after opening a file. Connection-oriented protocols assure correct and complete peer-to-peer communication. Correctly received messages are acknowledged. Retransmission of damaged or missing messages can be requested by sequence number, and duplicates can be detected and discarded based on the sequence number.

A connectionless protocol does not establish or maintain any relationship between individual data transfers. There is no sequencing, no database, no acknowledgement. How, then, can a connectionless protocol possibly work? On its own, it can't. But it can occupy one or more of the layers in a complete protocol stack. So long as there is a connection-oriented protocol above the highest-level connectionless protocol, it can take care of sequencing and retransmission.

Why would we want a connectionless protocol? Several reasons...

- On a clean, single-user at a time medium, like Ethernet, there's not much chance of messages becoming misordered, so the overhead of maintaining a connection at the datalink or network level can be avoided.
- In a packet-switched network, it is not possible to have a connection-oriented network layer, because packets will take different routes.
- If the network is totally reliable, then the transport protocol could connectionless.
- etc...

THE OSI LAYERS

So what does all this gibberish about (N)-layers really mean? Basically, it means that each layer is concerned only with its own functions and its own protocol, and knows nothing about the protocols used by the other layers. In fact, a particular layer only knows its own protocol, and what services are provided by the layer immediately below, how to invoke them, and how to get the results back.

The fact that each layer does not look at the data given to it by the superior layer results in the most valuable property of layered protocols: any layer can be easily replaced. So long as a layer does what it is supposed to do and offers the standard interface to the adjacent layers, its internal operation -- including the protocol it uses with its peer layer -- is an "implementation detail". This allows the physical medium to be changed without affecting the network applications, it allows the applications to be changed without concern for the details of the network, and it allows the operation of the network itself to be changed without affecting the upper or lower layers.

In practice, the definition of a particular layer is more or less precise depending on its position in the hierarchy -- the lower the layer, the clearer the definition; the higher, the more vague.

The seven OSI layers are as follows:

```
    +--------------+
 7. | Application  | The meaning of the data, the actions to perform, etc.
    +--------------+
 6. | Presentation | The format of the application data
    +--------------+
 5. | Session      | Application-to-application dialog control
    +--------------+
 4. | Transport    | End-to-end (host-to-host) communication
    +--------------+
 3. | Network      | How to get from end to end, hop-by-hop
    +--------------+
 2. | Datalink     | How make one hop, how to get from node to node
    +--------------+
 1. | Physical     | How to use the transmission medium
    +--------------+
```

The ISO layers were chosen according to certain principles, including:

- Layers are defined so as to group similar functions together.
- Layers are defined so as to minimize interactions across layer boundaries.
- Layers are defined to allow different protocols to be used within a layer without affecting the layer service definition.
- The number of layers is kept to a minimum consistent with the above principles.
- Each layer adds to the services of the layer below.
- Each layer requires one or more distinct peer protocols.
- Each layer has boundaries only with the immediately upper and lower layers.

These principles allow the software that implements each layer to be as simple as possible, and easily replaceable.

Unfortunately, there is also an unstated principle, resulting from the fact that the OSI committee is comprised of representatives of major corporations that have a large stake in existing products. This principle is that the model must fit these products. For this reason, we will see that the actual definitions of each layer are very complicated and tend to overlap.

Now let's take a very brief and simple look at the functions of the seven layers of the OSI model. In the coming weeks, we'll examine each layer in gruesome detail.

7. THE APPLICATION LAYER

...is concerned with the meaning and function (semantics) of the data being exchanged between two partners. In other words, the application layer does the actual work that the user needs done, requiring the user to know little or nothing about the underlying network or the remote system. Examples include file transfer programs, electronic mail programs, virtual terminal programs, remote user lookup programs, etc.

To use the file transfer example, the application layer may send messages to its peer like "Here comes a file whose name is FOO.BAR", and "Here is the contents of the file", and "This is the end of the file". But suppose that one system uses a different character set, or has different file formats. Then what?

6. THE PRESENTATION LAYER

...is concerned with the appearance and format (syntax) of the data. This is where character set translation might be done, or encryption and decryption, compression/decompression, perhaps (someday) even translation between different natural languages. The presentation layer provides a "common intermediate representation" of data, transparent to the application.

In the file transfer example, suppose one host is ASCII-based, and the other uses EBCDIC. The messages exchanged at the application level will be meaningless unless ASCII/EBCDIC translation is done -- files will have the wrong name, and their contents will be gibberish. Furthermore, one system might store files in stream format (lines with CRLF terminators) and the other in card-image format (80-column records). These conversions are handled by the presentation layer.

OK, so now two network applications can exchange meaningful messages, whose form and purpose are clear. But suppose multiple applications are running on the same system simultaneously. Which system gets which message? How do we keep messages from getting mixed up?

5. THE SESSION LAYER

The parameters of concern to the OSI session layer involve dialog management: whether the dialog is full duplex, half duplex, simplex. The session layer also may provide "message bracketing" or "quarantine" (grouping of messages into a single "atomic" unit -- useful in database transactions) and other message management functions. And it may be used to ensure that a session can be continued across an interruption in the underlying transport service.

OK, so we are able to conduct a dialog with a remote computer, but how do the messages actually get there?

4. THE TRANSPORT LAYER

...is concerned with the host computer where the partner process is running; it is responsible for providing a reliable stream of data between the two end systems. Thus it is called an "end-to-end" protocol -- it "transports" data from end to end (host to host). The transport layer provides sessions with access to a network in a way that shields them from specific knowledge about it.

Because of the great variation in network topologies and protocols, the transport layer is rather complicated, and must perform several distinct functions:

a. Since the underlying network may lose packets, deliver them out of order, or deliver multiple copies, the transport layer must be responsible for sequencing. It labels outbound messages (Transport Protocol Data Units, TPDU's) with sequence numbers, and assembles inbound TPDU's into the right order, discarding duplicates and requesting retransmission of missing ones.

b. Since the receiving system might not be able to keep up with the rate at which packets are being delivered, the transport layer must include a way for the receiver to tell the sender to slow down. This is called flow control.

c. On multiprocessing (timesharing) systems, there are multiple processes (jobs, programs) active at the same time, and more than one of them may be simultaneously using the same network connection. The transport layer allows each process to use the network connection without getting the data confused, by "multiplexing" the messages.

(picture of multiplexing here...)

OK, so the host computer is receiving a reliably sequenced stream of packets, and delivering data to the intended users. But suppose the packets had to travel though a complicated network to get from one host to the other. How do they find their way through the network?

3. THE NETWORK LAYER

...knows the route to take to get from one host to the other. When a message arrives at a particular node in the network, that node knows which node to send it to next. Each node node has this information, and decides which hop to take next, depending upon its knowledge of the layout of the network and the prevailing conditions (which nodes are down, how congested certain routes are).

So now we can can send messages (packets) hop-by-hop through the network. But how is a message transmitted reliably on each hop?

2. THE DATALINK LAYER

...is responsible for taking each step along the route, i.e. ensuring that messages arrive intact. This means framing each message unambiguously, and including error-checking information so that retransmission of damaged frames can be requested.

OK, so now we are transmitting unambiguously delimited, error-checked frames. But exactly how are they transmitted from one node to the next?

1. THE PHYSICAL LAYER

...knows how to use the physical medium for each hop. The software at this layer knows the details for controlling the hardware interface to the medium: the RS-232 UART, the modem, the synchronous line interface, etc. It is responsible for transmitting and receiving bits on the communication medium in sequence and, in general, converting between this "featureless" bit stream and a sequence of bytes that can be operated on by the computer.

EXAMPLE

Let’s look at a highly simplified example, using purely hypothetical protocols. Don’t worry about the details. This is just to give the flavor of what really goes on...

Suppose we have an application which simply sends a short text message from a user on one system to the screen of another user on another system.

The application is "send a message". The application program is called SEND. The user types "SEND CHRIS@CU20B Hi There!" to send the message "Hi There!" to the user named Chris on the system called CU20B.

First, the connection must be established. The application knows that "@CU20B" means that it must find out the address of CU20B. It calls upon a special subroutine in the system to look up the address of CU20B in its host tables. Having found it, it constructs a message of the form:

```
+------+------+
| user | text |
+------+------+
```

and calls upon the presentation layer to send it to the specified address as a screen message. The host address is passed as a parameter.

THE PRESENTATION LAYER looks at the text and translates it, if necessary, to some "canonical format" suitable for transmission (for instance, from EBCDIC to ASCII), and calls upon the session layer to send it to the specified address as a screen message.

```
+------+-----------------+
| user | translated text |
+------+-----------------+
```

THE SESSION LAYER (in this case) does nothing (there’s not really any dialog to manage...)

THE TRANSPORT LAYER adds sequencing information and a transport address (which identifies the user on the end system) and passes the resulting TPDU to the network layer for delivery (in this case, there’s only one TPDU, but if the message were long, there could be multiple TPDU’s whose sequence must be assured):

```
+----------+--------+--------------------------+
|          |        |                          |
|          |        |    "data"                |
|    *     |   *    +------+-----------------+ |
| sequence | TADDR  | user | translated text | |
|          |        +------+-----------------+ |
|          |        |                          |
+----------+--------+--------------------------+
```

Note that the TPDU consists of a sequence number and some data that is "hidden". Each layer treats the upper layer’s PDU as featureless data, and adds a new "layer of clothing" to it.

THE NETWORK LAYER knows how to get to the given address; it adds the destination address to the PDU and calls the datalink layer to send out the appropriate path, which may include any number of intermediate routing nodes:

```
+---------+--------------------------------------------------+
|         | "data"                                           |
|         +----------+-------------------------------------+ |
|    *    |          |        +------+-----------------+   | |
| address | sequence | TADDR  | user | translated text |   | |
|         |          |        +------+-----------------+   | |
|         +----------+-------------------------------------+ |
|         |                                                  |
+---------+--------------------------------------------------+
```

THE DATALINK LAYER sends the message from one point to the next in the path, in a form suitable for transmission, and for error detection and correction, i.e. as a datalink frame:

```
+-------+---------------------------------------------------+-------+-----+
|       | "data"                                            |       |     |
|   *   +---------------------------------------------------+   *   |  *  |
| begin |          (all the fields above...)                | check | end |
|       +---------------------------------------------------+       |     |
|       |                                                   |       |     |
+-------+---------------------------------------------------+-------+-----+
```

and transmits it via THE PHYSICAL LAYER to the next node in the network.

When the datalink frame reaches the next node, that node's datalink layer checks the block check. If it is bad, retransmission is requested. The process repeats until the message is transmitted successfully (or until the limit on retransmissions has been exceeded, in which case an unrecoverable error is reported). Once transmission is successful, the datalink layer removes its "clothing" from the frame and passes the "hidden data" up to the network layer.

In this manner, the packet travels through the network. At each node, the network layer looks at the address and asks "is this for me?". If not, the network layer figures out (somehow) which node to send it to next on its journey to the destination system, and passes the packet back down to the datalink layer to be transmitted on the appropriate line. The process repeats until the packet reaches the destination system.

At the destination system, the network layer peels off its protocol information and gives the remaining data to the transport layer, which examines the sequence number (its own protocol information) to make sure that no previous packets have been missed (if they have, the transport layer requests the corresponding transport layer to retransmit the missing packets), and then strips the sequence number. If the message was long enough to have been split into multiple packets, the transport layer will assemble them in the right order. The resulting sequenced data is provided them to the session layer.

The session layer reads and strips the socket number from the packet and feeds the data to the appropriate process, in this case a screen message handler, which embodies the presentation and application layers. The screen message handler knows that the remaining part of the message contains a user name and a message. It does any necessary format conversion on the message itself (presentation layer), and finally the user Chris sees the message "Hi There!" on her screen (application). Then the application transmits a confirmation to the originator of the message through the network in exactly the same manner as the original message was transmitted.

```
Application         - - - "Hi There!" - - - >               Application
   |                                                           ^
   v                                                           |
Presentation        - - - Let’s talk ASCII - - - >          Presentation
   |                                                           ^
   v                                                           |
Session             - - - Nothing to do! - - - >            Session
   |                                                           ^
   v                                                           |
Transport           - - - Seq number, TADDR - - - >         Transport
   |                                                           ^
   v                                                           |
Network        Network        Network        Network        Network
   |            ^  |           ^  |           ^  |             ^
   v            |  v           |  v           |  v             |
Datalink       Datalink       Datalink       Datalink       Datalink
   |            ^  |           ^  |           ^  |             ^
   v            |  v           |  v           |  v             |
Physical------>Physical------>Physical------>Physical------>Physical
```

3. THE PHYSICAL LAYER

Physical layer entities are connected by a physical medium. Thus the physical layer is the only one that communicates DIRECTLY with its peer.

The OSI physical layer provides the MECHANICAL, ELECTRICAL, FUNCTIONAL, and PROCEDURAL means to ACTIVATE, MANAGE, and DEACTIVATE physical connections for BIT TRANSMISSION between datalink entities. What does all this mean?

To interconnect a wide variety of computers and devices, physical layer standards are required in four areas (sorry for the list!):

1. MECHANICAL: Dimensions of plugs, configuration and assignment of pins, etc., e.g. RS-449 (9 and 37 pin), ISO IS2110 (25 pin).

2. ELECTRICAL: Voltage levels or frequencies on wires, etc. E.g. RS-232-C, CCITT V.24, V.28.

3. FUNCTIONAL: correspondence between electrical signals and data or control information, e.g. RS-232-C, CCITT X.24.

4. PROCEDURAL: rules that apply to the interaction between the two physical entities, distinction between data and control information, sequence of events, etc., e.g. RS-232-C, Hayes AT command set, CCITT X.21 and X.28.

The procedural aspects include mechanisms to:

1. ACTIVATE a physical connection, i.e. to establish a circuit between adjacent computers or devices

2. MANAGE the connection, e.g. monitor the circuit for errors or unexpected disconnection

3. TRANSMIT BITS on the medium, in sequence, so that they are received in the same order in which they are sent.

4. DEACTIVATE or relase the connection when it's no longer needed.

It is NOT the responsibility of the physical layer to correct errors, recover from deadlocks, or to manipulate the data in any way. In fact, the physical layer should be entirely unaware of the nature of the data it is transferring. The data should be seen only as a featureless stream of bits. But in reality, matters aren't necessarily so clearcut.

CIRCUIT ESTABLISHMENT

There are several kinds of connections over which data communication may take place:

1. Dedicated point-to-point links, such as leased phone lines.

2. Dedicated shared links, such as the Ethernet bus. When links are dedicated, each station can "see" its partner. In these cases, the physical address of the circuit is the same as the address of the controlling device. These circuits are permanently established and always available for use, in which case they need not be established or released at the physical level.

3. Multidrop (multipoint) links. In "multidrop" links, any pair of stations may establish a connection, but may transmit only when "polled" by the "master" of the circuit. In that case, the physical layer must provide the datalink layer with the means to uniquely identify the destination of a message.

```
"Master" -----+-----+-----+-----+-----+-----+-----+-----
              |     |     |     |     |     |     |
    Stations: 1     2     3     4     5     6     7    ...
```

4. Circuit-switched links. A switched circuit is one in which a "call" establishes an electrical connection between two stations, for their exclusive use until the connection is released. There are two kinds:

   a. DIGITAL, i.e. designed for use by computers, and therefore fully automated. A "virtual call" mechanism is provided, in which a dedicated physical link is established by means of commands to switching equipment imbedded in the network. This is the method used in X.21, described below.

   b. DIALUP through the voice telephone system, using modems. In this case, connections are normally made manually, by a person dialing the phone. But some modems have an "autodial" feature which produces the same effects (transmits the same signals) as manual dialing. In this case, the physical layer may support the dialing and answering methods used by the particular modems, for instance the Hayes AT command set and responses.

In most cases, the software must still "open" and "close" the circuits, if only to provide an efficient communication path between itself and the communication device, to associate the communication software with the device driver, etc. The process of opening a circuit often requires setting of communication parameters to allow the interfaces on both ends to communicate.

CIRCUIT DEACTIVATION

Deactivation, clearing, or release, of a circuit generally means that the software "closes" the "file" associated with the communication device, and if necessary, to terminate, or hang up, the call which established the connection. Any resources that were devoted to the connection are released for use by subsequent connections.

CIRCUIT MAINTENANCE

During the data transfer phase, the physical layer is responsible for monitoring the device for several special conditions:

- Data ready
- Data transmission error
- Data overrun
- Device available/unavailable for transmission
- Disconnection

In most cases, it is up to the higher layers to handle these conditions. The physical layer merely reports them. In other cases, the physical layer may do some of the work itself.

CIRCUIT PARAMETERS

Several characteristics of the circuit are of interest to the physical layer. These must be known or established at the time the circuit is opened. In many kinds of circuits, two devices cannot communicate at all unless certain of these basic parameters agree at both ends. Three important parameters are speed, parity, and line access discipline. They apply mostly to point-to-point connections.

SPEED

The physical signalling rate must agree at both ends, so that the receiver can identify the transmitted bits correctly. There are two major techniques for achieving this agreement. In one, the two ends support a variety of built-in rates, and these are manually set to be the same at each end. In the other, the transmission itself includes synchronization information. We will look at some examples shortly.

PARITY

In US ASCII, only 7 out of 8 bits are used for data. The 8th bit is sometimes used for error detection. The transmitter of a byte will set the 8th bit to 0 or 1, depending on how many 1-bits are in the 7 data bits. When the 8th bit is used this way, it's called the parity bit. The receiver checks the parity bit against the data, and if they don't agree, it knows the character was received in error.

There are 5 possible types of parity:

Even	Parity bit set to make total number of 1-bits even. Odd
	Parity bit set to make total number of 1-bits odd. Mark
	Parity bit always set to 1. Space
	Parity bit always set to 0. None
	No parity calculation is done, 8th bit available for data.

ANSI X3.16-1976 Defines even and odd parity, and specifies that ODD parity should be used with 7-bit ASCII in synchronous communication, and EVEN parity should be used with 7-bit ASCII in asynchronous communication, and that NO parity may be used with 8-bit data.

When parity is in use:

- Errors can be detected, but not corrected, by the receiver.
- 8-bit binary data cannot be transmitted in byte form.

It sometimes happens that one device uses parity and another does not. In other cases, some device situated between the two communicating devices uses parity. To communicate in these situations, the connection must use the required kind of parity.

ACCESS TO MEDIUM: "PLEX"

We have seen how data is transmitted by the physical layer. But WHEN is the data to be transmitted? It turns out that because of the characteristics of the devices that are communicating, or of the medium that connects them, it is not always possible for a device to transmit whenever it wants to.

There are two ways of deciding when to transmit:

- The datalink layer gives the physical layer bits to be transmitted, and the physical layer transmits them immediately. Thus the datalink layer decides when to transmit.
- The physical layer has some knowledge of the medium which is shielded from the datalink layer, and therefore decides on its own, and so needs a buffer of its own.

Or some combination of the two -- the borderline is not always clear. The rules for when to transmit are called a "LINE DISCIPLINE". There are several possibilities.

If communication is always one way (like a radio broadcast), we call the connection "SIMPLEX". This is rarely done in data communication.

```
 (...picture...)  (one wire)    A -------> B  (ONE-WAY)
 (A can transmit any time, B can never transmit)
```

If data can go both ways, that is if TWO devices can transmit on the same physical connection, we call the connection "DUPLEX". If transmissions can go both ways simultaneously, we say the connection is "FULL DUPLEX" (TWO-WAY SIMULTANEOUS). But if they can go only one way at a time, it is "HALF DUPLEX" (TWO-WAY ALTERNATE).

```
 (...picture...)  (one wire possible for half duplex, 2 needed for full)
```

It is also possible to connect many computers together over one connection, and the word for this is "MULTIPLEX".

Half Duplex

In HALF DUPLEX connections, each end must tell the other when it may transmit.

In terminal-to-host connections (e.g. IBM mainframe), the terminal grants access to the computer by sending an ASCII CR, and the computer grants permission to the terminal by sending an ASCII XON (Ctrl-Q) or other special control character. If one side sends to the other before permission is granted, characters will be lost, because the other side isn't listening.

The act of terminating transmission with a special character which grants the other side permission to transmit is called a "line turnaround handshake".

The physical layer is supposed to be concerned only with bits, not characters. It's not supposed to look at the data at all. So this kind of line discipline would seem to belong in the datalink layer (right?).

But on the mainframe side, the software never gets the characters at all until the communication front end sees the CR. So it really happens in the device. Conversely, after the IBM mainframe has transmitted its lines or screens, it is its next READ request that triggers emission of the XON, transparent to the software. Since the software never sees the line turnaround characters, we might as well consider this line discipline a physical protocol.

Full Duplex

In FULL DUPLEX connections, both sides can transmit simultaneously. This means:

- Neither side needs permission to send.
- Remote computer can echo your typing.
- Buffer overruns can be prevented.

Preventing buffer overruns is called "FLOW CONTROL". When the connection is full duplex, then the receiver can send a special character to the sender whenever its buffer is getting too full, and the sender will stop until the receiver sends another special character to tell it to resume sending. These special characters are called:

XOFF	ASCII Control-S (stop sending)
XON	ASCII Control-Q (resume sending)

and this type of flow control is called XON/XOFF.

Some full duplex systems support XON/XOFF and others don't. It only works if both sides do it.

Is XON/XOFF a physical or datalink level mechanism? Since the data is being examined for specific characters, it would seem to belong in the datalink layer. But often XON/XOFF is implemented in the device driver, transparent to the datalink-level software. So maybe it's at the physical level after all?

Multiplex

This is another kind of "plex" in which many computers can share the same transmission medium. How can they do this? Two ways: frequency division multiplexing (FDM), where each circuit is assigned its own frequency range, much like television or radio channels. A prominent example is the "broadband" network built on cable-TV technology.

The other way is called "time division multiplexing" (TDM), in which each station takes turns using the medium. There are many schemes, including:

- Multidrop, in which a "master" station controls all the other stations, granting permission to transmit to one at a time.
- CSMA/CD as in Ethernet. The physical layer senses carrier and detects collisions, reporting these conditions to the datalink layer, which decides when to transmit.
- Token Ring, in which the owner of the token gets to transmit for a limited amount of time, and then must give up the token.

DATA TRANSMISSION

The physical layer is responsible for delivering BITS to their destination in the same order in which they were transmitted. Bits are the "SERVICE DATA UNITS" of the physical layer. These bits must be converted to some kind of electrical or other signal for transmission and, obviously, the transmitter must use the SAME ENCODING as the receiver.

How do we transmit a bit on the communication medium? For example, one voltage is used for binary 0, another for 1. These voltages are called "space" and "mark", respectively. Space (0) may be +12V, and mark (1) is -12V. There must also be a convention to show that the connection is active, but no data is being transmitted, i.e. the line is "idle". An absence of voltage means there's no transmission at all, the connection is inactive.

Another method is to modulate a carrier wave in some way, for instance by modifying its frequence (FM), amplitude (AM), or phase (PSK).

The capacity of a transmission medium is called its "BANDWIDTH", usually measured in bits per second, sometimes correctly called "BAUD", sometimes incorrectly. Baud is equivalent to bits-per-second if a single signalling element can have only two states, equivalent to 0 and 1. Some signalling schemes can represent more than two states in one signal. For instance, if there were four voltages, rather than 2, then one signal could represent any of four values (00, 01, 10, 11), and 1 baud would be 2 bits/sec. In general,

1 baud = log2(n) bits/sec

where n = number of values a signalling element may assume.

But a "BYTE" is the smallest amount of data that can be moved into and out of the computer's memory; on most computers, a byte has 8 bits. A byte is used to represent a "character" in a particular character set, like ASCII or EBCDIC, or (part of) some other quantity: a number, an instruction, an address, etc.

Since computers can transfer data only one or more bytes at a time, data communications also tends to be byte-oriented, transmitting a byte at a time, rather than bit-oriented.

How to transmit a byte on wires? Two ways, parallel and serial. The ISO physical layer allows either way.

PARALLEL: All the bits of a byte at once, each on its own wire.

```
Device A       Wires       Device B
  Bit 0 ------------------> Bit 0
  Bit 1 ------------------> Bit 1
  Bit 2 ------------------> Bit 2
  Bit 3 ------------------> Bit 3
  Bit 4 ------------------> Bit 4
  Bit 5 ------------------> Bit 5
  Bit 6 ------------------> Bit 6
  Bit 7 ------------------> Bit 7
Control ------------------> Control
```

The control wire has the mark voltage whenever the other 8 wires contain data, so that the receiver knows when to "read" them.

Advantages:

- High bandwidth

- Simple logic
- Obvious delimitation of bytes

Disadvantages:

- High cost over long distances (many wires),
- Propagation delay varies between wires, so bits might not arrive together.
- If communication to be 2-way simultaneous, need 2x as many wires, otherwise data can only go one way at a time.

Parallel transmission usually used only for very short distances, such as between devices within a computer, or between computer and printer, but it is rarely used in data communication.

SERIAL TRANSMISSION

In serial transmission, the bits of a byte are sent one after another, on the same wire.

So the physical layer must provide the mapping between bytes and bits. The first question that springs to mind is, what order to send them in?

In character or byte-oriented transmission, it is customary to transmit the least significant (low-order) bit first, and the most significant (high order) bit last. This is specified in ANSI X3.15-1976 "Bit Sequencing of ASCII in Serial-by-bit Data Interchange."

The next question is, how does the receiver distinguish the bits from each other, and how does it know the boundaries between the bytes?

Two ways: Framing (asynchronous) and Timing (synchronous)

ASYNCHRONOUS TRANSMISSION

Serial, asynchronous transmission is the most widely used form of data communication. It's designed for use between two devices that are not synchronized, for instance a computer and a terminal controlled by a human. The computer cannot know when the human will type a character. That's why it's called asynchronous.

So when a character arrives at the computer, how does the computer know?

Bytes are delimited by framing them between a "start bit" (space) and a "stop bit" (mark). When no transmission is occurring, the line contains a steady mark voltage. When this voltage changes to space, the receiver knows that the voltage should be sampled for the next 8 bit times to form a character. If the next bit is not a mark, then a "framing error" is reported.

```
           ----- 8-bit byte -----
+---------------------------------------------------+
|  1 |  x |  x |  x |  x |  x |  x |  x |  x |  0 |
+---------------------------------------------------+
   ^    ^                                   ^   ^
   |    | High Order data bit               |   | Start bit
   | Stop Bit                               | Low order data bit
```

This format is specified in ANSI X3.16-1976, "Character Structure and Character Parity

Sense for Serial By-Bit Data Communication in ASCII". As you can see, 10 bits are required to transmit an 8-bit byte asynchronously. Thus 10 baud is equivalent to 1 character per second.

There are also encodings that are not based on ASCII, for instance a 5-bit start/stop code called BAUDOT, once widely used with Teletypes, and still used by Telecommunication Devices for the Deaf (TDD).

Here’s what asynchronous transmission of ASCII "C", 67 decimal, looks like:

```
             +-------------------------------------------------+
             |  1 |  0 |  1 |  0 |  0 |  0 |  0 |  1 |  1 |  0 |  ---->
             +-------------------------------------------------+
                     ^                                  ^
                     | High Order data bit              | Low order data bit
                  |<------------- data bits ------------->|
                     0    1    0    0    0    0    1   1
+12V              +----+    +----+----+----+----+         +----+
                  |    |    |                   |         |    |
  0V - - - - - - -|- - - - -|- - - - - - - - - -|- - - - -|- - - - - - - -
                  |    |    |                   |         |    |
-12V --------+----+    +----+                   +----+----+    +--------->
      idle    stop                                         start   idle
              bit                                          bit

    ^^^^^^^^^^^^    ^    ^    ^    ^    ^    ^    ^    ^     ^^^^^^^^^^^^^
    16X sampling                1X sampling                  16X sampling
```

What is a bit time? How does the receiver know at what intervals to sample the line? Answers:

1. Both ends must be set to exactly the same speed in bits per second (baud).

2. Bit time = 1/bps (example: 300 baud bit time is 1/300 = .00333 sec).

3. During idle period, receiver samples much faster than baud rate to detect leading edge of start bit. Then, halfway through 1 bit time, it samples once per bit time, to hit the "middle" of each bit.

In async transmission, baud rates MUST match. Otherwise, the receiver will sample each bit multiple times, or skip bits altogether, and therefore read the data incorrectly. There will also be a very high number of framing errors, which allows software to detect that something is amiss.

```
 (show erroneous sampling on picture...)
```

In most cases, the encoding for, and transmission of, bits on the physical medium is handled by a hardware device like a UART, USRT (Universal (A)Synchronous Receiver/Transmitter), modem, or Ethernet controller, and therefore the software need not be concerned with the details. Typically, the physical layer software simply reads and writes bytes from and to the device controller, usually via the services of the device driver. Data is transferred, and any physical-level errors that occur (like overruns, framing errors, etc) are reported through the device flag registers.

SYNCHRONOUS TRANSMISSION

Asynchronous communication uses a single method to identify data bits and to frame data bytes, namely the start and stop bits. But these 2 bits add 20% transmission overhead to each byte.

In SYNCHRONOUS communication, two separate methods are used for bit identification and byte delimitation. Bits are identified like in parallel transmission, via a separate "control" signal. Except in this case, the control signal is driven by a clock, and pulses regularly. The receiver reads a data bit each time the clock pulses.

Start and stop bits are unnecessary, so there need be no interruption between bytes. So how does the receiver know where one byte ends and the next begins? A special "synch" character (usually ASCII SYN with 8th bit on, 10010110) is used.

Initially, the receiver goes into "SYNC SEARCH mode" looking for this pattern. In this mode, it ignores all data that is NOT a SYN. When it finds a SYN, it turns on its "synchronization achieved" flag, and presumably the software sees this and takes it out of Sync Search mode, which now means it treats each group of 8 bits as a distinct data byte.

To acheive synchronization, usually 3-5 SYNs are sent in a row -- subsequent SYNs are discarded by receiver hardware. This is because (a) it is fairly likely that the first SYN will be mangled, (b) it is not unlikely that a SYN pattern can be found across two data bytes.

```
 000011101010110010110100101101001011000010010110011010110011000011010101000
              ^       ^       ^       ^       ^       ^       ^       ^
              SYN     SYN     SYN     Data starts here
```

Advantages of synchronous communication include:

- High bandwidth can be achieved because bits arrive at a steady predictable rate, and there's no per-character overhead.
- Modems using modulation techniques dependent on a constant data flow can be utilized. These modems are more expensive, but can achieve much higher line speeds (maybe 4 times higher).
- Transmission need not be byte-oriented.

But synchronous transmission has several intrinsic complications:

- What if there is noise, or momentary failure, of the transmission medium? The two sides might become unsynchronized. But there's no way for the physical layer to know this. It simply feeds (what it believes to be) bytes to the datalink layer. The datalink layer must figure out for itself when the bytes stop making sense.
- What if SYN itself occurs in the data? The synchronous receiver must be "told" which mode to be in: "sync search" (unsynchronized) or "data" (synchronized). It is up to datalink software to determine when there have been transmission errors or synchronization has been lost.
- What if the receiver loses synchronization, and goes into sync search mode? How does the sender know to start sending SYNs?
- What does the transmitter send when the software has not provided it with data? The clock pulses constantly, once per bit time. This implies that the

software must constantly feed bytes to the transmitter. Usually the SYN pattern is sent continuously when there is no data to send; this helps to keep the two ends synchronized. This poses some problems for datalink software. When SYNs might appear inside the message -- are they data, or idles? The receiver can't tell the difference, but the sender can find out when this happens by catching "underrun" interrupts from the interface ("you didn't feed me data fast enough!"). To complicate matters, the synchronous interface, when in data mode, can also be put into "strip sync" mode, which tells it to discard all arriving SYNs.

EXTENDING PHYSICAL CONNECTIONS

Physical connections may be extended in many ways, depending on the medium: modems, line drivers, repeaters, microwaves, satellites, networks, etc. The key point is that the electrical characteristics at each end is the same. If some conversions are done in order to the extend the distance, they are undone before the signals reach their final destination.

We will see in the coming weeks that there are ways to extend networks at the datalink and network levels as well, each with its own advantages and limitations.

SOFTWARE

On most computer systems, the software to operate at the physical level is quite simple, consisting mostly of OPEN, READ, WRITE, and CLOSE statements. The OPEN statement includes the name of the communication device, e.g.

```
100 OPEN "COM1:1200,S,7" AS #1      (IBM PC BASIC)
fd = open("/dev/tty06","rw");       (UNIX C)
```

The file system looks up this filename and calls in the appropriate device driver. From this point, READ, WRITE, and CLOSE calls simply invoke the device driver, e.g.

```
READ #1, X$ : CLOSE #1              (IBM PC BASIC)
r = read(fd,&x,1);  close(fd);      (UNIX C)
```

However, most high-level programming languages do not include access to the "finer points", for instance sampling the device registers or modem. This usually requires assembly language.

Furthermore, the communication device drivers included in most operating systems are poor performers. For example, the IBM PC's COM driver does not keep up with input faster than about 1200 bps, and it does not provide an option for XON/XOFF flow control. Yet the IBM PC hardware is perfectly capable of speeds of 56Kbps or greater. For this reason, many PC communication software vendors replace the PC's COM driver with their own, "STEALING" the communication device INTERRUPT from the operating system by substituting its own device handler address into the system's INTERRUPT VECTOR, and doing its own buffering and flow control. These must be restored to their previous state upon connection release, or else the device would be unusable by subsequent processes.

EXAMPLES OF PHYSICAL PROTOCOLS

We've discussed the functions of the physical layer: circuit establishment, maintenance, and release, and data transmission on the electrical level. Now we'll look at several physical-level protocols to see how they perform these functions.

(*** Pass around an RS-232 Connector ***)

EIA STANDARD RS-232-C

EIA Standard RS-232-C (and CCITT V.24 in Europe) specify the circuits and voltages required for SYNCHRONOUS and ASYNCHRONOUS SERIAL communication. There are 25 circuits, and therefore a 25-pin connector is required. RS-232 does not describe the mechanical configuration of the connector. For many years, this was a 'de facto' standard, which became formalized by the ISO in 1980 (about 15 years after the fact) as ISO International Standard 2110. 37-pin and 9-pin connectors are specified in EIA RS-449 and ISO 4902. We'll look at the 25-pin configuration.

Circuits are provided for data transfer, connection control, and gounding. Each circuit is assigned to a particular pin. A minimal duplex connection uses:
  Pin 2: Transmitted data
  Pin 3: Received data
  Pin 7: Signal ground (reference ground for the voltages on other pins).
These three wires are sufficient for transmission of data in both directions. What are all the other 21 RS-232 circuits for? They fall into several categories:
  Circuit establishment and maintenance
  Timing for synchronous operation
  Line discipline
  Protective ground
  Undefined
We'll look mainly at asynchronous communication...

Six of these signals are to allow Data Terminal Equipment ("DTE") such as a computer or terminal, to coordinate its operation with Data Communication Equipment ("DCE"), so that each device knows that the other is turned on, functioning, etc. RS-232-C assumes that DTEs are always connected to DCEs, and not to other DTEs.

A modem (MOdulator/DEModulator) is a DCE that converts the discrete "digital" RS-232-C signals output from a UART and converts them to "analog" signals for transmission over telephone connections as if they were audible voice. One modem is in "originate" mode and transmits on a predefined frequency, and the other is in "answer" mode and transmits on a different frequency. Each modem knows to listen for the other's "carrier" signal at the agreed-upon frequencies.

Here are the modem signals:

Pin 20	DTR (Data Terminal Ready) The DTE (terminal, PC) tells the DCE (modem) that it's turned on.
Pin 6	DSR (Data Set Ready) The DCE (modem) tells the DTE (PC, terminal) that it's turned on.
Pin 8	DCD (Data Carrier Detect) (Also called CD, and RLSD - Received Line Signal Indicator) The DCE tells the DTE that it is receiving the other DCE's carrier.
Pin 4	RTS (Request to Send)

	The DCE asks the DTE's permission to send it some data.
Pin 5	CTS (Clear to Send) The DTE grants permission to the DCE to send data.
Pin 22	RI (Ring Indicator) The DCE tells the DTE that someone is calling it up.

Many of the RS-232-C circuits are largely unused. For instance, 5 of them are "secondary" versions of primary TD, RD, RTS, CTS, and CD, to allow two communication channels to share a single interface connector. Has anyone ever heard of a connection that actually used these signals? (** yes **)

A connection between 2 computers using asynchronous modems looks like this:

```
----------------------------------------------------------------------
             (originate)                            (answer)
DTE1   wires   DCE1        Phone Lines                DCE2  wires  DTE2

SG   1 ------- 1                                         1 ------- 1

TD   2 ------> 2                                         2 <------ 2

RD   3 <------ 3                                         3 ------> 3
                   <------One-Way Carrier Frequency
RTS  4 ------> 4                                         4 <------ 4

CTS  5 <------ 5                                         5 ------> 5
                   Other-Way Carrier Frequency---->
DSR  6 <------ 6                                         6 ------> 6

SG   7 ------- 7                                         7 ------- 7

DCD  8 <------ 8                                         8 ------> 8

DTR 20 ------> 20                                       20 <------ 20

RI                                                      22 ------> 22
----------------------------------------------------------------------
```

Here is a typical RS-232 physical protocol:

1. User 1 calls the phone number of DCE2.
2. DCE2 "hears" the ringing and asserts (turns on) RI to DTE2.
3. DTE2 tells DCE2 that it should answer the call by asserting DTR.
4. DCE2 sends its carrier signal to DCE1.
5. DCE1 detects carrier and asserts DCD to DTE1.
6. DCE1 sends carrier to DCE2.
7. DCE2 detects carrier and asserts DCD to DTE2.
8. Now DTE1 and DTE2 both know they're connected together.
9. DTE1 and DTE2 can exchange data over their TD and RD circuits. This data will be modulated and impressed upon the appropriate carrier wave.
10. When user1 is done, she causes DTE1 to turn off DTR.

11. DCE1 stops sending carrier and hangs up the phone.

12. DCE2 turns off CD

13. DTE2 turns off DTR.

14. DCE2 stops sending carrier, and hangs up the phone.

Some modems are HALF DUPLEX. This means that instead of dividing the analog frequency spectrum into two carrier bands and using both at the same time, each one uses the full bandwidth. But to do this, they have to take turns, which means they can't let both DTEs transmit at the same time. The modems and computers coordinate line access using the RTS and CTS signals, which are handled by software at the physical layer.

If any component of a modem connection stops working, then all the other components will find out automatically, and the connection can be released:

1. If DTE1 crashes, its DTR goes off, so DTE1 stops sending carrier, DTE2 notices this and turns off DCD, so DTE2 knows the connection is broken, and it turns of DTR so the modem can accept the next call.

2. If DCE1 stops working, it will stop sending carrier, so that DCE2 and DTE2 will find out, and its DSR and DCD signals will go off so DTE1 will know too.

Similar reasoning applies when DCE2 or DTE2 stop working. The only thing that does not happen automatically is establishment of the connection: someone has to dial the phone.

SYNCHRONOUS CONNECTIONS

On the physical level, synchronous connections use the same RS-232 circuits as asynchronous, plus a few in addition. The most important are:

Transmission Signal Element Timing (DTE to DCE), pin 24 (DA) Transmission Signal Element Timing (DCE to DTE), pin 15 (DB) Receiver Signal Element Timing (DCE to DTE), pin 17 (DD)

These signals allow the transmitter to include clock pulses on a separate circuit from the data, so the receiver will know when to sample each bit. The two different transmit leads allow either the DTE or the DCE to provide the clock pulse.

SMART MODEMS

The RS-232-C standard includes all the facilities needed for automatic answering of dialup data calls, but there is no provision for automatic calling. In fact, there is no widely accepted "official" standard for automatic dialing of telephones. (There is a standard, RS-366-A, 1979, which is not widely applied.) Thus RS-232, when used in conjunction with dialed phone connections, requires a human operator to dial the number and detect whether the response is:

1. no answer

2. busy

3. a voice

4. data carrier

and, in the last case only, to activate the RS-232 physical protocol.

Many modern modems include a command language to allow the DTE to issue dialing, hangup, and mode-setting commands. After connection is established, command mode is usually disabled, except perhaps for a special escape sequence. Most popular command language is Hayes AT command set (ATD, ATH, etc), and has assumed the stature of ’de facto’ standard, imitated by many other manufacturers, and supported by most asynchronous communication software. The key commands are:

AT	Attention, precedes most commands ATDnumber
	Dial the number (transmit at ORIGINATE frequency) ATS0=1
	Prepare to answer a data call (transmit at ANSWER frequency) +++
	Escape back to Hayes command interpreter ATH
	Hang up the phone

Character responses indicate whether the call was completed (CONNECT), BUSY, no answer, etc. If the call is completed, RS-232 signals are asserted automatically, in accordance with the RS-232 protocol. Thus software which can send and receive characters, and which can monitor and control the key modem signals, can activate, utilize, and deactivate a physical connection, without human intervention.

Is this a physical protocol? Even though it’s dealing in characters, the characters are not being transmitted between the two DTEs, but rather between the local DTE and the local DCE. They are used to establish the connection between the two DTEs. Therefore, it’s a physical level protocol.

SOFTWARE

A great body of software operates at the RS-232 physical level. This includes most popular ASYNCHRONOUS COMMUNICATION PROGRAMS. They typically include commands to dial Hayes or other modems, to monitor and/or control modem signals like DTR, CD, etc, to send characters in either direction, and to terminate the connection when done.

Additional protocols are often built on top of this physical base. For instance, a TERMINAL EMULATOR follows a kind of "protocol" when interpreting a predefined set of escape sequences sent by the host computer.

Software that operates purely at the physical level will find itself doing not only terminal emulation, but also "human emulation". For example, many software packages include a "SCRIPT LANGUAGE", which allows routine sorts of interactions between a person and a computer to be programmed -- the computer sends what the person would have typed, and looks for the responses that the person would have looked for. When such a script includes control of AUTODIALERS, one computer can call another up in the middle of the night and have extended chit-chat, without any human assistance, thus rendering humans obsolete.

Still, the occasional functioning human may want to move data from one computer’s disk to another’s. This may be done at the physical level using a procedure called "RAW DOWNLOAD". If you have a PC doing terminal emulation, then it is copying characters that arrive at the PC’s communication port onto the screen. If it can do that, then it can also copy them to the PC’s disk. If you cause the remote computer to "type" a file on your screen, but slyly capture the characters into a disk file, you have performed the act of "raw downloading".

Conversely, if the PC can copy the characters you type to its communication port, then it can also copy characters from a PC disk file to the port. If you set up the remote computer to collect these characters into a file, you have done "RAW UPLOADING".

Raw down- and upload can be quite effective on CLEAN, FULL DUPLEX connections capable of XON/XOFF. Otherwise, the technique is fraught with peril.

There is a somewhat more reliable, but less general technique for uploading, called ECHOPLEX. It may be used only on full duplex connections, where the remote computer echoes your typein. Your program sends the file to be uploaded a character at a time, and reads back the echo of each character. If it agrees, the program goes on to the next character. Otherwise, it sends a BACKSPACE or RUBOUT, and then resends the character. Obviously, there are various complications, but these are relatively minor.

CCITT X.3 and X.28 PHYSICAL LAYER PROTOCOL

If the Hayes AT command set is a physical-level protocol, then another similar and important protocol migh be considered in the same class: CCITT Recommendation X.3, which describes the functions of a "packet assembler/disassembler" (PAD), and X.28, which describes the methods available to a "start/stop" DTE (typically an asynchronous character terminal or PC) to place a data call through a public packet-switched network, and to establish and control the communication parameters to be used in the connection.

(Actually, this protocol also has elements of the presentation layer in it, and some observers place it there...)

In this case, the user dials up a PAD on a public network like Telenet or Tymnet, using the RS-232 protocols, plus Hayes or other dialing protocols. But this is only the first step in establishing the physical link. At this point, you must issue PAD commands to set up communication parameters appropriate for the computer you wish to connect to, and then direct the PAD to put you through.

```
"SET? p:v,p:v,..."  sets the given parameters to the given values
"PAR? p,p,p,..."    displays the values of the specified parameters
"C xxx"             calls network address xxx (like a phone number).
```

The settable parameters include baud rate, local vs remote echo, XON/XOFF flow control, and various terminal-related parameters. Example:

```
SET 2:1,3:2,5:0
C X123
```

means select local echo, send terminal input a line at a time, and don't use XON/XOFF, in other words a half-duplex terminal setup, and then connect to system "X123". The parameters are numeric so as not to favor any particular language.

The connection is released automatically when you logout from the remote system, or manually when you "escape back" to the PAD (using a special escape sequence like <CR>@<CR>) and give the CLR command.

Many of the popular PC communication programs include functions to dial up popular information services like The Source through PADs. You merely tell the PC to connect you to The Source, and it handles the details of dialing the PAD, setting parameters, and connecting to The Source.

IEEE 802.3 (ISO 8802/3) ETHERNET PHYSICAL LAYER PROTOCOL

The Ethernet physical layer is EXTREMELY simple. There's no opening or closing of the connection, no call setup, etc. The medium is always there, and all transmissions go to all connected stations, so all the physical layer has to worry about is transmitting & receiving the bits, and detecting and reporting line status.

An Ethernet connection consists of an Ethernet controller, which is a board in the computer, a transceiver cable, and the transceiver itself (Medium Access Unit, MAU), which "bites" into the Ethernet bus. The transceiver cable has four wires: Data In & Out, Control In & Out.

(draw picture...)

Manchester encoding is used for data, in which data and clock are combined in "bit symbols". Each symbol is split in half, with each half containing the binary inverse of the other half, so a transition always occurs in the middle of a bit symbol. An upwards transition is a 1, a downwards transition is a 0. When there is no data, the line is kept at 1, for at least 2 bit times. The next transition signals the start of more data.

```
 IDLE...................    0       0       1       0   IDLE.......
 -------+-------+-------+---    +---    +    ---+---    +-------+-------+---
                           |   |   |       |       |   |
        +       +       +    ---+    ---+---    +    ---+       +       +
```

The control wires can each carry 3 different signals:

	IDLE	CS1	CS0
Control Out:	Normal	MAU-Request	Monitor (Receive Only).
Control In:	MAU-Avail	MAU-Not-Avail (Carrier Sense)	Error (Collision Detect)

These serve similar functions as some of the RS-232 modem signals, e.g. MAU-Request is like RTS. MAU-Available is like DSR/CTS. MAU-Not-Available is like CTS OFF.

Software interface to Datalink Layer: 3 procedures, 3 global variables:

Procedures:

RECEIVE-BIT	Reads a bit, returns its value to DL layer.
TRANSMIT-BIT(b)	Sends given bit.
WAIT(n)	Waits for n bit times.

Variables:

COLLISION-DETECT	Set by PMA when it detects a collision, i.e. more than one station transmitting at once.
CARRIER-SENSE	Set by PMA when it is receiving data, like UART data-ready flag. Datalink layer won't send if this is on.
TRANSMITTING	Set by Datalink Layer when it wants to transmit.

Datalink layer sets TRANSMITTING flag, then calls TRANSMIT-BIT() with each bit to be transmitted, in sequence. Then turns off flag when done transmitting.

The Ethernet physical layer is implemented entirely in "firmware", inside the Ethernet controller. And so is a good portion of the datalink layer. For this reason, there is little or no software to access the Ethernet transmission medium on the physical level.

CCITT X.21 PHYSICAL LAYER PROTOCOL

CCITT Recommendation X.21 specifies the "interface between DTE and DCE for synchronous operation on public data networks":

```
+-----+    +-----+               +-----+    +-----+
| DTE |----| DCE | - (network) - | DCE |----| DTE |
+-----+    +-----+               +-----+    +-----+
|                |               |                |
|<---- X.21 ---->|               |<---- X.21 ---->|
```

There are actually two parts to this protocol: the physical interface and circuit assignments (X.24, very similar in spirit to RS-232-C, which gives connector and pin assignments for DTE-DCE connection), and the functional specification (X.21 itself) -- how to use these circuits to establish, release, and utilize point-to-point connections.

X.24 CIRCUITS:

First, let's look at the circuits specified in X.24:

```
  DTE                  DCE      Meaning                   RS-232-C Analogy
+-----+              +-----+
|     |   Transmit   |     |
|     +------------->+     | Send data to partner          Transmit Data
|     |              |     |
|     |   Control    |     |
|     +------------->+     | On-Line, "off-hook"           DTR
|     |              |     |
|     |   Receive    |     |
|     +<-------------+     | Receive data from partner     Receive Data
|     |              |     |
|     |  Indication  |     |
|     +<-------------+     | Connected with partner        Carrier Detect
|     |              |     |
|     |  Bit timing  |     |
|     +<-------------+     | To synchronize reception of bits
|     |              |     |
|     |  Byte timing |     |
|     +<-------------+     | To delimit bytes within bit stream (optional)
|     |              |     | (what a good idea!)
+-----+              +-----+
```

When the Control and Indication circuits are both ON, then data is being exchanged transparently between the datalink layers of the two partners; this is called data phase. Otherwise, control information is being exchanged between the DTE and DCE according to the X.21 protocol in order to establish or break a physical connection; this is called control phase.

How does the DCE know the DTE is turned on if its Control circuit is off, and how does the DTE know the DCE is turned on if its Indication circuit is off? Answer: they transmit a steady stream of 1-bits to each other on their data circuits.

X.21 PROTOCOL

To initiate a call, the DTE turns on Control and stops transmitting 1's. The DCE sends a SYN and a series of '+' characters. DTE then sends a SYN followed by a dialing command, and waits for reply. (The SYN is provided in case the optional byte-timing signal is not supplied.)

The network sends SYN and BEL (ring) to the called DTE, which accepts the call by turning on Control. The network gives line identification info to both DTEs and then turns on both Indication circuits.

Now the two DTEs are in data phase, and have a full duplex data connection. Notice the similarity to an autodial modem with the RS-232 signals.

To clear a connection, a DTE merely turns off its Control circuit, and the DCEs and other DTE go back to control phase. This similar to RS-232-C, in which the terminal can drop DTR to terminate the connection.

X.21 SOFTWARE IMPLEMENTATION

Given an X.24 physical interface to a network and a device driver that takes care of the bit and byte delimitation and the transmission of 1's while the line is inactive, then the program to execute X.21 protocol is fairly simple. Let's assume we have functions available to test selected control signals and turn them on and off, and that connections are point-to-point (rather than multidrop). We would need only two software functions: one to open a connection, and another to close it. Here's some pseudo-C-language code for calling function:

```
(1)  call(number,line) {
(2)     p = open(line);
(3)     if (p < 0) then return(p);
(4)     set(p,Control,ON);
(5)     x = read(p);
(6)     if (x != "+++") then return(-1);
(7)     x = write(p,number);
(8)     address = read(p);
(9)     for (i = 10; i > 0 && test(p,Indication) == ON; i--) sleep(1);
(10)    if (i == 0) return(-1) else return(p);
(11) }
```

The program is shown in C rather than BASIC for compactness and clarity, and some liberty has been taken with the data types, omitted declarations, etc.

Line (1) is the function definition. The function's name is "call" and it takes two parameters, the number to call (like a phone number) and the physical communication line on which to place the call (like selecting COM1 or COM2 on a PC). The function returns a negative number if it fails, and returns a file descriptor for the open line if it succeeds. The function's definition terminates with the closing bracket on line (11).

```
(2)     p = open(line);
(3)     if (p < 0) then return(p);
```

Statement (2) opens the communication line. This presumably makes the device driver stop transmitting 1's. If the line could not be opened, the function reports failure by returning the same negative number that the "open" function returned. If it succeeds, the device descriptor is returned in the variable p, for use by subsequent read and write statements.

```
(4)     set(p,Control,ON);
(5)     x = read(p);
(6)     if (x != "+++") then return(-1);
```

Statement (4) turns on the Control circuit, which lets the DCE know to expect commands to appear on the DTE's Transmit circuit. Statement (5) attempts to read a response from the network. Statement (6) checks to see if the response is a series of plus signs; if not, the "call" function returns failure (this example simplistically assumes exactly 3 plus signs are supplied).

```
(7)     x = write(p,number);
```

Once the network has indicated its willingness to communicate, our program sends a dialing command in statement (7). The format of the number as defined in X.21 is quite complicated, and has many fields and subfields for requesting particular addresses, facilities, etc.

```
(8)     address = read(p);
```

After dialing, the program waits in statement (8) for a response from the network, in the form of a called line address identifier. This is kept for future reference (usually needed only for multidrop connections).

```
(9)     for (i = 10; i > 0 && test(p,Indication) == ON; i--) sleep(1);
(10)    if (i == 0) return(-1) else return(p);
```

In statement (9) the program waits for the DCE's Indication circuit to come on. It looks once a second for 10 seconds. If the circuit does not appear, then failure is reported. Otherwise, the function returns successfully, providing the higher-level software with the file descriptor of the open connection to use for data exchange.

The program to clear the connection is even simpler:

```
(1)  clear(p) {
(2)      set(p,Control,OFF);
(3)      for (i = 10; i > 0 && test(p,Indication) == ON; i--) sleep(1);
(4)      close(p);
(5)      if (i == 0) return(-1);
(6)      return(1);
(7)  }
```

It just turns off Control, waits for Indication to go off, closes the communication line, and then returns reporting success or failure depending on whether Indication actually went off. The close function presumably starts the device transmitting 1-bits continuously again.

Thus, the interface presented to higher level software is quite simple. Here's a program fragment that opens a connection, sends a message, reads the reply, and closes the the connection:

```
(1)  n = call("*123456789+",line);
(2)  if (n < 0) then error("Can't place call");
(3)  x = write(n,message);
(4)  if (x < 0) then error("Can't send message");
(5)  x = read(n,reply);
(6)  if (x < 0) then error("Can't read reply");
(7)  print(reply);
(8)  x = clear(n);
(9)  if (x < 0) then error("Can't clear call");
```

Notice how this program, like all good software, checks for and reports errors everywhere they could possibly occur.

SUMMARY

We've seen how physical connections are established, used, maintained, and released, and we've examined several popular physical-layer protocols. And as we've seen, a great deal of software operates purely at this level. However, such software has certain drawbacks:

1. It usually cannot detect transmission errors. 2. It never can correct them. 3. It cannot guarantee that the receive can keep up with the sender.

We will look at solutions to these problems next time, when we visit the DATALINK LAYER.

4. THE ISO DATALINK LAYER

The physical layer is responsible for establishing and releasing a single physical connection between adjacent network nodes, and for transmitting bits in sequence between the two nodes.

The datalink layer uses the physical layer to provide point-to-point services in which physical transmission errors may be detected and possibly corrected, and it shields higher layers from the characteristics of the physical medium.

Review of ISO terminology:

- Service data unit (SDU) = info received from or passed to superior layer
- Protocol control information (PCI) = control fields added at this layer
- Protocol data unit (PDU) = SDU with PCI added to it

The higher layer gives the datalink a chunk of data (an SDU) to deliver to the partner at the other end of the physical connection. The datalink layer encapsulates the SDU (without segmenting or blocking it) within datalink protocol information fields, forming a datalink protocol data unit (DPDU), called a "frame", and calls upon the physical layer to transmit it. The datalink protocol fields always include:

- Unambiguous marking of the beginning and end of the frame
- Frame check sequence (FCS) for error detection

For instance:

```
                        +-----+-----+
 Higher Layer:          | PCI | SDU |  (PCI + SDU = PDU)
                        +-----+-----+
                              |
                              V
                 +------+-----------+-----+------+
 Datalink Layer: | FLAG |    SDU    | FCS | FLAG |  (PCI = FLAGs + FCS)
                 +------+-----------+-----+------+
                              |
                              V
 Physical Layer:     (transmits the frame)----------------->
```

Thus, the datalink layer may always be relied upon to provide error-checked transmission of the data that is provided to it.

In the example above, the flag fields define the frame boundaries and allow the frame check sequence to be identified, so that the receiver of the frame can tell whether there have been transmission errors. Since the data or control fields might contain bytes that correspond to the flag byte, the datalink layer is also responsible for transparency.

Optionally, a datalink frame may also include:

- Sequencing information
- Address information
- Control information

And, optionally, the datalink layer may do error recovery itself by reconstructing or requesting retransmission of damaged frames, or it may simply pass an error indication up to the higher layer, which may request retransmission itself.

All error-checking protocols include a datalink layer: the asynchronous Xmodem and Kermit protocols, as well as synchronous protocols like IBM Bisync, DEC DDCMP, ISO HDLC, IBM SDLC, and ANSI ADCCP. Some of these protocols predated the ISO definition for the datalink layer, and in fact the ISO definition was designed to accommodate them.

LINK ACTIVATION AND MAINTENANCE:

A datalink connection is started when the two datalink entities make contact with each other and exchange parameters concerning which protocol options they might be using, maximum frame lengths, timeout intervals, etc. If the physical link becomes inoperative during the connection, the datalink entities may monitor it until it becomes available again, then reestablish their connection and re-exchange parameters.

FRAMING AND TRANSPARENCY:

A datalink frame is marked by a unique pattern (flag) byte at the beginning, and its end is marked in any of several ways. The datalink entity is able to identify the beginning of the frame based upon the flags byte. But what happens when the flag byte occurs in the data, or in one of the control fields? How can this value be included in the packet without the frame reader prematurely identifying the end of the frame?

There are three ways to achieve transparency, and each has an associated class of datalink protocols: character-oriented protocols, byte-count protocols, and bit-oriented protocols.

In *character-oriented protocols*, a message is framed and formatted using special characters like SOH, STX, ETX, ETB, and DLE from a particular character set, like ASCII or EBCDIC. Thus these protocols are said to be "code dependent". When these characters must occur in the data, they are prefixed (typically by DLE) to indicate they are (or are not) being used for control purposes. This technique is called "byte stuffing" -- a extra bytes are "stuffed" into the message. IBM's Binary Synchronous Communications Protocol (BSC, or "Bisync") is the best-known byte-oriented protocol. The length of a message is therefore dependent on the character of the data, how many bytes must be stuffed, etc.

Byte-count protocols use a special character sequence to mark the beginning of the frame, and a length field to indicate where the frame ends. This allows for special characters within the data, and makes the length of the message independent of the contents of the data. Two well-known examples are DEC's Digital Data Communications Message Protocol (DDCMP), and IEEE 802.3 MAC. DDCMP is code-independent except for the special characters it uses to mark the beginning of the frame.

In *bit-oriented protocols*, the message begins and ends with a special bit pattern, called a "flag", which is allowed to occur nowhere else within the frame. All other fields are positionally located relative to the flags. Transparency is achieved by inserting extra bits in any byte that matches the frame byte. For example, if the flag byte is 01111110, then whenever this pattern occurs inside the frame, it is changed to 011111010 by the transmitter. The receiver deletes any 0-bit that follows five consecutive 1-bits. This technique is called "bit-stuffing". Bit-oriented protocols are code-independent; nothing in the datalink layer depends on the system's character code. Bit stuffing is the currently favored technique for achieving transparency; examples include ANSI ADCCP, ISO HDLC, and IBM SDLC. However, it requires interfaces that are not intrinsically byte-oriented, and cannot be used elsewhere (e.g. on asynchronous connections).

ERROR CONTROL:

Frames may be altered during transmission by various phenomena: noise (electrical interference), insertion of SYN "idle" characters in synchronous transmission, etc., and it may also be lost upon receipt due to buffer overruns. The receiver must be able to check whether the frame was received completely and correctly.

Error-checking schemes can be evaluated on two criteria: their efficiency in catching errors (the ratio of errors caught to those that remain undetected), and their cost in additional transmission.

There are two kinds of error control. One is a frame-check sequence (FCS), a code consisting of one or more bytes appended to the frame, also sometimes called a block check (BC) or block check character(s) (BCC). The other, usually used in forward error correction (FEC, explained below), adds error-checking information to each byte in the frame.

There are many, many FCS techniques. They vary in the length of the FCS, the method used to compute the FCS, and the reliability of the method in detecting various kinds of errors. The FCS is computed by the sender, recomputed by the receiver, and compared with the value transmitted by the sender. If the two values agree, the frame is accepted, otherwise it is in error.

The FCS is to the frame as the parity bit is to a byte -- it allows detection of some errors, allows other errors to slip through undetected, but does not allow detected errors to be corrected. The most common FCS techiques are checksum and CRC.

Checksum

A CHECKSUM is a numeric sum of all the bytes in the frame, i.e. their bit values are summed arithmetically (as if the bytes were numeric rather than character data), and the sum (possibly truncated) becomes the FCS, which may be 8 bits, 16 bits, or more. The arithmetic used in accumulating the sum may be normal addition with carries (binary 1 + 1 = 10), or modulo-2 (one’s complement) addition, in which carries are discarded (1 + 1 = 0). Here’s an example with normal addition:

```
ASCII        Decimal      Binary
Character    Value        Value
    M          77        01001101
    E          69        01000101
    S          83        01010011
    S          83        01010011
    A          65        01000001
    G          71        01000111
    E        + 69      + 01000101
Checksum =    517      1000000101
```
(note overflow out of 8-bit byte)

If 8-bit bytes are being summed, and the maximum value of an 8-bit byte is 255, then the maximum number of bytes that can occur in a message with a 2-byte (16-bit) checksum without possibility of overflow is 256. If the average value of the bytes within the message is 128, then the message may be 512 bytes long. When bits overflow from the checksum, then it does not reflect the values of the high order bits of the data bytes. Thus an arithmetic checksum’s effectiveness diminishes with the length of the message.

If modulo-2 arithmetic is used, then each bit position *n* in the checksum is the "exclusive OR" of the values of bit *n* of all the data bits, and each checksum bit has equal validity:

```
ASCII        Decimal     Binary
Character    Value       Value
  M            77       01001101
  E            69       01000101
  S            83       01010011
  S            83       01010011
  A            65       01000001
  G            71       01000111
  E          + 69     + 01000101
Checksum =     67       01000011
```

Now suppose that each of two bytes has a 1-bit error in the same bit position:

```
    Original     Error
    Bytes        Bytes

    10010101     10110101
  + 00110110   + 00010110
    11001011     11001011  <-- Checksum
```

The checksum is the same. The two errors have cancelled each other out, and the checksum will not detect the errors. This happens with both arithmetic and modulo-2 checksums.

If there have been no carries out of the checksum, or if modulo-2 arithmetic is used, and if we assume that all errors are equally likely, the probability of an error going undetected is the ratio of the number of all errors that can cancel each other out to the number of all possible errors, which works out to be 2^{-n} (two to the minus-n^{th} power), where n is the number of bits in the checksum. For an 8-bit checksum, the ratio is 1/256; for a 16-bit checksum it's 1/65536.

In practice, data communication errors rarely occur in isolated bits, so the probability of errors cancelling each other out is lower than it seems at first glance. Also, to reflect real performance, these probabilities must be multiplied by the probability that an error will occur at all. For instance if the bit error rate on a particular line is 1/100,000, then the probability that an error will occur in a given 100-byte (800-bit) message is 800/100,000 = 8/1000, and the probability of an error going undetected by a 16-bit checksum on this message is therefore:

$$\frac{8}{1000} \times \frac{1}{65536} = \frac{1}{8192000}$$

i.e. one in 8.2 million. For an 8-bit checksum (as used in Xmodem), it's:

$$\frac{8}{1000} \times \frac{1}{256} = \frac{1}{32000}.$$

The checksum is implemented very simply in software. For instance, in BASIC, if the message to be sent is in a variable M$, then the checksum is computed like this:

```
1000 C = 0
1010 L = LEN(M$)
1020 FOR I = 1 TO L
1030   C = C + ASC(MID$(M$,I,1))
1040 NEXT I
```

and then appended to the message like this (assuming it's an 8-bit checksum):

```
1050 M$ = M$ + CHR$(C AND 255)
```

and transmitted, e.g. with a PRINT statement. The BASIC functions used above are `LEN` (returns the length of a string), `MID$(x$,p,n)` (returns substring of `x$` starting at position p, length n), and `ASC` (returns the ASCII number of the given character). The `AND` operator returns the logical AND function of its two operands; `C AND 255` returns the low-order 8 bits of the number C (255 decimal = 11111111 binary).

The receiver gets the message with an `INPUT` statement and then checks the checksum as follows:

```
2000 C = 0
2010 L = LEN(M$)
2020 FOR I = 1 TO L-1
2030   C = C + ASC(MID$(M$,I,1))
2040 NEXT I
2050 IF (C AND 255) = ASC(MID$(M$,L,1)) THEN accept ELSE reject
```

For a 16-bit checksum, each 8-bit checksum byte would have to be done separately:

```
1050 M$ = M$ + CHR$((C AND -256)/256) + CHR$(C AND 255)
```

with corresponding complications in the receiving program. The number -256 selects the high-order 8 bits of a 16-bit number (-256 decimal = 1111111100000000 binary, assuming a 16-bit word). (By the way, it would have been more straightforward to use 65280 rather than -256, but IBM PC BASIC can't represent any positive integer larger than 32767 in its 16-bit word). Dividing this number by 256 shifts it to the right 8 bits, so that it can be treated like a character. This kind of manipulation is very common in data communications programming.

To further reduce the probability of self-compensating error bursts, there are variations on checksum techniques. For instance, each block of data can have two checksums, one for the odd-numbered bytes, and one for the even ones. If the probability is high that an error burst will span a byte boundary, then an error that might otherwise be self-compensating would be split into two different checksums and caught.

A similar technique, used more often in higher-level protocols, is the Fletcher checksum, which is basically a 2-byte checksum in which the first byte is a regular 8-bit arithmetic sum, and the second byte is a kind of checksum on the checksum, with both bytes then adjusted so that the sum of all the bytes in the message, including the checksum bytes, will be zero, which makes it very easy to check on the receiving end.

Cyclic Redundancy Check

A CYCLIC REDUNDANCY CHECK (CRC) is a quantity formed by treating the entire message as a large binary number and dividing that number by another binary number. The CRC is the remainder. The most common CRC is based on the divisor 10001000000100001. This number is customarily expressed as a polynomial:

$$x^{16} + x^{12} + x^5 + 1$$

where x is 2 and a superscript number is an exponent; x^{12} means "raise x to the 12th power". Polynomial notation is used for readability, and so that polynomial arithmetic can be used. The polynomial shown above is the one recommended by the CCITT, but there are other 16-bit CRCs in wide use, such as the "CRC-16":

$$x^{16} + x^{15} + x^2 + 1$$

and there are also 12-bit and 32-bit versions. The latter is becoming increasingly important, and is being incorporated into existing standards like IEEE 802.3 Ethernet and FDDI:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} +x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Most books on data communications do not explain how CRC works or why it gives the results it does; here it is, in capsule form.

The data message itself is called the "message polynomial" $M(x)$ and the divisor is called the "generating polynomial" $P(x)$. The "degree" r of $P(x)$ is the exponent of the highest term, which is 16 in the 16-bit versions. Let's work through an example in which:

$$M(x) = x^9 + x^7 + x^3 + x^2 + 1 = 1010001101$$

$$P(x) = x^5 + x^4 + x^2 + 1 = 110101$$

The CRC procedure works like this:

$M(x)$ is multiplied by x^r, which amounts to shifting $M(x)$ r bits to the left:

$$x^5 \times (x^9 + x^7 + x^3 + x^2 + 1) = x^{14} + x^{12} + x^8 + x^7 + x^5 = 101000110100000$$

In other words, r zeros are are tacked on to the right of the message. These are placeholders for the CRC that will be calculated. The resulting quantity is divided by $P(x)$, giving a quotient $Q(x)$ and a remainder $R(x)$:

$$\frac{101000110100000}{110101} = 101010110\text{, remainder } 01110.$$

The remainder is added the message using modulo-2 addition (no carries):

```
101000110100000
        + 01110
---------------
101000110101110
```

and we call the result $T(x)$, the message to be transmitted. $T(x)$ is exactly divisible by $P(x)$ because when you divide A by B and get a remainder R, then A – R is exactly divisible by B and, in modulo 2 arithmetic, A – R is the same as A + R.

When $T(x)$ is transmitted, the receiver simply divides it by $P(x)$ and checks to see if the remainder is zero. If so, no error is detected.

But what if there are transmission errors? A number of bits may be changed by noise. The noise pattern can be called $E(x)$. For a message in error, $T(x) + E(x)$ will be received (where "+" is modulo 2 addition -- no carries). If this sum is evenly divisible by $P(x)$, then no error will be detected. In fact, since $T(x)$ is evenly divisible by $P(x)$, then $T(x + \mathrm{E}(x))$ will be evenly divisible by $P(x)$ only if $E(x)$ is itself evenly divisible by $P(x)$. Therefore, $P(x)$ must be picked so that it is very improbable that a pattern of error bits will be a multiple of it. For this reason, it should be a prime number (a number divisible only by one and itself). The CCITT did extensive studies of error patterns on switched transmission lines in order to arrive at its recommended polynomial.

If a frame has a single bit in error, then $E(x) = x^i$, where i is less than the message length. If

$P(x)$ has more than one term, then x^i cannot be evenly divided by it. Thus all single-bit errors are detected.

Similarly, if a frame has two bits in error, then $E(x)$ has two terms. Therefore if $P(x)$ has three or more terms, then it cannot be evenly divided into $E(x)$, and all double-bit errors can be detected.

If $E(x)$ has any odd number of bits in error, then it can be proven that $E(x)$ is not evenly divisible by $(x + 1)$, so that any $P(x)$ that has $(x + 1)$ as a factor will catch all odd numbers of error bits.

It can also be proven that all error bursts of length less than the degree r of $P(x)$ can be caught.

And it can be shown that an error burst whose length is the same as the degree, r, of $P(x)$ can be caught except when the error burst is identical to $P(x)$. The probability of this happening is $(1/2)^{r-1}$.

Finally, for error bursts longer than r, it can be shown that the probability of undetected errors is 2^{-r}.

The formal proofs of these statements, for those who are interested, are given in reference 9.

Bits in Error	Probability of Detection	For $P(x)$ $r = 16$
Single	1.0	100.0%
Double	1.0	100.0%
Any odd number	1.0	100.0%
Error burst < r+1	1.0	100.0%
Error burst = r+1	$1-(1/2)^{r-1}$	99.99695%
Error burst > r+1	$1-(1/2)^{r}$	99.99847%

It might be noted here that an 8-bit modulo-2 checksum can be characterized as a CRC generated by

$$P(x) = x^8+1$$

and so the above analysis would apply to it as well.

The Achilles' heel of the CRC method is that whenever an error burst occurs that is a multiple of the generating polynomial, it will go undetected. Such an occurrence was thought to be very unlikely 10 or 15 years ago when the various standard checking polynomials were standardized, but the appearance of new equipment has changed the situation. For instance, in RF modems that use quadrature or phase shift techniques, a single error can propogate into 2 or 3 symbols, and in certain modem designs, the result will be as if the CRC generating polynomial had been added (modulo-2) to some part of the message, so that the resulting error will go undetected. In fact, certain modems of recent vintage generate errors which match the CRC-16 polynomial with disturbing frequency.

Software implementation of CRC calculations does not follow the mathematical model we've been looking at. The computer does not allow strings of arbitrary numbers of bits to have arithmetic performed on them. Computer arithmetic occurs within computer "words" of fixed length, and data is stored in bytes. Therefore, the polynomial division discussed above has to be simulated on a byte-by-byte basis.

Here is an IBM PC BASIC program to calculate the CRC-CCITT using a magic number, 4225 (derived from the CCITT polynomial), along with various shifts (simulated by division), ANDs, exclusive ORs, and special manipulations (as in line 5060) to achieve unsigned 16-bit arithmetic which PC BASIC is otherwise disinclined to do.

```
5000 CRC = 0
5010 FOR I = 1 TO L
5020   C = ASC(MID$(M$,I,1))
5030   FOR J = 1 TO 16 STEP 15
5040      Q = INT(CRC XOR FIX(C / J)) AND 15!
5050      Y = Q * 4225!
5060      IF Y > 32767! THEN Y = Y - 65536!
5070      CRC = (INT(CRC / 16!) AND 4095) XOR INT(Y)
5080   NEXT J
5090 NEXT I
5099 RETURN
```

Putting a RETURN statement at the end turns this code fragment into a subroutine that can be called after defining M$ (the message) and L (the length of the message).

```
4000 M$ = "This is a new message"
4010 L = LEN(M$)
4020 GOSUB 5000
4030 M$ = M$ + CHR$(((CRC AND -256)/256) AND 255) + CHR$(CRC AND 255)
4040 PRINT M$
```

The receiver gets the message and checks the CRC as follows:

```
4100 L = LEN(M$) - 2
4110 X$ = MID$(M$,L+1,2)
4120 GOSUB 5000
4130 Y$ = CHR$(((CRC AND -256) / 256) AND 255) + CHR$(CRC AND 255)
4120 IF X$ = Y$ THEN accept ELSE reject
```

This method of computing the 16-bit CRC-CCITT may not be pretty, but it works, and it's compatible with the methods used in the real world.

Here, for comparison, is a C-language CRC function. The data string is passed as argument `s`, and the length as `len`. "^" is the exclusive-OR operator, "&" is the logical AND operator, and "`x >> 4`" shifts the number `x` 4 bits to the right.

```
crcchk(s,len) char *s; int len; {
    unsigned int c, q;
    long crc = 0;

    for ( ; len-- ; len > 0) {          /* For all chars in string */
        c = *s++;                       /* Get the character */
        q = (crc ^ c) & 15;             /* Do low-order 4 bits */
        crc = (crc >> 4) ^ (q * 4225);
        q = (crc ^ (c >> 4)) & 15;      /* then high-order 4 bits */
        crc = (crc >> 4) ^ (q * 4225);
    }
    return(crc);
}
```

Forward Error Correction:

FCS methods allow errors to be detected, but they can be corrected only by requesting retransmission. There is another method that allows errors to be detected *and* corrected by the receiver. This is called Forward Error Correction (FEC). FEC carries relatively high transmission overhead, but in some situations (transmission through satellites or from spacecraft) it's worth it to avoid retransmissions.

FEC is an extension of parity. If an 8-bit byte includes 7 data bits and an even or odd parity bit, then there are only 128 valid bit combinations. In order to change one valid pattern into another, *at least two* bits must be changed. Thus a code using a single parity bit has a "minimum distance" of two between any two valid characters.

If the minimum distance is three, then any single error turns a valid code word into an invalid one, which is a distance one away from the original, and a distance two away from any other valid code word. Thus the original can be uniquely reconstructed. In a code with a minimum distance of three, any double bit error can be detected, and any single bit error can also be corrected. But how do you locate the bit that was in error?

A "Hamming code" (named after R.W. Hamming of Bell Labs, see reference 7) adds k parity bits to each m-bit data byte, to form an $m+k$ bit code, where $2^k \geq m+k+1$ (so that the number k is big enough to pinpoint the location of any of the $m+k$ bits in error, or can be 0 if there are no errors). The bits in the resulting code word are labelled in order 1, 2, 3, 4, . . . from left (most significant bit) to right (least). The parity bits go at positions 1, 2, 4, 8, . . . (powers of two). For instance, here is a code word with m=4 and k=3:

```
1 2 3 4 5 6 7 <-- code word position number  (c1, c2, . . . c7)
    x   x x x <-- data bit positions
    1   2 3 4 <-- data bit position number   (m1, m2, m3, m4)
1 1 0 0 1 1 0 <-- code word (parity mixed with data)
x x   x       <-- parity bit positions
1 2   3       <-- parity bit position number (p1, p2, p3)
```

Here's a table of the position numbers:

Code Word Error Position	Parity Bit Position Number p1	p2	p3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Looking at column p3 you can see that if an error occurs in code word bit 1, 3, 5, or 7, then the least significant bit of the parity code is 1. Similarly, if the error is in code word bits 2, 3, 6, or 7 then the middle parity bit is 1. And if the error is in bits 4, 5, 6, or 7 then the most significant parity bit is 1.

So... if p1 is used as a parity bit for (c1,c3,c5,c7), and p2 for (c2,c3,c6,c7), and p3 for (c4,c5,c6,c7), then the number formed by (p1,p2,p3) tells the position of the error bit, if any.

For example, suppose the data word 0001 was to be transmitted. Inserting the even parity bits at the right positions, we construct the code word 1101001, and transmit it, but a transmission error occurs in c5, so that coded message 1101001 becomes 1101101:

```
1 2 3 4 5 6 7 <-- code word position number  (c1, c2, . . . c7)
    x   x x x <-- data bit positions
    1   2 3 4 <-- data bit position number   (m1, m2, m3, m4)
1 1 0 1 0 0 1 <-- code word before transmission
1 1 0 1 1 0 1 <-- code word after transmission
x x   x       <-- parity bit positions
1 2   3       <-- parity bit position number (p1, p2, p3)
```

The receiver checks the parity of the three groups of bits:

Bit Group	Value	Parity
c4,c5,c6,c7	1 1 0 1	1 (odd) = p1
c2,c3,c6,c7	1 0 0 1	0 (even) = p2
c1,c3,c5,c7	1 0 1 1	1 (odd) = p3

The position number formed by (p1,p2,p3) is 101, and therefore the bit in error is c5. To correct the error, the receiver of the message simply complements this bit (i.e. changes it from 1 to 0).

This example shows how a Hamming code can be used to detect and correct single bit errors. By adding an overall parity bit for the entire code word, double bit errors can also be detected (but not corrected); when two bits are in error, the overall parity will be correct, but the position number will indicate an error (but since there are two errors, only one of them can be corrected).

In real life, we transmit 7-bit or 8-bit data bytes, and require correspondingly larger code words. For instance, with 7-bit data bytes, we can detect and correct single-bit errors with 4 parity bits, resulting in an 11-bit code word. Also, additional bits can be added to allow detection and correction of more errors per code word; in general, a code that can detect *n* errors can correct *n*/2 errors.

Since FEC cannot correct *all* errors, there still must be a retransmission request mechanism if we want totally error-free data. A single-bit correcting FEC can be used in environments where the probability of a 1-bit affecting a particular code word is significantly higher than a multiple-bit error.

In some situations, retransmission is not a viable option, for instance when voluminous data is being sent back from a probe to a distant planet. In this case, much more complicated FEC methods are used, such as the BCH (Bose, Chaudhouri, Hocquengham, used by Intelsat) and Reed-Solomon linear block codes, or convolutional codes such as the Viterbi algorithm. One space probe worker reports that the typical, near-standard, method (which was used on the Voyager spacecraft) is to first first compress the data (using the Rice algorithm), then encode using a rate 223/255 Reed-Solomon code over the alphabet GF(2^8), then interleave to depth 4 (for help with bursts), then inner-code with a 1/2 constraint length 7 convolutional code. For those who are interested, these methods are described in references 11 and 6.

Since space probes are sending back mainly picture data, it is not disastrous if some errors cannot be corrected -- we just wind up with pictures with spots on them, and even these can be fixed up using image processing techniques.

CONNECTIONLESS VS CONNECTION-ORIENTED PROTOCOLS:

We've seen how the datalink layer does framing and error detection. However, the OSI definition of the datalink layer says it also can do connection, sequencing, and flow control. We have seen at the physical layer what it means to establish and release connections, and to do flow control. But these mechanisms are a little bit different when the unit of information transfer is a structured frame, rather than a featureless stream of bits.

Most OSI protocols are "connection-oriented". That means, they proceed through three phases: connection establishment, data transfer, and connection release. When a connection is opened, a database is created for the connection, containing information like the address of the peer, the current sequence number, the flow-control status, and a retransmission buffer for the most recent (or several most recent) messages. For the rest of the connection, this database is referred to very efficiently using shorthand "pointers", much like the file numbers that programmers use after opening a file. Connection-oriented protocols assure correct and complete peer-to-peer communication. Correctly received messages are acknowledged. Retransmission of damaged or missing messages can be requested by sequence number, and duplicates can be detected and discarded based on the sequence number.

A connectionless protocol does not establish or maintain any relationship between individual data transfers. There is no sequencing, no database, no acknowledgement. How, then, can a connectionless protocol possibly work? On its own, it can't. But it can occupy one or more of the layers in a complete protocol stack. So long as there is a connection-oriented protocol above the highest-level connectionless protocol, it can take care of sequencing and retransmission.

Why would we want a connectionless protocol? Several reasons...

- On a clean, single-user at a time medium, like Ethernet, there's not much chance of messages becoming misordered, so the overhead of maintaining a connection at the datalink or network level can be avoided.
- In a packet-switched network, it is not possible to have a connection-oriented network layer, because packets will take different routes.
- If the network is totally reliable, then the transport protocol could connectionless.

Datalink protocols can be either connection-oriented or connectionless. We will look at examples of each.

FLOW CONTROL METHODS:

Datalink software must be able to avoid buffer overruns. This means it must be able to control the rate at which data is being sent to it. The XON/XOFF method used in asynchronous transmission is generally not used because (a) network protocols tend to send arbitrary 8-bit data, in which the XOFF and XON patterns can occur, and (b) in synchronous transmission, the transmitter has to keep sending constantly anyway. So datalink flow control usually means that the receiver must control the rate at which datalink *frames* are sent to it (rather than individual characters), as well as the maximum size of a datalink frame.

The maximum frame size can be negotiated during datalink connection establishment as a function of the buffering capability, but other factors might also be considered, like the quality of the physical connection. The higher the bit error rate, the greater likelihood of retransmissions. Longer packets give greater efficiency (data/protocol ratio) but are more likely to be corrupted and take longer to retransmit than shorter packets.

The frame transmission rate can be regulated in several ways:

- Stop-and-wait. Each frame must be acknowledged before the next one is transmitted. BSC and Xmodem use this method.
- Explicitly. Datalink frames can contain control information that grants or denies permission to the partner to transmit. HDLC in Unbalanced Normal Response mode is an example.
- Buffer credits. Datalink frames can contain control information specifying how many buffers are free for use.
- Window size (explained under Sequencing).

SEQUENCING:

Sequencing is an optional feature of the datalink layer. In multinode routing networks, sequencing is more properly an end-to-end rather than a point-to-point function, because packets can take different routes; a routing node has no way of knowing whether a packet is missing because it was lost in transmission, or because it took a different route.

In virtual or switched circuit networks, where all packets follow the same route, it makes some sense for sequencing to occur at the datalink level. This way, it becomes a network function, and frees the host computer from having to do it. However, the datalink layer may still omit this function and leave it to a higher layer.

For packets to be delivered to the upper layers in proper order, the datalink frames must carry a sequence number as part of their datalink protocol control information. This is typically a small number, ranging from 0 to 7 and then "wrapping around" to 0 again. In stop-and-wait systems like BSC, a modulo-2 sequence number is sufficient (a frame is either "this one" or "the previous one").

The sequence number allows the receiver of frames to determine whether any frames are missing or duplicated, and to request retransmission of missing ones and ignore duplicated ones.

Sequencing and flow control are closely related, since sequencing can be used as a flow control mechanism. In stop-and-wait connections, frame number $n+1$ is not sent until frame n has been acknowledged. However, in connections with long round-trip delays (such as satellite links), stop-and-wait is very inefficient: the faster the transmission speed, the more the (constant) delay time determines the throughput.

Therefore a sequencing-*cum*-flow-control scheme called *windowing* may be used. The sender may send many frames in a row without waiting for acknowledgment. Each unacknowledged frame is kept in a list, or "window". Transmission stops only when the window fills up. Meanwhile, the receiver is acknowledging the frames. Whenever the earliest frame in the sender's window is acknowledged, the window "slides" forward a notch. Thus, if acknowledgements arrive at a steady rate, transmission can be continuous.

The major complication with "sliding windows" is how to recover from errors. There are two methods: selective retransmit and go-back-to-*n*. In selective retransmission, each frame is acknowledged explicitly. If a frame is received in error, or a "hole" in the window prevents it from sliding forward, then a "negative acknowledgement" (NAK) is sent for the damaged or desired frame, and only that frame is retransmitted. Each frame may be retransmitted up to a specified retry threshold before the connection is given up for lost.

In the the other method, go-back-to-*n*, the receiver will only accept packets in sequence. This scheme requires fewer explicit acknowledgements; an ACK for frame *n* also acknowledges all previous frames. However, this means that a NAK for frame *n* amounts to a request for retransmission of that not only that frame but also all later frames.

The size of the window must not be larger than the range of sequence numbers, or else packet numbers would be ambiguous. With selective retransmission the window size can be no greater than $n/2 - 1$. This is because in the worst case, the sender has sent an entire windowful of frames, and the receiver has received and ACK'd them all, but all of the ACKs were lost in transmission. The receiver has already slid its window forward, but the sender hasn't. The sender, having received no acknowledgements for its window full of frames, retransmits them all. If their window size spanned the range of sequence numbers, the receiver would erroneously accept these old frames as new ones.

In go-back-to-*n*, the window size may be as large as $n - 1$. There is never any doubt about which frame the receiver is NAKing, because it only accepts frames in sequence.

CASE STUDIES:

1. XMODEM (asynchronous)

- Packet format: SOH NUM –NUM 128-data-bytes BCC
- Packet length: 132 (fixed). Xmodem has an implied byte count of 132.
- Transparency: bare 8-bit data requires fully transparent data path. Framing character (SOH) can appear in data or control fields, as can XON and XOFF, commonly used for flow control on full-duplex asynchronous connections. Transparency depends upon implied byte count.
- Error control: 8-bit checksum or 16-bit CRC, but only on data packets; responses are not error checked, nor are EOT or other control messages.
- Flow control: stop-and-wait.
- Sequencing: 1-127 (packet 0 not used), only on data packets, not on responses.
- Code dependence: Since all control fields in Xmodem packets are binary numbers, Xmodem would seem to be code-independent. However, the responses are particular ASCII characters like ACK, NAK, CAN, etc. So it's a mongrel.

2. KERMIT (asynchronous)

- Frame format: SOH LENGTH ... BCC CR, framed by SOH and CR.
- Frame length: variable, maximum 20 - 800000 (negotiated), indicated by length field.

- Transparency: 7- or 8-bit data negotiated, no control characters appear between SOH and CR, but are encoded as printable characters and prefixed (byte stuffing).
- Error control: 6- or 12-bit checksum, or 16-bit CRC negotiated, located based on packet length field + negotiated block check length. Retransmission is requested for corrupted packets, corrupted acknowledgements are ignored and cause timeout and retransmission.
- Flow control: XON/XOFF, stop-and-wait, or sliding window with selective retransmission (negotiated).
- Sequencing: Packet number 0-63 (window size 31 max) on all packets, allows matching of acknowledgement to packet being acknowledged, so that sliding window with selective retransmit is possible.
- Code dependence: Kermit control fields, including the packet length and block check, are expressed as ASCII characters.

3. IBM Binary Synchronous Communication (BSC, BISYNC) (synchronous)

BSC is a synchronous character-oriented protocol developed by IBM in the late 1960s for communication between its mainframes and batch or block-mode terminals. It can be used with ASCII or 6-bit character codes, but is most commonly used with EBCDIC (an ASCII version is standardized as ANSI X.28, 1971, 1976).

Frame formats:
SYN SYN SOH header ETB BCC PAD (pad = 11111111)
SYN SYN SOH header STX text ETB/ETX BCC PAD
SYN SYN STX text ETB/ETX BCC PAD
SYN SYN DLE STX transparent text DLE ETB/ETX BCC PAD
SYN SYN CC (CC is ENQ, EOT, etc, not error-checked)

The last format is used for responses, like ACK, NAK, as well as for connection establishment and release.

Connection establishment: Sender sends ENQ, receiver responds with ACK.

Connection release: Sender sends EOT.

Frame length: variable, delimited by control characters.

As in all synchronous protocols, a series of SYNs precedes the frame to ensure the receiver knows where the byte boundaries are. Then comes a frame enclosed in special framing characters. The frame consists of an optional header starting with SOH (Start of Header). The text (data) portion of the frame starts with STX (Start of Text) and ends with either ETB (End of Transmission Block), or ETX (End of Text, no more data blocks to be sent). The ETB or ETX is followed by the block check character(s) (BCC). The receiver checks the BCC and sends ACK or NAK in response.

Transparency: In addition to SOH, STX, ETB, and ETX, a number of characters are special in BSC, including:

ITB	End of intermediate transmission block, when messages are broken up into sections for error checking purposes. Like ETB or ETX, ITB is followed by the

	BCC. But Unlike ETB or ETX, ITB does not require a response.
EOT	End of Transmission, or multipoint line control.
NAK	Negative Acknowledgement, previous block was received in error.
ENQ	Connection establishment, or polling/addressing in multipoint connections.
DLE	Data Link Escape, primarily used to prefix special characters when they appear in data.

In normal data mode, the control characters trigger their designated functions, and SYNs (which might be "idled" by the transmitter) are discarded. If any of these special characters might appear in data, BSC must go into "transparent text" mode. In transparent mode, the control characters (including SYN) are treated as data unless they are preceded by DLE. Transparent text begins with DLE STX rather than STX alone. For example:

Text mode:	SYN SYN STX C D E SYN F G ETB BCC PAD
Transparent:	SYN SYN DLE STX C D E SYN F DLE SYN G ETB DLE ETB BCC PAD

In the first frame, the data is "C D E F G" and the SYN is discarded. In the second, the data is "C D E SYN F G ETB".

Error control: CRC-16 (for EBCDIC BSC), retransmission requested via NAK up to retry limit. CRC includes all characters following the first SOH or STX after a line turnaround, up to and including the block terminator (ASCII BSC uses vertical plus longitudinal parity, but ASCII BSC is uncommon). Timeouts handle loss of frames. Certain messages (ACKs and NAKs, EOT, etc) are not error checked.

Flow control: Stop-and-wait. Each block must be ACK'd before next one is sent (BSC is designed for half-duplex operation). There are additional rules regarding line turnaround, delayed transmission, reverse interrupts, etc.

Sequencing: ACK0 and ACK1 for even and odd blocks (ACKs in BSC are not single characters, but rather 2-character sequences). The data blocks themselves have no sequence numbers. Because the protocol is stop and wait, the sender can keep track of even and odd numbered blocks, and make sure the ACKs match.

Code dependence: BSC is a character-oriented protocol that depends heavily on the particular character code. For this reason there are separate EBCDIC and ASCII versions.

Note that clear separation of the synchronous receiver-transmitter (USRT) and the protocol is not possible. For instance, the USRT knows to send SYNs as idle characters when it has nothing else to transmit. But in transparent data mode, SYNs in the data must be preceded by DLE. But the data has already been given to the USRT by the protocol. So quite often, the USRT and the protocol are combined into a single piece of BSC hardware, which is then not usable with other protocols.

4. DEC DDCMP (asynchronous or synchronous)

DDCMP is a byte-count-oriented protocol designed by DEC in 1974. It may be used on serial or parallel, full- or half-duplex, synchronous or asynchronous connections.

Packet format: SYN SYN CLASS COUNT FLAG RESPONSE SEQ ADDRESS CRC DATA CRC

The CLASS field specifies whether the frame type: data, control (ACK, NAK, etc.), or maintenance. COUNT is the byte count. The FLAG tells the receiver whether to expect SYN characters after the frame (so it can turn on its "sync search" and "strip sync" hardware). SEQ and RESPONSE are the frame numbers of the current frame, and the frame most recently received. ADDRESS is used in multipoint connections, otherwise it is ignored. Then comes a "header CRC", which allows the packet receiver to accept or reject the length field with some confidence, followed by variable-length 8-bit transparent data (up to 16363 bytes), and then a CRC for the data.

Packet length: Up to 16373 bytes.

Framing: Frame begins with Class Flag: SOH (data), ENQ (control), or DLE (maintenance). Byte count follows immediately, defines end of frame.

Transparency: Byte count allows any data at all to follow. But transmitter must not "idle a SYN" anywhere in the packet if receiver has not been told to turn on its "strip SYN" logic.

Error Control: CRC-16 on 6-byte header, another one on data. Corrupted packets are NAK'd explicitly, or retransmitted because of timeout when not ACK'd. What if the header of a DDCMP message is corrupted. The receiver detects this because the header checksum is wrong, but then how does it find the beginning of the next message?

Flow control: Full duplex transmission allowed, up to 255 un-ACK'd frames may be outstanding. An ACK for frame n also acknowledges all previous frames.

Sequencing: Frames numbered 0-255, and each frame also carries the number of the frame most recently received in sequence (i.e. with no gaps). Bad packets may be NAK'd explicitly, or implicitly by not updating the response number. Retransmission is go-back-to-*n*.

Code dependence: DDCMP is code-independent except for the use of SOH, ENQ, and DLE as class flags.

5. HDLC, SDLC, and ADCCP (synchronous)

These three bit-oriented protocols are very similar, varying only in minor ways. HDLC is the ISO standard, ADCCP is the ANSI equivalent, and SDLC is the IBM version. In fact, HDLC and SDLC are (not necessarily compatible) subsets of ADCCP. All of these protocols date from 1973-74. Bit oriented protocols address deficiencies in character protocols like BSC or hybrid protocols like DDCMP: they are code independent, reliable (fully error-checked), flexible, and (unlike BSC) can take advantage of full duplex connections. The operation of this family of protocols tends to be rather complicated, mostly for historical reasons, since they were originally designed for the multipoint master-slave environment, and there is much ado about which "mode" a particular station is in, what commands it's allowed to issue, and so forth.

All frames are in a consistent format, and all are error-checked. The basic frame layout is:

```
+------+---------+---------+-------------+-----+------+
| FLAG | ADDRESS | CONTROL | INFORMATION | FCS | FLAG |
+------+---------+---------+-------------+-----+------+
```

The flag is the 8-bit quantity 01111110. The other fields are in fixed positions relative to the flags, and of fixed sizes, except for the information (data) field which can contain zero or more bytes. The minimum size frame is six bytes. There is no theoretical maximum.

Transparency: The only byte that must not occur within a frame is the flag byte itself, 01111110. To prevent such an occurrence, any sequence of five consecutive 1-bits that appears between the flag bytes has a 0-bit inserted after it by the transmitter, and any 0 bit appearing after 5 consecutive 1-bits is stripped by the receiver.

Error Control: 16- or 32-bit CRC-CCITT frame check sequence (FCS).

Addressing: The normal address field is 8 bits, but with "extended addressing" it may be extended to any number of bytes, by setting the high order bit to 1 when another byte follows, and to 0 in the final byte.

Protocol: the Control Field is normally 8 bits, in this format:

```
+-----------------------------------------------+
|  8     7     6     5     4     3     2     1  | Bit number
+-----------------+-----+-----------------+-----+
| Receive Count   | P/F | Send Count      |  0  | Bit(1)=0: I-frame
+-----------------+-----+-----------+-----+-----+
| Receive Count   | P/F | Function  |  0     1  | Bits(2,1)=01: S-frame
+-----------------+-----+-----------+-----------+
| Modifier M1     | P/F | M2        |  1     1  | Bits(2,1)=11: U-frame
+-----------------+-----+-----------+-----------+
```

There is also a 2-byte extended format, in which the count fields are increased from 3 to 7 bits (why is this desirable?). The low order bits tell what type of frame we have:

- Information (I) frames, used for data transfer. These include the data to be transferred, along with send and receive counts for sequencing and flow control.
- Supervisory (S) frames, used for acknowledgement, error recovery, and flow control. These include RR (receive ready), RNR (receive not ready), REJ (reject, i.e. go-back-to-*n*), and SREJ (selective reject). RR and RNR are found in all implementations, but REJ and SREJ are useful only in two-way-simultaneous (TWS, i.e. full duplex) connections, and even then are optional. S-frames include a send count, but the receive count is replaced by a function code (RR, RNR, REJ, or SREJ).
- Unnumbered (U) frames, used to control the link itself. These include mode-setting commands, connection and disconnection, link reset, and parameter exchange (XID). U-frames have "modifiers" (function codes) in place of both send and receive counts. There is also a "frame reject" U-frame, used to negatively acknowledge invalid frames.

Datalink connections can be "unbalanced" (master/slave) or "balanced". In an unbalanced configuration, one station may send only "commands" and the other may only send "responses" (most important functions, like I, RR, RNR, and REJ, are considered both commands and responses). In balanced configurations, both stations can send commands and responses.

Unbalanced configurations can be in "normal response mode", in which they use the Poll/Final (P/F) bit to control line access, or "asynchronous response mode" (anyone can transmit any time). (Note, the use of the word "asynchronous" in this context has nothing to do with asynchronous serial transmission, it just means that frames are exchanged without any special synchronization between the two partners.) Balanced configurations are always in asynchronous response mode. There are three "classes of procedure", one for each of these combinations.

P/F	Master (Poll)	Slave (Final)
0	Slave should not answer	More messages to come
1	Slave should answer	Final message of sequence

Use of the P/F Bit

These protocols incorporate a number of commands, sent in S-frames, to control the flow of frames, including Receive Ready (RR, grants permission to send), Receive Not Ready (denies permission to send, or requests that sending stop), Reject (REJ, valid frames rejected because earlier ones missing), and Selective Reject (SREJ, specified frame rejected). RR and RNR are appropriate to Two-Way-Alternate (TWA, i.e. half duplex) connections, and REJ and SREJ are appropriate to Two-Way-Simultaneous (TWS, i.e. full duplex) connections.

Sequencing, flow control, and error recovery are accomplished via the send and receive counts. Each partner numbers its own frames using a 3-bit modulo-8 counter (or, in extended versions, 7-bit modulo-128), and also includes in its message the number of the most recent valid frame it has received. Corrupted frames need not be explicitly NAK'd, rather, the receiver simply does not update its receive count. The sender eventually retransmits.

In full duplex connections, the sequencing scheme allows sliding windows, either with go-back-to-*n* (REJ) or selective retransmission (SREJ), depending on the capabilities of the partners.

The window size is negotiated at connection establishment. It may be as large as *modulo*–1 for go-back-to-*n* (REJ), or *modulo*/2 – 1 for selective retransmission (SREJ). The larger the window size, the better the chances for continuous transmission. Smaller window sizes may be used for flow control.

SDLC is generally used in unbalanced normal-response mode, in accordance with IBM's pervasive philosophy, which stresses centralized management and control, hierarchical topology, and half-duplex communication. This is a reflection of the early development of SNA, which assumed an environment consisting of a central mainframe and many attached polled multipoint terminals and devices, rather than a general host-to-host (peer-to-peer) network. SDLC implementations generally do not include the addressing and count field extensions, nor the 32-bit FCS, nor selective reject (SREJ). SDLC is specified in "IBM Synchronous Data Link Control General Information", IBM document GA27-3093 (there may be a newer version).

HDLC, and in particular its subsets LAP (Link Access Procedure), LAPB (an improved LAP), and LAPD, are widely used. LAPB is an asynchronous balanced implementation of HDLC used in X.25 networks, and LAPD is the link level for ISDN.

ADCCP (reference 1) in its fullness serves more as a reference model than a datalink protocol. Even its subsets, like HDLC, generally support only a few of the many possible options. HDLC is specified in ISO standards 4335, 7809, and 7498, which replace some older versions and allow for the various extensions (modulo-128 counts, etc).

6. ANSI/IEEE 802 Standards

For local area networks, standards bodies (ANSI and the IEEE) have divided the datalink layer into two sublayers: Media Access Control (MAC, the lower sublayer), and Logical Link Control (LLC, the upper sublayer).

ANSI/IEEE Standard 802.3-1985 is a MAC standard for Ethernet or other broadcast networks, in which contention is resolved using CSMA/CD. There are equivalent standards for token bus (802.4), and token ring (802.5, ANSI X3.139).

MAC handles access to the physical medium employing the appropriate contention technique (like CSMA/CD), and it handles framing and error checking. In Ethernet, for instance, a frame looks like this:

Name	Length
PREAMBLE	7 bytes, for circuitry synchronization
SFD	1 byte, Start Frame Delimiter = 10101011
DESTINATION ADDRESS	2 or 6 bytes
SOURCE ADDRESS	2 or 6 bytes
FRAME LENGTH	2 bytes
LLC DATA	variable
PAD	enough to achieve minumum frame size
FCS	32-bit CRC

The preamble serves a purpose similar to the SYNs that precede a block in synchronous transmission -- to get the receiving circuitry prepared for the data that is about to arrive. The start-frame delimiter is the reference point by which the other fields (addresses, length, data, and FCS) are located. So Ethernet frames achieve transparency via byte count, like DDCMP.

An FDDI frame is more like an HDLC frame (see reference 2):

Name	Length
PREAMBLE	8 or more bytes, for circuitry synchronization
STARTING DELIMITER	1 byte
FRAME CLASS	8 bits, C L F F Z Z Z Z
DESTINATION ADDRESS	2 or 6 bytes
SOURCE ADDRESS	2 or 6 bytes
INFO	variable
FCS	32-bit CRC
ENDING DELIMITER	1 byte
FRAME STATUS	1 or more bytes

but the (draft) standard does not state how transparency is achieved.

The MAC sublayer ignores frames with incorrect CRCs, and therefore delivers only correctly-transmitted frames to the LLC sublayer. Transparency is achieved using a byte count. The addresses are Ethernet hardware addresses (every Ethernet controller in the world is supposed to have a unique address).

ANSI/IEEE Standard 802.2, Logical Link Control (LLC), describes two protocols that may be used at the upper datalink sublayer. A "connectionless" service (Type 1 operation) allows PDUs to be exchanged without the need for datalink connection establishment, and without acknowledgement or error recovery. All of this is left to the higher layers.

Type 2 operation is similar to asynchronous balanced HDLC operation, with modulo-128 sequence numbers, but without the framing or error checking, which is done by the MAC sublayer.

Thus the IEEE 802 and ANSI FDDI standards assign the "classic" datalink functions -- framing and error-checking -- to one sublayer (MAC), and the "higher-level" functions -- connection establishment, flow control, sequencing, and error recovery -- to another (LLC).
<u>REFERENCES</u>:

1. ANSI X3.66-1979, "Advanced Data Communications Control Procedure", American National Standards Institute, 1430 Broadway, NYC 10018.

2. ANSI X3.139-198x, Fiber Distributed Data Interface (FDDI) Token Ring Media Access Control (MAC)", X3 Project 380-D Draft Proposed Standard, 1986.

3. ANSI/IEEE Std 802.2-1985/ISO-DIS 8802/2, "Logical Link Control", IEEE and Wiley-Interscience, 1984.

4. Brodd, W.D., *Operational Characteristics: BSC versus SDLC*, Data Communications, Oct 1983.

5. Brodd, W.D., *HDLC, ADCCP, and SDLC: What's The Difference?*, Data Communications, Aug 1983.

6. Edelson, R.E., et al., "Voyager Telecommunications: The Broadcast from Jupiter," in Science, V 204 # 1, June 1979.

7. Hamming, R.W., *Error Detecting and Error Correcting Codes*, Bell System Technical Journal, v29, pp.147-160, April 1950.

8. Meijer, A., and P. Peeters, "Computer Network Architectures", Computer Science Press, 1982, pp.20-21 (datalink overview), 36-51 (HDLC), 148-156 (DDCMP), and 358-360 (ISO datalink layer definition).

9. Martin, J., "Teleprocessing Network Organization", Prentice-Hall, 1970.

10. McNamara, J.E., "Technical Aspects of Data Communications", 2nd ed., Digital Press, 1982, pp.146-167 (Bisync, DDCMP, SDLC).

11. Odenwalder, Joseph P., "Error Control", chapter 10 of "Data Communications, Networks, and Systems", T.C. Bartee, Editor, Howard W. Sams & Co., 1987.

12. Robinson, J., "Reliable Link Layer Protocols", Network Working Group RFC935.

5. OSI LAYER 3 - THE NETWORK LAYER

The network layer and its sublayers (layers 1-3) form the "communications subnetwork", which is often implemented outside of the host computers. It provides a kind of "data pipe" between end systems, allowing them to communicate with one another without regard for the characteristics of the underlying communication media, or the topology of the network.

Recall that the physical layer delivers bits, in sequence, from one node to the next, and the datalink layer encapsulates these bits into messages that can be error-checked.

```
7. Application                                                  Application
      |                                                              ^
      V                                                              |
6. Presentation                                                 Presentation
      |                                                              ^
      V                                                              |
5. Session                                                      Session
      |                                                              ^
      V                                                              |
4. Transport                                                    Transport
      |                                                              ^
      V                                                              |
3. Network        Network        Network        Network        Network
      |             ^  |           ^  |           ^  |              ^
      V             |  V           |  V           |  V              |
2. Datalink       Datalink       Datalink       Datalink       Datalink
      |             ^  |           ^  |           ^  |              ^
      V             |  V           |  V           |  V              |
1. Physical------>Physical------>Physical------>Physical------>Physical
```

The media connecting the nodes in a network, and for that matter the associated datalink protocols, may be different for each hop:

```
                             +--------+
Ethernet                     |        | Leased Synchronous Line to Cleveland
-----------------------------+  Node  +----------------------------
802.3 LLC Type 1 datalink    |        | HDLC datalink
                             +--------+
```

The network layer routes packets from node to node through the network until they reach their ultimate destination (end system). The network layer must know in which direction to relay a packet, which means it must know not only the layout ("topology") of the network, but possibly also the prevailing conditions -- which nodes are up, which are down, which of several possible routes is the least congested.

Like any ISO layer, the network layer does two major things:

1. Provides a standard interface to the upper layers, which shields them from having to know anything about the lower (datalink and physical) layers.

2. Does its own work, in this case routing packets through the network.

NETWORK LAYER INTERFACE TO THE TRANSPORT LAYER

You can think of the network layer as a subroutine called by the transport layer. It provides an interface that is independent of underlying communication medium in all things other than quality of service (e.g. you can't ask for 10Mbps from a 9600 baud line, but transport layer uses the network the same way for all types of connections). The network layer is invoked with parameters like:

- Desired operation: establishment (OPEN), maintenance (exchange of network information), and release (CLOSE) of network connections, which appear to be point-to-point connections to the transport entities. And of course, data transfer -- transparent data transfer between transport entities.
- When OPENing a connection, Quality Of Service (QOS) parameters, incl. permissible error rate, throughput, delay, sequencing, etc. A bursty interactive application like virtual terminal service might choose a low-delay (no-satellite) link, whereas batch-mode applications might opt for high bandwidth, regardless of delay. For a given type of service, there is a known cost -- very important in management of long-haul network connections. On public packet-switched networks, which provide layer 1-3 service, charges are at known rates -- so many cents per packet.
- When doing data transfer, address of message to send, and network address to send it to. The network layer uniquely identifies each of the end systems (transport entities) by their network addresses, possibly independently of the addressing used by underlying layers.
- When doing maintenance functions, request for expedited transfer of (limited-size) NSDUs, subject to separate flow control constraints (optional), usually used only for network management.

The "network layer subroutine" will return some code as to whether it succeeded or failed, within the limits set by the quality of service parameters.

FUNCTIONS OF THE NETWORK LAYER

This is the work that the network-layer subroutine actually does, carried out by peer-to-peer protocol of the network layer.

- ROUTING & RELAYING within a network, and possibly between networks.
- MULTIPLEXING network connections onto a single datalink connection.

  `(...show picture...)`
- SEGMENTING or BLOCKING may be done, so long as NDSU boundaries are preserved, i.e. so long as it is done transparently to the transport layer. When segmentation is done, then so must sequencing. (Why?)

  `(...show picture of blocking, segmenting...)`
- SEQUENCING, when requested by transport entities, or when segmenting.
- FLOW (CONGESTION) CONTROL.
- Error detection, based on error notification from datalink layer, possibly augmented according to quality of service (QOS) parameters.
- Error recovery, depending on QOS parameters.

- Network layer management.

The network layer services the information given to it by the transport layer by encapsulating it in network-layer protocol information, to form a "packet", and then gives the packet to the datalink layer for transmission.

```
                                   +-----+------+
Transport Layer:                   | PCI | TSDU |
                                   +-----+------+
                                         |
                                         V
                             +-----+------------+
NETWORK LAYER:               | PCI |    NSDU    |  "packet" = NPDU
                             +-----+------------+
                                   |
                                   V
                +------+-------------------------+-----+------+
Datalink Layer: | FLAG |          DSDU           | FCS | FLAG |  "frame"
                +------+-------------------------+-----+------+
                            |
                            V
Physical Layer:   (transmits the frame)----------------->

(SDU = Service Data Unit)
(PDU = Protocol Data Unit)
(PCI = Protocol Control Information)
```

The packet finds its way through the network to the destination system using the prevailing routing strategies, known to and implemented by the network layer.

The network layer allows transport entities to transfer data between themselves, independent from routing and relay considerations within a network or between adjacent networks, shielding them from knowledge of how the underlying datalink connections are used.

Let's concentrate on the two primary functions of the network layer: ROUTING and CONGESTION CONTROL.

On simple point-to-point connections, such as those between Kermit or Xmodem partners, there's no question of routing. Each station is connected directly to the other, so the network layer is "null".

On BROADCAST NETWORKS like Ethernet, or networks like Token Ring or Token Bus, in which all stations are connected to a common medium and all "visible" to each other, access is controlled by the Media Access Control (MAC) function of the datalink layer. In this case, the network layer has little or nothing to do except address translation (Address Resolution Protocol), and possibly internetwork routing, discussed later.

ROUTING

When a packet arrives at a node, the node must decide what to do with it. Either the packet is for itself (in which case, it is absorbed), or it is for another node. In that case, the question is simply: on which line should the packet be forwarded?

```
                     ^
                     | Line A
                     |
                  +--+--+
     Arriving     |     | Line B
     ----------->+     +----------->
     Packet       |     |
                  +--+--+
                     |
                     | Line C
                     v
```

How does the node make this choice?

<u>Routing Tables</u>

Network packets contain a source and destination address, or just a destination address, or a virtual circuit number. Each routing node uses this information to decide which line to forward the packet on. It looks up the address in its routing tables. If the address is only reachable via one route, then that one is used. Otherwise, it must make a choice.

In a star network, it's simple:

```
      104
     / | \
    /  |  \
  102 103 104
```

Example of a routing table for node 104, and the corresponding network (NEXT means "next node", i.e. which line to send the packet on).

ADDRESS	NEXT
100	101
100	103
101	101
102	102
103	103
103	101
104	-
105	102

```
104----------101
 | \         /\
 |  \       /  \
 |   \     /    \
102   \   /      \
 |     \ /        \
 |     103--------100
 |
105
```

With a table like this, the relay node "simply" looks up the destination address and then picks one of the entries for that address, using some criterion to choose the best path.

Since there may be more than one path through the network from one node to another, there is always the possibility that packets will get into a "LOOP". This can happen in the illustration above if 102 wants to send a packet to 100 and:

```
 104 chooses 101 as the best route to 100
 101 chooses 103
 103 chooses 104
```

Seems silly. How could this happen? There are many possible scenarios, but they all boil down to the fact that we have many computers operating independently, and their routing tables might not be consistent.

Where do the routing tables come from? They may be:

. maintained locally (by the manager of each system), . maintained centrally and periodically downloaded to each network node, . maintained in a distributed way.

<u>Styles of Routing</u>

There are two major styles of routing: fixed and dynamic, and table management may be either centralized or distributed:

```
           Centralized   Distributed
         +-------------+-------------+
   Fixed |             |             |
         +-------------+-------------+
 Dynamic |             |             |
         +-------------+-------------+
```

In practice, there are several popular routing styles:

- source (sender specifies route)
- fixed (static, directory)
- adaptive (fixed, but can adapt to topology changes)
- virtual circuit (VC) (dynamic setup, fixed thereafter)
- dynamic (can adapt to changing traffic conditions)

In source routing (rarely used), the sender of a packet specifies the entire route to take. The best-known example of source routing is the address field on Unix UUCP mail ("foo!bar!baz!blort!cucca!christin"). Source routing is also used in network management, e.g. when the network layer wants to see if a message can be sent along a certain route, and how long it will take.

FIXED ROUTING is used in many real networks. Often, the routing tables are coordinated by letters or phone calls, or sometimes not coordinated at all. Once a network grows sufficiently large, a "network information center" tends to emerge, whose main function is to keep track of the host and routing tables, and feed them periodically to the members.

These procedures require each system manager to install the changes manually. There's no guarantee that this will happen, and certainly not that it will happen everywhere at the same time, so the routing tables tend to become increasingly inconsistent.

If routing table updates can be applied automatically from a "routing control control center", then a fixed-routing network can adapt to changing network conditions -- not only topology changes, but even traffic patterns, if the updating takes place frequently enough. For instance, each node might send a message to the control center once a minute, telling its queue lengths, packets transmitted per second, etc, and the center can recompute the optimum routes and download them to the nodes.

But there are always tradeoffs. Obviously, the network would depend heavily on the continuous and correct operation of the control center. And the larger the network, the bigger and more powerful the control center must be, the higher the network bandwidth in its vicinity.

Finally, even with with automatic update of routing tables, it is possible that some nodes will get the message later than others, allowing the network to operate at times with inconsistent routing tables, which can result in looping.

Virtual Circuit Routing

In VC networks (of which X.25 networks are the best-known example), the route is determined dynamically at CONNECTION ESTABLISHMENT, and remains fixed throughout the connection. Each component of the network - node, gateway, host, interface, etc -- must maintain state information for each virtual circuit that goes through it, including routes, addresses, sequence numbers, flow control status. The VC number is a shorthand pointer to this data structure, so that packet relay is an easy chore for routing nodes (like a file descriptor).

In a VC, packets are always delivered in order. This is ensured by SEQUENCE numbers and by the fact that packets can't overtake each other if they all follow the same route.

CONGESTION can be controlled because each node knows what connections it must allocate resources for.

The DISADVANTAGE of VC routing is that resources (transmission time, buffers, etc) can go idle on a given path, while other routes may be badly congested. Recovery from "bad" errors on a virtual circuit must be done with a "reset" in which all nodes on the path reinitialize their data structures.

Virtual circuits are appropriate when relatively long-term connections are needed, e.g. for bulk data transfers, virtual terminal sessions, etc. The simplified routing is more efficient, and worth the extra circuit setup time.

Dynamic Routing

Dynamic routing is the key feature of "connectionless" networking (found on local area networks, and on wide-area networks like the ARPANET), in which there is no connection establishment or release phase at the network layer. Each packet finds its own way through the network, and packets may be delivered out of order to the end system. Each packet must carry a complete destination address (such packets are called "datagrams"), and each node must do potentially complicated routing lookups and computations. The advantage is that this strategy allows resources to be allocated more fairly network wide. However, the burden falls on the end system, rather than the subnetwork, to ensure reliable data delivery via sequencing, retransmission request, etc. Connectionless networking is appropriate to short, limited communications: database transactions, one-to-many communication (multicast, broadcast), sampling of scattered data sources, and internal network functions (nodes exchanging routing information).

The most common dynamic routing strategy is called LEAST-COST routing. Costs are assigned to each link, and the path of least cost is used. If the cost of each link is the same, then the cost from A to B is equivalent to the number of hops -- this is called "shortest-path" routing.

More often, costs are assigned based on the characteristics of the line: its speed, error rate, length, or even the monthly bill.

Costs may be fixed per link, or adapt to changing conditions (delays, noise). Packets may be stamped with departure and arrival times to help nodes calculate costs. Nodes send special update messages to each other about changing costs; the bigger the network, the more of its bandwidth is consumed by these updates. Here is an example routing table with costs assigned to each link:

ADDRESS	NEXT	COST
100	101	10
100	103	15
101	101	3
102	102	7
103	103	12
103	101	27
104	-	0
105	102	11

```
        (3)
  104----------101
   | \         /\
(7)|  \       /  \
   |   \(12) /    \(7)
  102   \   /(24)  \
   |     \ /        \
(4)|     103--------100
   |           (3)
  105
```

The looping problem crops up here too. Least-cost algorithms, when applied continuously within a network, can change costs in ways that can cause loops. In the diagram below, for instance, if a message is to go from A to D, it will travel the path (A,B,C,D) of lowest cost (1+3+4=8). However, suppose that while the packet is enroute from B to C, the cost of the (C,D) link changes from 4 to 20. Then C will send the packet back to B. But B will send it back to C, etc etc (assuming that B had not yet learned the new cost from C to D).

```
     B
    /|\
(1)/ | \(10)
  /  |  \
 /   |   \
A    |(3) D
 \   |   /
  \  |  /
(5)\ | /(4)
    \|/
     C
```

This situation will correct itself eventually, if B finds out the new cost of the (C,D) link. But if updates occur infrequently, our poor packet will resonate between B and C many, many times.

Some network protocols avoid looping by maintaining a "node visited" list in the network packet. Since this technique can be very expensive, it is usually used only in virtual circuit networks, where it needs to be done only once on the call setup packet. But a loop itself might change conditions sufficiently to cause itself to be broken -- our "resonating packet" might raise the BC link cost beyond the BD cost, and then B would finally send the packet straight to node D. There are also other loop elimination techniques, including node-counts, packet lifetime control, optimality principles, etc.

FLOW CONTROL AND CONGESTION CONTROL

Besides routing, the other important function of the network layer is flow and congestion control. In fact, congestion control is a central issue in dynamic routing, where the object is to choose the least congested route.

Flow Control Mechanisms

There are two basic flow control mechanisms, the same ones used at the datalink level: windowing and explicit flow control commands. The send and receive sequence counts that appear in HDLC frames can also be used in connection-oriented network packets. When a network node does not want to receive any more packets over a particular network connection, it can simply withhold acknowledgement by not updating its receive count. Eventually, the sender's window will fill up and it will stop transmitting. If incoming traffic needs to be halted immediately, a special network-level command (equivalent to datalink commands like RNR) can be sent.

Transmission Queues

Because flow control occurs, each network node must have a mechanism for storing (buffering) packets that can't be transmitted right away. This mechanism is called a "queue", or FIFO (first-in-first-out) list. Network nodes, like any computers, have limited memory, and so the queues are of finite size. The network node maintains a separate transmission queue for each line. When a packet comes in, the node determines from its routing strategy which line to forward it on, and then places it in the queue for that line. If the queue starts to get too full, then the node will want to refuse packets that need to be forwarded on that line until the queue can be emptied. Using RNR, or "source quench", or withholding acknowledgments, for these packets will move the queue "upstream".

Why must flow control occur on both the datalink level and the network level between two adjacent nodes? Let's look at a real network protocol for an illustration.

THE X.25 NETWORK PROTOCOL

CCITT Recommendation X.25 does not describe a network architecture, but rather an "interface" between a DTE (that is, a computer) and a DCE which is itself an interface to a packet-switched network. In other words, X.25 describes the services it provides to the transport layer, but gives no particulars about the methods used. In a way, it's almost like RS-232, which specifies the interface between a terminal and a modem -- the terminal is totally ignorant of how the modems deliver the data to the terminal on the other end.

```
+-----+ X.25 interface +-----+   (         )   +-----+ X.25 interface +-----+
| Dte +----------------+ DCE +---( network )---+ DCE +----------------+ DTE |
+-----+                +-----+   (         )   +-----+                +-----+
```

The X.25 protocol has three layers:

1. physical (X.21),

2. datalink (LAPB, a subset of HDLC), and

3. network (X.25 itself).

The X.25 network protocol is the most widely cited (and possibly implemented) example of a network layer protocol. There are those who contend that X.25 is really more like a transport protocol, since it delivers error-free packets in sequence end-to-end, and is not concerned with routing.

Types of Service

X.25 provides three types of service: Permanent Virtual Circuit (PVC), Virtual Call (or Virtual Circuit, VC), and Fast Select. PVC means the path between two end systems is always available. VC means that a "call" must establish the circuit, then it is used, then it is released. Fast select means a circuit is established, used, and released all in a single message. There is no datagram ("connectionless") service in X.25. But that does not mean datagrams cannot be used "underneath" it at the datalink level (as they are in certain X.25 networks, like Canadian Datapac).

Addressing

The end system (DTE) has a network address. Between the DTE and the DCE are 4096 "logical channels", allowing for up to 4096 network connections per DTE. A process initiating a connection simply chooses a free logical channel.

Packet Format

X.25 packets are transmitted in the I-field of LAPB datalink packets. Recall that LAPB is an "asynchronous balanced" subset of HDLC. Here's a LAPB packet:

```
+------+---------+---------+------------+-----+------+
| FLAG | ADDRESS | CONTROL | INFORMATION | FCS | FLAG |
+------+---------+---------+------------+-----+------+
                                  ^
                                  |
                       X.25 packet goes in here
```

(Looks just like an HDLC packet.) Recall that the HDLC CONTROL field contains Send and Receive sequence numbers:

```
   8     7     6     5     4     3     2     1    Bit number
+-----------------+-----+-----------------+-----+
| Receive Count   | P/F | Send Count      |  0  | Bit(1)=0: I-frame
+-----------------+-----+-----------------+-----+
```

Now, inside the HDLC INFORMATION field goes the X.25 packet, which looks like:

```
  1             2         3         4 ...         Byte number
+--------------+---------+---------+-------------+
| Q D xx GROUP | CHANNEL | CONTROL | INFORMATION |
+--------------+---------+---------+-------------+
```

The first four bits are called the Group Format Indicator (GFI):

- The Q (qualifier) bit allows data packets to be split into two flows, e.g. command and data.
- The D bits requests end-to-end (rather than DCE-to-DTE) confirmation.
- The xx bits tell whether the packet has 3-bit (xx=01) or 7-bit (xx=10) sequence numbers.

The GROUP (4 bits) and CHANNEL (8 bits) fields specify the logical channel number (12 bits total, 0-4095) of the network connection. The INFORMATION field can be up to a maximum length chosen by the subscriber (16-4096 bytes).

The X.25 CONTROL field looks a lot like the HDLC CONTROL field:

```
   8     7     6     5     4     3     2     1     Bit number
+-----------------+-----+-----------------+-----+
| Receive Count   |  M  | Send Count      |  0  |  Bit(1)=0: data packet
+-----------------+-----+-----------------+-----+
```

The xx bits of the X.25 packet are like a length indicator for the CONTROL field. If the xx bits of the GFI are set to 01, then the count fields are 3 bits each, and the CONTROL field is one byte long. If xx is 10, then the counts are 7 bits each and the CONTROL field is two bytes long. The M (more) bit occupies the same position as the P/F bit in the HDLC control field, and performs an analogous function -- it means that there is "more to come". This is used when DTE packets are segmented by the network layer in order to fit into small network node buffers, so that the receiver will know how to reconstruct the original packet upon receipt.

X.25 CONNECTION ESTABLISHMENT AND RELEASE

On virtual circuits, a "call" is placed from one DTE to another using a Call Request control packet, choosing a free logical channel for this purpose. The called DTE's address is in the information field of the call request packet. Addresses are assigned according to the X.121 Numbering Plan:

```
P ZXXX NNNNNNNNNN
```

which specifies world zone, country, network within country, and DTE within network. If the call is completed (answered), a virtual circuit has been set up on that channel, and all packets containing that channel number will travel on that virtual circuit to the called DTE. Calls are terminated, and virtual circuits and logical channels released, via a Clear request control packet.

If just a very short message is to be transferred, Fast Select service may be used, in which call setup, data transfer, and call release are all combined into a single packet.

X.25 DATA TRANSFER

After call setup, data is transferred in "data packets". The GFI bits are of special interest here. If the Q bit is set to 1, then the information field of the packet contains control information (e.g. commands to the PAD), otherwise it contains user data. The D bit means that the packet should be confirmed by the end system, rather than the local DCE (this is a can of worms...). The M bit is used to indicate that a packet has been segmented and must be reassembled on the other end.

If unrecoverable errors occur, the logical link may be "reset" by a special command packet. This means that all sequence numbers go back to zero, and all packets currently in transit are discarded. X.25 itself cannot recover from a reset; it is left to the transport layer to decide which packets to retransmit.

X.25 FLOW CONTROL

Flow control occurs only during data transfer phase. Clear, Reset, or Restart command packets are not subject to flow control, and may "pass" data packets, which can cause them to be lost.

Explicit flow control occurs only between the DTE and DCE -- not end-to-end. It can be accomplished by explicit commands -- RR (Receive Ready), RNR (Receive Not Ready), as in HDLC) -- and is also achieved by windowing, which in turn depends on packet sequence numbers (send and receive counts). End-to-end flow control occurs only as a result of "back pressure"... DTE1 sends RNR to DCE1, so eventually DCE1's buffers fill up, and it tells DCE2 to stop sending, and eventually DCE2's buffers fill up too, so it sends RNR to DTE2.

Why does each packet have two sets of send and receive counts? The LAPB (datalink) numbering is sequential for all packets sent between DTE and DCE, i.e. it does not distinguish among virtual circuits, because the datalink layer does not know they exist! The X.25 packet sequence numbers are per virtual circuit.

The X.25 sequence numbers allow the system at the other end of the network connection to determine whether all packets have arrived in sequence, and it allows flow control to be performed at the network level, per network connection. For instance, if the DCE had several paths into the network, and one of them was congested, it should not prevent the DTE from transmitting over logical connections that use the other, uncongested paths.

```
                  C
                 /   <---(congested path)
DTE             /
 A-----------B-----D
           DCE \
                 \   <---(clear path)
                  E
```

If bit 1 in the control field is 1, then the X.25 packet is a control, rather than data, packet. The RR, RNR, and REJ control packets are used for explicit flow control in X.25 network protocol, just as they are in HDLC and LAPB datalink protocols. RR means Receive-Ready, and also acknowledges all packets up to the one indicated in its receive count. RNR means Receive-Not-Ready (stop sending packets!). REJ (optional) requests retransmission of all packets starting with the indicated sequence number.

Flow control between the DTE and DCE is therefore accomplished in two ways: window size and explicit RR and RNR commands. The receiver controls the rate at which packets are sent to it. It will slow down this rate if (a) packets arrive faster than it can process them, or (b) it has been flow-controlled by the next receiver along the virtual circuit. Thus congestion along the network can be avoided as the flow control mechanism propogates "upstream". (Think of a traffic jam on the LIE...)

X.25 ROUTING

Routing, the classic function of the network layer, is simply not an issue in X.25. The DTE "calls" the other DTE and gets a virtual circuit through a packet switched network, the details of which are entirely unknown.

OTHER EXAMPLES OF NETWORK LAYERS

Networks appear to fall into two categories: those that make the customer worry about routing, and those that take care of it themselves. IBM SNA and DEC DNA impose a lot of management headaches on the users. Typically, each routing node requires detailed configuration by its owner, on a continuing basis.

Other networks, like X.25 and Arpanet, provide the customer with a "communications subnetwork" that handles the physical, datalink, and network layers, plus a simple interface between the host and the network.

In all cases, however, there must be agreement among all members of the network as to what the address of each host is, and preferably also a common mapping of host name to host address, so that users don't have to remember long cryptic numbers when communicating over the network.

IBM SNA

SNA networks are divided hierarchically into sections called subareas. This structure mirrors IBM's physical network architecture, which is also hierarchical; subarea nodes are typically the hubs of star-shaped clusters, so that routing within a subarea is very simple. SNA routing functions (called Path Control) are partitioned accordingly. Virtual Route Control makes logical links among subareas, multiplexing the various sessions and handling flow control (using windows that can dynamically grow and shrink). Explicit Route Control determines the actual physical routes used; the routing tables are predefined, not dynamic. Finally, Transmission Group Control maps multiple physical links into one logical link, so that multiple links between two subarea nodes can be used to increase throughput. SNA architecture originally allowed only 64 subarea hosts and 64K "logical units". Networks soon grew too large for this restriction, now with "extended addressing" there can be 256 hosts and 8M logical units.

DEC DNA

Originally, a DECnet network could have up to 254 nodes. Demand for network connections soon outstripped supply, so Phase IV of DECnet allows up to 63 areas, with 1022 nodes within an area. Routing occurs at the DECnet "Transport" level (misleading terminology), according to a forwarding database maintained by the system manager in each routing node. There may also be "end nodes" that do not perform any routing functions. Paths are fixed along the chosen route unless (a) the network topology changes (a system goes down, or a line is broken), or (b) the cost assigned to a given link is changed (by operator command). Loops are avoided via a field in the packet that contains a nodes-visited count. If this number exceeds a defined threshold for the route, or if the packet sticks in the network beyond its maximum defined lifetime, it will be discarded.

DOD ARPANET

ARPANET uses distributed adaptive datagram routing. It pioneered this approach in the late 1960s, and has stuck with it, one of the few large-scale networks that uses this approach. ARPANET nodes send routing tables and statistics to each other periodically. For manageability, ARPANET is broken up into a distinct communication subnetwork composed of IMPs (Interface Message Processors) and their interconnections, which perform the routing and lower-level functions, plus the hosts themselves, attached to the IMPs using any of various IMP-Host protocols depending on the bandwidth of the attachment. The IMPs exchange routing information among themselves at very frequent intervals, and are also subject to a certain amount of central control from "headquarters". The hosts have host tables downloaded to them periodically from the network control center.

INTERNETWORK ROUTING

The topic of routing must also include routing between networks. In this case, networks themselves act like nodes -- the packet traverses one network after another until it finds the right one, and the traverses its nodes until it reaches the final destination. For this to work, there must be mechanisms for networks to discover each other, to determine reachability, to route packets among themselves, etc. The same routing considerations apply to interconnected networks as apply to the interconnected nodes within a network. Thus, routing can occur at two levels (intranetwork and internetwork), and therefore the network layer is often divided into two corresponding sublayers.

Two networks are connected by a "gateway", which has a physical interface to each network and an address on each. If we consider a network to be a collection of nodes that know each other's addresses, then a gateway is a special node that knows the addresses of TWO networks, and has routing tables for both networks in its memory.

There are two major internetworking strategies: The first assumes that the networks to be interconnected are all connection-oriented, i.e. that packets arrive at the gateway properly sequenced, and also that the networks are closely matched in services. This is called the "hop-by-hop" approach. The second, which requires an explicit Internetwork Protocol (IP), assumes that the networks are a mix of connectionless and connection-oriented, with differing services.

The difference comes in where the end-to-end reliability functions are performed. In hop-by-hop interconnection, internetwork routing and end-to-end reliability are combined in the network layer, as if each hop were a miniature transport connection. The network layer of all network nodes must support this style of operation, even for local (intranetwork) communication. This can add overhead to local traffic. The end systems, however, need no particular software support.

IP concentrates on dynamic routing (the real business of a network protocol) and leaves the transport layer on the end system to deal with reliability issues. In the IP approach, the internetworking overhead occurs only when required for internetwork traffic. But the end systems must support an IP protocol, so that the network layer is broken into two sublayers: IP on the end systems, and intra-network routing on the network nodes.

Internetworking Examples:

- CCITT X.75
- ARPANET Internet Protocol (RFC791)
- ISO 8648, "Internal Organization of the Network Layer"
- ISO 8473, "Protocol for Providing Connectionless-mode Network Service"
- ANSI X3S3.3 86-118, ISO TC97/SC6/N 4053, ARPA RFC995 (April 1986)

X.75 INTERNETWORK PROTOCOL

CCITT X.75 is a protocol for interconnecting two X.25 networks; it is not a generalized internetworking protocol. X.75 is essentially an enhanced version of X.25 that operates gateway-to-gateway (DTE to DTE, rather than DTE-to-DCE like X.25). X.75 is transparent to the user of an X.25 network. That is, the user is unaware that multiple networks are involved.

X.75 uses the similar call setup and packet procedures to X.25, but allows 56Kb links, multiple links between gateways for redundancy and higher throughput, 7-bit sequence numbers at frame level, and includes additional fields in the call setup packet.

X.75 gateways are like any virtual circuit routing node -- they have to keep a window full of packets for each connection, along with state history, etc, so that a virtual circuit is maintained end to end.

ARPANET IP PROTOCOL (1981)

Because the ARPANET is composed of many interconnected local networks, there is also an Internetwork Protocol (IP). It provides no guaranteed delivery, no flow control, no sequencing. All that is left up to other layers (the ARPANET transport layer, TCP, which provides end-to-end retransmission and sequencing, or to the underlying network). Each datagram is sent with a specified Quality of Service, Time to Live (self-destruct timer), Options (timestamps, security, special routing), and a Header Checksum. There is no error control for data, no acknowledgments (either end-to-end or hop-by-hop), no retransmission at this level. If an error is detected, the datagram is simply discarded.

Internet addressing is via a 32-bit address, divided into an 8-bit field (to designate which network), followed by a 24-bit field (the address within the network). (There are variations on this.) When a packet must cross a network boundary, it goes through an Internet gateway, which implements the Internet Protocol, and also the gateway-to-gateway protocol (GGP) to coordinate routing and other control information, and exterior-gateway-protocol (EGP) to convey net reachability information between neighboring gateways (Are You There?).

IP works transparently on top of communication subnetwork (network layer & below):

```
   +-----------+     +-----------+
A--| network X |--B--| network Y |--C
   +-----------+     +-----------+
```

A thinks it's sending packets to B. When B gets a packet, it passes it up to the next layer, which turns out to be IP rather than Transport. IP sees that the packet is really for C on network Y, so tells network layer of network Y to send it to C.

Thus, IP is transparent to the network. It requires software on user end to be more intelligent -- IP protocol has to be built into hosts and gateways. In X.75, the internetworking is transparent to the user, and is built into the network, but the gateways must be more intelligent in order to maintain a virtual circuit across networks. And IP places the burden on the transport layer to provide error detection and correction -- packet sequencing, etc. But the TCP/IP transport layer does this anyway.

A summary of the contents of the internet header follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                Example Internet Datagram Header
```

IHL: 4 bits	Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.
Type of Service	8 bits (like QOS) precedence, reliability, etc.
Total Length: 16 bits	Total Length is the length of the datagram, measured in octets, including internet header and data, up to 65536. The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram.
Flags: 3 bits	Various Control Flags:

Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing.

Protocol: 8 bits

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed. The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

Source Address: 32 bits

Destination Address: 32 bits

Options variable, various (security, source routing, timestamp)

NETWORK LAYER SUMMARY

The basic difference in network protocols is in the demands placed upon the transport layer. Connection-oriented or virtual circuit protocols allow a simple transport layer (like ISO TP1, which does practically nothing), whereas connectionless protocols demand an error-correcting, resequencing transport layer (like ISO TP4).

There are three major costs incurred in network routing: the storage required in each network node for routing tables and transmit queues; the computation required to calculate paths; and the bandwidth consumed by routing update messages. The routing table size is in linear proportion to the number of network nodes, and calculation and update transmission overhead rises in proportion to the table size.

In fixed routing schemes, the routing calculation is trivial -- simply a table lookup -- and there are no regular update messages. But fixed routing allows some links to become congested while other potentially usable links go unused. Thus, frequent manual intervention by the network managers is required to keep the network "in tune".

In dynamic routing arrangements, the routing calculation can be relatively complicated, involving cost comparisons, loop control, etc., which depend on data that must be frequently exchanged amongst the network nodes. The larger the network becomes, the more of its resources are devoted to its own routing scheme; when the tables become too large, the network becomes overloaded trying to "optimize network performance".

Virtual circuit networks strike a compromise between fixed and dynamic routing networks. The best route for a particular connection is determined dynamically on the assumption that conditions won't change "too much" during the session, but if that assumption should prove false, nothing can be done.

Any strategy will work well for a small, lightly loaded network. However, when networks become large or heavily used, dynamic datagram routing can lead to congestion, and in such cases virtual circuit routing might be a better choice, OR... the network can be segmented into manageable subnetworks with dynamic routing, and an explicit internet protocol can be put on top.

REFERENCES

Text: Network layer material on pp.8-9, 22-23, 52-68 (X.25), 91-104 (SNA, only for the curious, and stronghearted), 156-160 (DECnet), 233-260 (X.25), 350-357 (ISO definition).

Schwartz, M., "Telecommunication Networks", Addison-Wesley (1987), ch.5-6.

Piscatello, D.M., et al., "Internetworking in an OSI Environment", Data Communications, May 1986, pp.118-136.

Weissberger, A.J., et al., "What the New Internetworking Standards Provide", Data Communications, February 1987, pp.141-156.

RFC791

6. OSI LAYER 4 - THE TRANSPORT LAYER

The transport layer resides in the host computer, and provides end-to-end reliable data transfer between the two systems where the communicating applications are running. It is the highest layer that is directly concerned with the movement of data between machines, and it is the lowest layer that always resides on the end-user systems.

```
+-------------+    +-------------+    +-------------+    
| User        |    | User        |    | User        |    Application
| Application |    | Application |    | Application |    Programs
+------+------+    +------+------+    +------+------+
       |                  |                  |
       |                  |                  |
+------+------------------+------------------+------+
|              Session Layer (maybe)                |    Host Computer
+---------------------------------------------------+    Operating System
|           T R A N S P O R T   L A Y E R           |
+-------------------------+-------------------------+
                          |
  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
                          |
+-------------------------+-------------------------+    Communication
|                   Network Layer                   |    Subnetwork
+---------------------------------------------------+
|                  Datalink Layer                   |
+---------------------------------------------------+
|                  Physical Layer                   |
+---------------------------------------------------+
```

The transport layer depends upon the network layer to deliver its packets through the network, and the network layer, in turn, depends on the datalink layer to perform each point-to-point hop, and to detect and possibly also correct transmission errors.

Since more than one user of a computer may wish to use the network at the same time, the transport layer may "multiplex" multiple user sessions onto a single network connection. Conversely, it can also "split" a single session over multiple network connections in order to improve performance.

```
                     users                      user
                    A  B  C                      D
                     \ | /                       |
                      \|/                        |
                       |                        /|\
                       |                       / | \
                    Network                   Network

                  Multiplexing               Splitting
```

The transport layer uses the underlying COMMUNICATION SUBNETWORK to deliver the user's data. It has some knowledge of the network's characteristics, and matches them with the quality of service requested by the user, while shielding the upper layers from any concern with the network itself.

It is the transport layer's responsibility to ensure that data gets from the source to the destination system completely and correctly. To do this, it engages in transport-layer protocol with its peer transport layer on the other system. This means that it takes the service data units (SDUs) from the upper layers, forms them into network-sized chunks (by segmenting or blocking), and adds on its own protocol information so that its peer transport layer can do the required error checking and recovery.

```
                                    +-----+------+
Upper Layer:                        | PCI | SSDU |
(e.g. Session)                      +-----+------+
                                          |
                                          V
                              +-----+------------+       End-to-end
TRANSPORT LAYER:              | PCI |    TSDU    | <----> Transport protocol
(e.g. ISO TP, ARPA TCP)       +-----+------------+
                                      |
                                      V
                        +-----+------------------+
Network Layer:          | PCI |    NSDU          |  (PCI = Routing info)
(e.g. X.25, IP)         +-----+------------------+
                                  |
                                  V
                +------+-------------------------+-----+------+
Datalink Layer: | FLAG |          DSDU           | FCS | FLAG |Point-to-point
(HDLC, etc)     +------+-------------------------+-----+------+
                             |
                             V
Physical Layer:   (transmits the frame)----------------->

(SDU = Service Data Unit)
(PDU = Protocol Data Unit)
(PCI = Protocol Control Information)
```

If the underlying network already delivers a reliable stream of data, as in connection-oriented virtual circuit networks, then the transport layer has little work to do, except opening and closing the connection, transferring data, and perhaps multiplexing.

But in a connectionless packet-switched or datagram network, packets can take different routes. Packets can arrive at the end system out of order, some can be discarded by the network (because of congestion), or duplicated (retransmission at the network level). The network and lower layers have no way of knowing about this. Furthermore, a packet could be damaged during delivery from the network front end to the host system. Only the end system can detect and correct such errors, and this is the function of the transport layer. Therefore the transport protocol must be prepared to:

```
(a) discard corrupted packets,                          -  Error detection
(b) resequence misordered packets,                      )
(c) retransmit unacknowledged packets,                  )  Sequencing
(d) discard duplicated packets,                         )
(e) engage in end-to-end flow control with its peer.    -  Flow Control
```

A transport layer that can do all this can deliver a reliable stream of packets to its upper layers.

From all this, it should be evident that the piece of protocol information most important to the transport layer is its packet SEQUENCE NUMBER. And when packets have sequence numbers, this means the associated protocol is CONNECTION-ORIENTED, as most transport protocols are.

But SEQUENCING poses special problems at the transport layer. Recall that in the most unreliable type of underlying network, packets may rattle around for unpredictable amounts of time and arrive out of order, not at all, or in multiple copies.

Ambiguity of Packet Sequence Numbers

Sequence numbers are always of fixed length, and "wrap around" after the maximum value is reached. For instance, 3-bit sequence numbers proceed from 0 to 7, and then start over at 0 again. Care must be taken in selecting the actual range of sequence numbers at the transport level.

Let's assume we have a transport protocol with 3-bit sequence numbers and a window size of 7, and that allows "bulk ACKs" (i.e. an ACK for packet n also ACKs all previous packets). (This implies a go-back-to-n retransmission strategy.) Host A sends packets 0, 1, 2, and 3 to B, but packet 0 is delayed in the network. B receives 1, 2 and 3, but does not ACK because it hasn't seen 0 yet. A times out waiting for the ACK and retransmits 0. B then ACKs packet 3 (and implicitly therefore also 0, 1, and 2). Then A sends 4, 5, 6, and 7, and B ACKs these by ACKing 7. Then A wraps around and sends its next packet with sequence number 0. However, the old, delayed packet 0 arrives at B at this point; B can't tell the difference, so the old packet 0 is accepted as the new one.

```
    (packet 0 takes a long, roundabout trip through the network...)
          /-----------+
 A       /            |
  0 ----/             |
  1 ------->          |
  2 ------->          |
  3 ------->          |
                      |
(A times out waiting  |
 for ACK...)          |
                      |
  0 ------->          |
  1 ------->          |
  2 ------->          |
  3 ------->          |
                      |
    <-------ACK(3)    |
                      |
  4 ------->          |
  5 ------->          |
  6 ------->          |
  7 ------->          |
                      |
    <-------ACK(7)    |
                     /
                    /
        Old packet 0 arrives just when B was expecting the new one.

  0 ------->  (new packet 0 arrives but is rejected as a duplicate)
```

So the application gets bad data, accepted as though it were good.

How can the problem of recycled sequence numbers be avoided? (Ask the class)

The most common method is to ensure that the sequence number space is large enough not to wrap around in less than the maximum life expectency of a transport-level packet in the network. Transport protocols tend to have a much larger range of sequence numbers than lower levels use. For instance, typical datalink and network packets have 3-bit or 7-bit sequence numbers, but transport protocols may use 16-bit or 32-bit sequence numbers.

Persistence of Transport Packets Across Connections

But the fact that one transport connection can be closed and another opened soon thereafter poses another problem: how can the new transport connection identify and ignore duplicate packets left over from the old one?

For instance, suppose the first connection is opened (packet 0), one data packet is sent (packet 1), and then closed (packet 2). Howevever, since a confirmation for the data packet was not received within the timeout interval, it was retransmitted, and then confirmed. Now, a second connection is opened (packet 0). At this point, the duplicate data packet (which was held up in the network) finally arrives, and because it has the right sequence number (1), it is accepted, erroneously.

Several methods can be used to avoid the leftover-packet problem:

1. Don't request a new connection until enough time has passed to ensure that no old packets can be left floating around in the network. The drawback here is the delay that is imposed.

2. Don't reset sequence numbers between transport connections. Assuming the sequence number range is big enough, this will work, provided the system doesn't crash and forget its previous transport sequence number.

3. Choose an initial sequence number that is guaranteed to be higher than any leftovers, even if all memory of previous connections has been wiped out. This can be done by basing the number on a time-of-day clock. This method can fail if the system's date and time have been set incorrectly. Some networks may provide a network-wide "global" clock for this purpose (among others).

If sequence numbers are not to be recycled for each transport connection, they must be negotiated during connection establishment. Each side must acknowledge the other side's proposed initial sequence number before data can be exchanged.

TRANSPORT LAYER EXAMPLES

There are two major "public" transport protocols: ARPANET TCP and ISO TP, plus the equivalent layers in proprietary networks like SNA and DNA, which we won't discuss. ISO TP also has a US variant promulgated by the NBS and FIPS. But first let's look at a simple asynchronous point-to-point protocol.

THE KERMIT TRANSPORT LAYER

Kermit's upper layers call upon the transport layer to provide an ordered, and complete sequence of data. The transport layer, in turn, uses the services of the datalink layer to assure that packets are delivered without error. In the Kermit protocol, there is no network layer between the transport and datalink layers, because the PHYSICAL connection is always strictly point-to-point.

Kermit's TRANSPORT LAYER has two major functions: sequencing and error recovery. Kermit's DATALINK layer finds the beginning and end of an incoming packet based on the MARK field, finds the end (and the block check) based on the LEN field, and then checks the block check.

```
+------+--------+-----+------+---------------+-------+
| MARK | LENGTH | SEQ | TYPE | DATA ......... | CHECK |
+------+--------+-----+------+---------------+-------+
```

If a packet arrives in damaged condition, the datalink layer reports a special code "Q" to the transport layer. If a packet does not arrive within the timeout interval, a code of "T" is reported. If it arrives undamaged, then its actual type is reported. Thus Kermit's datalink layer provides error detection and notification, but not error recovery.

Kermit sequence numbers range from 0 to 63, and then recycle (this range is the highest power of two representable in a single printable ASCII character). A Kermit session always begins with packet number 0. The transport layer examines the sequence number and packet type returned by the datalink layer. If a T or Q code was returned, then the transport layer simply retransmits its most recent packet. Otherwise, the transport layer examines the sequence number of the new packet. If it is the expected sequence number, the packet is accepted, the packet number incremented (modulo 64), and the next one is transmitted. Otherwise, the most recent packet is retransmitted. Retransmission occurs up to a maximum retry threshold; if the desired packet cannot be obtained within the retry limit, the transport layer signals failure.

Here is a simple programming example, not in any real language...

```
call datalink(read,type,seq,inbuf)       Read a packet.
for try = 1 to retry-limit {             Try this many times.
  if seq = n and type != 'Q'             If expected seq # & not damaged,
     then return(type)                     return its type.
  call datalink(write,outbuf)            Otherwise retransmit last packet.
  call datalink(read,type,seq,inbuf)     And try again to read.
}
return('T')                              Failed, say we timed out.
```

While Kermit's default transport protocol is stop-and-wait, there is also optional sliding window operation with selective retransmission (not widely implemented). When the window size is greater than one, Kermit must maintain two arrays (windows), one for arriving packets, one for transmitted ones. Each array contains the packets themselves, and the outbound array also stores timing and retry information for each packet.

When a packet arrives, Kermit must check whether the sequence number falls within the current window, and it must generate NAKs for the "most desired packet" (the one most likely to block advancement of the window).

```
     Packet Retries  ACK'd?
     ------ -------  ------
       7       1       n   <-- most desired packet
       8       2       y
       9       1       y
      10       1       y
      11       3       n
      12       1       y
      13       1       y
```

Because Kermit is a simple, point-to-point protocol with no underlying network routing considerations, there are no complications about leftover network packets or packet-number ambiguities.

Kermit's method of closing the transport connection, however, provides an interesting lesson: A disconnect request ("B") is sent and ACK'd:

```
X        Y

DR------>
<------ACK
```

What if the ACK is lost? X retransmits the DR, but Y has already shut down and isn't listening any more.

```
THE ARPANET TRANSPORT LAYER
```

ARPANET Transmission Control Protocol (TCP), defined in RFC793 (1981), provides a reliable host-to-host data delivery protocol for packet-switched networks. It is the standard currently in use by the Dept of Defense, and was developed under DoD sponsored research. It is also widespread on university campuses and in commercial LANs. It is older than ISO TP, and influenced its design. TCP will gradually be replaced by ISO TP.

TCP is designed to run over networks that can lose, damage, misorder, or misdeliver packets, and therefore assumes all responsibility for error detection and recovery and for sequencing. (Recall that IP is a connectionless non-error-checking datagram protocol.) It is considerably more complicated than Kermit's simple transport mechanism.

```
    +---------------------+
    |     Application     |
    | (TELNET, FTP, etc)  |   Upper layers
    +---------+-----------+
              |
              |          Application Program
- - - - - - - - - - - - - - - - - - - - - - - -
              |          Operating System
              |
    +---------+-----------+
    |        TCP          |   Transport layer
    +---------+-----------+
              |
    +---------+-----------+
    |        IP           |   Network upper sublayer (Internet Protocol)
    +---------+-----------+
              |
Host          |
- - - - - - - - - - - - - - - - - - - - - - - -
IMPs          |
              |
    +---------+-----------+
    |     Subnetwork      |   Network lower sublayer (routing),
    +---------------------+   datalink & physical layers.
```

Thus, TCP has fewer layers than the OSI model. Application programs generally call upon TCP directly.

TCP converts between network packets and application data. An application feeds TCP a stream of data bytes, and TCP breaks the stream up into packets of a size appropriate to the network, and decodes incoming packets into an identical data stream, transparently to the application.

A TCP header looks like this. Note that there's no length field. That's because the length is specified in the enclosing IP packet, and IP tells TCP what the length is.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

There is also a connectionless equivalent, called UDP - User Datagram Protocol, which has headers like this:

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+----------------
```

As you can see, UDP provides no sequencing information, and no acknowledgements. Network software that uses UDP must therefore handle the reliability issues normally handled at the transport layer.

UDP is used with Inquiry Response Protocol (IRP) and Name Server Protocol (NSP).

Now let's look in some detail at what TCP must do in order to provide reliable end-to-end data delivery between pairs of processes on top of a less reliable internet communication system.

(1) Multiplexing -- connecting pairs of processes:

To allow for many processes within a single host to use TCP communication facilities simultaneously, TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a "socket". A pair of sockets uniquely identifies each connection: Process A on host X is connected to process D on host Y.

```
 Host X
   user A
          \                                           Host Y
   user B ----(X.A)---(X.C)---(X.B)---->            user D
          /                                       /
   user C           <---(Y.E)---(Y.F)---(Y.D)---- user E
                                                  \
                                                    user F
```

Some sockets are reserved for special applications, like Telnet or FTP servers, which are always "listening" for a connection.

(2) TCP Connection Establishment:

Because it must provide a reliable stream of data, and perform end-to-end flow control, TCP is a CONNECTION-ORIENTED protocol. This means that TCPs must initialize and maintain certain status information. The combination of this information, including:

- sockets
- sequence numbers
- window sizes

is called a CONNECTION. The maximum "segment size" is specified during initial connection, as is the initial sequence number (TCP packets are called segments).

A TCP connection can be used over and over again. New instances of a connection are referred to as incarnations of the connection. How does TCP identify and reject duplicate segments from previous incarnations of the connection?

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number is chosen by the data-sending TCP based on a 32-bit clock that cycles every 4.6 hours (longer than any packet can "live" in the network), and the initial receive sequence number is learned during the connection establishing procedure. Thus, sequence numbers don't "start over at zero" with a new connection.

But if a packet won't be accepted if it doesn't have the right sequence number, and the sequence number is not known in advance, then how can a connection be established in the first place?

The two TCPs must tell each other their initial sequence numbers. This is done by setting a bit called "SYN" (for synchronize), which tells the receiver that the segment's sequence number should be accepted as the first one in the connection. The process requires each side to send its own initial sequence number and to receive a confirmation of it in an acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment:

```
1) A --> B  SYN my sequence number is X
2) A <-- B  ACK your sequence number is X
3) A <-- B  SYN my sequence number is Y
4) A --> B  ACK your sequence number is Y
```

Because steps 2 and 3 can be combined in a single message this is called a "three-way handshake". ("Hello, my name is X." "Hello X, my name is Y." "Hello Y.")

(3) TCP Basic Data Transfer:

TCP is able to transfer a continuous stream of OCTETS (8-bit bytes) in each direction between its users by packaging some number of octets into segments (packets) for transmission through the internet system (network).

In general, the TCPs decide when to block and forward data at their own convenience. Applications, however, can request an explicit "push" of data out of the host and into the network. For instance, a batch-mode application like file transfer might allow TCP to block large amounts of data, whereas a highly interactive application like full-duplex (remote echo) virtual terminal service might request that each character be "pushed", so that its echo will return in a reasonable amount of time. A half-duplex local-echo terminal connection might "push" whenever the user types a carriage return.

In order to detect when data is lost, duplicated, or delivered out of order, TCP assigns a 32-bit SEQUENCE NUMBER to EACH OCTET (yes, octet) transmitted, and requires a positive acknowledgment (ACK) from the receiving TCP (the sequence number of the packet is the sequence number of its first byte). The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received, which means that "go-back-to-n" retransmission is used.

There are no explicit "negative acknowledgements". Rather, when the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

DAMAGE -- which can occur when the packet is transferred between the network front end and the host -- is handled by adding a 16-bit 1's-complement CHECKSUM to each segment transmitted, checking it at the receiver, and discarding damaged segments, and letting timeouts handle the retransmission.

(4) Flow Control and Sequencing

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission. This flow control mechanism explains the use of octet numbers rather than packet sequence numbers. This is an example of the "explicit buffer credit" flow control mechanism, which differs from windows by being able to vary dynamically.

(refer to TCP header picture -- "window" field tells the maximum size the next packet can be)

Buffer credits can lead to problems, however. In some TCP implementations, hosts will grant credit for very small numbers of bytes (like 2). Clearly, there should be some sensible minimum credit, like 100 or 1000, otherwise the connection will be swamped because of the high per-packet overhead (ratio of control fields to data, ACKs, etc).

TCP SOFTWARE CONSIDERATIONS:

On a multiuser timesharing system, the multiplexing function means that the transport entity has access to different users' data. For this reason, the transport function must occur in a secure place, such as in the operating system kernel. If it were done in a user programs, e.g. by linking in a library, then malicious users could easily spy on or masquerade as other users.

Putting TCP in the OS has another advantage too, namely that it can appear to programmers as part of the OS's normal file service. TCP connections can be opened, closed, read from, and written to, just like disk files, communication lines, or other serial devices.

THE OSI TRANSPORT LAYER

The OSI Transport Layer, defined in ISO DP 8072 and 8073 (1984), and the equivalent CCITT X.214 and X.224, has a flexible design to accommodate the variety of network and datalink layers that can lie underneath it. Thus it is even more complicated than TCP.

For instance, TCP always assumes that the underlying network service (IP) is unreliable. But ISO allows for several grades underlying network service:

Type A: Good service. Totally reliable network service, a connection-oriented virtual-circuit network that does not reset (any examples? Teletex?).

Type B: Fair service. Network layer reports but does not recover from failure ("acceptable error rate but unacceptable rate of signalled failures"). This means that if a packet is delivered, it's good, otherwise you'll be notified of errors.

Type C: Poor service. Like IP -- misordered, missing, misdelivered, or damaged messages may be delivered, with no error notification.

To allow reliable end-to-end communication over a variety of network services, 5 classes of OSI transport service are defined, TP-0 through TP-4.

Class	Name	For use with:
0	Simple	Type A Networks (Teletex)
1	Basic Error Recovery	Type B Networks (X.25)
2	Multiplexing, no Error Recovery	Type A Networks
3	Multiplexing and Error Recovery	Type B Networks
4	Multiplexing, Error Detection & Recovery	Type C Networks (datagram)

All five classes do connection establishment, release, and data transfer, segmenting, and error reporting. Class 0 does nothing else. Here are SOME of the other functions of each class:

Function: Class:	0	1	2	3	4
Error Release	Y	N	Y	N	N
Blocking	N	Y	Y	Y	Y
Expidited data transfer	N	Y	Y	Y	Y
Data TPDU sequencing	N	Y	Y	Y	Y
Explicit flow control	N	N	Y	Y	Y
Multiplexing	N	N	Y	Y	Y
Resynchronization (after reset)	N	Y	N	Y	Y
Error Recovery	N	Y	N	Y	Y
Error Detection (Checksums)	N	N	N	N	Y
Retransmission on Timeout	N	N	N	N	Y
Resequencing	N	N	N	N	Y
Splitting	N	N	N	N	Y

Notice that no class is strictly a superset of any other class.

You might think that since X.25 is a connection-oriented network, you could use TP-0 on top of it. TP-1 is required because because it does sequencing, which is necessary when the

X.25 network connection is reset (sequence numbers go back to 0).

How does all this relate to users? Do users get to pick which transport class they want? Does one cost less than another? Probably the user has no real say in the matter. The transport layer automatically adapts itself to the underlying network, of which it has knowledge, and from knowledge of which it shields the upper layers, and the user.

. ISO TP Connection Establishment and Release

The ISO standards provide connection-oriented service only: transport connections must be explicitly established and released. However, there are addenda (ISO 8072/DADI and 8602) that describe a connectionless transport protocol; in this case, the issues of sequence assurrance, etc, are pushed up to the next higher layer. ***

Transport "primitives" (commands) are Request, Indication, Response, and Confirm. The transport layer "requests" a particular service by sending a message to its peer layer on the other system, which gets an "indication" and then "responds" to it. This response arrives at the requestor as a "confirmation". (This same discussion applies to all layers except the lowest.) For instance, to establish a connection, a connect-request is transmitted, which is received as a connect-indication, then out goes a connect-response, which is received as a connect-confirmation.

A connect-request (CR) specifies parameters like:

- Protocol class (TP-4, 3, ..., 0)
- TPDU maximum size
- Normal or extended TPDU format (7-bit or 32-bit sequence numbers)
- Checksum selection
- Quality of service parameters, including throughput, transit delay, priority, and reliability (residual error rate)
- Explicit flow control
- Support for expedited TPDUs

The QOS parameters are then passed down to the network layer.

The initial sequence number is chosen to avoid packets from old connections arriving during a new connection, as in the TCP case. In TP-4 the sequence number plus the incarnation number uniquely identifies a TPDU.

. ISO TP Data Transfer:

Data may be transmitted in normal (DT) or expedited (ED) data units. Sequence numbers are 7 or 31 bits, depending on what was negotiated. They refer to WHOLE TPDUs, not octets within the TPDU. Here’s a normal 7-bit sequence number TPDU. The 31-bit sequence-number version has the checksum (if any) starting in 3 bytes later, followed by the data.

```
   1        2    3          5          6          8
 +--------+----+-----------+----------+----------+----------
 | LENGTH | DT | T-ADDRESS | SEQUENCE | CHECKSUM | DATA ...
 +--------+----+-----------+----------+----------+----------
```

When the transport layer segments its service data units, it indicates the final TPDU in the sequence by setting the "spare" bit in the sequence number (try saying this in English!).

DTs are explicitly ACK'd by the reciever. The ACK carries the sequence number of the NEXT expected TPDU. Multiple TPDUs can be ACK'd cumulatively, which implies go-back-to-n retransmission in case of errors.

The ACK also contains a credit field for window management. The credit field tells how many TPDUs it is ready to receive. Credit + NEXT tells the packet number of the first packet that cannot be transmitted in the current window.

(SHOW THIS... SEE NEXT PAGE)

The NEXT number specifies the sender's lower window edge, and NEXT + CREDIT the upper. Initial credits are exchanged at CR and CC time.

```
  1        2           3           5        6
+--------+-----------+-----------+--------+----------------
| LENGTH | AK CREDIT | T-ADDRESS | NEXT # | parameters...
+--------+-----------+-----------+--------+----------------
```

Why must flow control occur at the transport level if it is already done by the lower levels? Because... this is END-TO-END flow control. The receiving transport entity might well be in a different computer from the network entity, and this computer might be slower, or have smaller buffers. The transport layer must be able to prevent its buffers from filling up.

The TP-4 error detection method is a "Fletcher Checksum", a 16-bit arithmetic checksum applied to the entire TPDU, used if negotiated at connection establishment time. (Why not a CRC? Because TP-4 is more likely to be running on a host, and a checksum is easier to program in software.) Here is the Fletcher checksum algorithm, coded in BASIC. Assume that M$ is the message to be checksummed, including two (initially zero) bytes of checksum, and that L is its length, and N is the position of the first checksum byte (this might be 1, or L - 1).

```
100  M$ = "This is a test message" + CHR$(0) + CHR$(0)
110  L = LEN(M$) ' Length of message, including checksym bytes
120  N = L - 1   ' Position of checksum in message

1000 C0 = 0
1010 C1 = 0
1020 FOR I = 1 TO L
1030   C0 = (C0 + ASC(MID$(M$,I,1))) AND 255
1040   C1 = (C1 + C0) AND 255
1050 NEXT I
1060 X = (-C1 + (L - N) * C0) AND 255
1070 Y = (C1 - (L - N + 1) * C0) AND 255
1080 MID$(M$,N,1) = CHR$(X)
1090 MID$(M$,N+1,1) = CHR$(Y)
```

The result is that the modulo-256 sum of all the data bytes including the two checksum bytes should be zero. Thus, to check a received checksum, do:

```
1100 C0 = 0
1110 C1 = 0
1120 FOR I = 1 TO L
1130   C0 = (C0 + ASC(MID$(M$,I,1))) AND 255
1140   C1 = (C1 + C0) AND 255
1150 NEXT I
1160 IF C0 = 0 AND C1 = 0 THEN PRINT "IS OK" ELSE PRINT "IS NO GOOD"
```

This method (the only one defined for OSI Transport error detection) is efficient, but will not detect insertion or loss of leading or trailing 0 bytes, nor misordering of certain octets.

If corruption is detected, the packet will be discarded. If it does not arrive within the specified interval, a TIMEOUT and retransmission will occur. TP-4 has many timers; one for unack'd CRs, another for unack'd DT's, another for unACK'd ED's, etc. -- 12 of them in all. Setting of timers at their best levels (so they don't go off prematurely, but still catch missing packets promptly) is a tough job.

COMPARISON

Kermit vs TCP vs TP4

Connection establishment:

Kermit: Connection always starts with packet 0. Since connection is a single circuit, packets don't take alternate paths, can't pass each other, so no leftover packet problem.

TCP: 3-way handshake on initial packet numbers, based on TOD clock. Active (client) and passive (server) transport entities.

TP4: 3-way handshake on initial packet number.

Data transfer:

Kermit: Performs error recovery but not detection (which is done by datalink layer, which informs transport layer), sequencing and resequencing. Packets are numbered 0-63. Flow control by stop-&-wait or sliding window with selective retransmission.

TCP: Performs error detection AND recovery, sequencing and resequencing. Each byte has a sequence number, 0 - 2^31-1. Flow control by credit allocation (per byte). Go-back-to-n retransmission.

TP4: Like TCP, but packets are numbered (7-bit or 32-bit sequence numbers), rather than bytes. Flow control by credit allocation (per packet). Go-back-to-n retransmission.

Connection release:

Kermit: 2-way handshake.

TCP & TP4: 3-way handshake.

SUBSTITUTIONS

A highly touted benefit of layering is that equivalent layers can be substituted for one another. Substitutions often occur at or near the transport layer.

Is it possible to run TCP over, say, X.25? Not directly (TCP and IP are too closely intertwined). But TCP/IP can indeed run over X.25. In one scenario, an X.25 subnetwork can be used unmodified. In this case, the host TCP/IP must be modified to know how to place the X.25 call, and close it when finished. In the other scenario, the subnetwork can be modifed to interface to X.25 as well as to its own network; this allows more efficient operation, and is transparent to the host software.

A common substitution is IBM's IEEE 802.5 Token Ring interface for the lower layers usually used by TCP/IP (such as Ethernet, Host-IMP protocol). TCP packets are embedded in IP packets. The IP layer calls directly upon the LLC sublayer of the Token Ring datalink code.

IBM PC Kermit (in its most recent release) is able to call upon NetBIOS to send its packets to other PCs on a PC network. In this case NetBIOS, a session-level protocol, fills in for Kermit's physical layer, so that the datalink and transport functions are duplicated by the two protocols.

Conversely, IBM PC-DOS software that uses NetBIOS session-level calls can sit on top of TCP/IP, perhaps running on Ethernet. This could allow PC networks, print servers, shared dBASE's, etc, to coexist on a wide area Internet.

TRANSPORT LAYER SUMMARY

The transport layer must ensure reliable, correct, and complete delivery of data from one end system to another, regardless of the characteristics of the underlying network. UNLESS... a higher layer elects to do this, on top of a "connectionless" transport service like UDP or TP0.

If more than one user of an end system can have a transport connection at one time, the transport layer must provide a SECURE MULTIPLEXING function so that multiple users can have access to the same network connection, and data is delivered to its intended users. Multiplexing introduces its own set of problems, however: how can the transport layer distinguish between highly interactive sessions (like virtual terminal connections) and batch-oriented ones (like bulk file transfer)?

The transport layer must provide END-TO-END FLOW CONTROL. This is normally done using a sliding window mechanism, controlled either by acknowledgements or by explicit granting of buffer credits.

If the underlying network is connection-oriented and totally reliable, then the transport layer need do little beyond multiplexing and flow control. Otherwise, it must also perform sequence and error control, and even address verification.

SEQUENCE CONTROL is accomplished via the transport packet sequence numbers. Some transport protocols assign sequential numbers to each packet (0, 1, 2, etc), others to each byte within the packet (ARPANET TCP). The sequence number allows duplicate packets within a connection to be discarded, missing ones to be detected (and retransmission requested), and out-of-order packets to be correctly sorted. Special problems occur when duplicate packets from an old connection arrive at a new connection.

If the underlying network service does not provide error-checked packets, then ERROR CONTROL is accomplished at the transport level using a checksum. Corrupted packets are normally discarded. The sender eventually times out awaiting an acknowledgment and retransmits.

REFERENCES

Textbook, Transport Layer material on pp.23-24, 69-80 (skim), 160-165 (DECnet), 105-112 (SNA), 345-349 (ISO Definition).

Kermit book, pp.214-220.

Schwartz, M., "Telecommunication Networks", Addison-Wesley (1987), ch.7.

Stallings, W., "A Primer: Understanding Transport Protocols", Data Communications, Nov 1984, pp.201-215.

Postel, J., Editor, "Transmission Control Protocol", RFC793 (1981).

Postel, J., Editor, "User Datagram Protocol", RFC768 (1980).

CCITT Recommendation X.214, "Transport Service Definition for OSI for CCITT Applications" (1984).

CCITT Recommendation X.224, "Transport Protocol Specification for OSI for CCITT Applications" (1984).

7. THE SESSION LAYER

If the transport layer (at least when it's connection-oriented, as most are) provides reliable streams of data between end systems, then what do we need additional higher layers for? Why doesn't the application program itself simply read and write data directly to and from the transport layer?

(example of two programs, open, read/write, close...)

Well, in the ARPANET that's exactly what happens. TCP is implemented within the file system of the host computer, and application programs open, read from, write to, and close transport connections exactly as they would files. This imparts simplicity to the network design, but unloads many tasks onto the application programmer -- tasks like synchronization of dialog, character set conversion, data format conversion, etc -- that might more properly and consistently be done elsewhere, and requires that each application program duplicate these functions (hopefully, in a compatible way!)

For this reason, TCP/IP networks do not abound in applications, generally providing only three major ones: virtual terminal service (TELNET), file transfer (FTP), and electronic mail (SMTP). Each of these applications is large and complex.

The OSI approach says that it makes more sense to put these common tasks in a common place. Rather than require these functions to be coded into each application program, they are collected into the network software, implemented in the operating system or in program libraries to be linked with the application program.

These high-level functions fall into three categories: session control (dialog management), presentation services (data syntax conversion), and application services (the actual processing of the data).

OSI LAYER 5 - THE SESSION LAYER

There are several competing session-layer standards, including ones from CCITT (X.215, same as ISO 8326), and ECMA, each with different concerns (e.g. ECMA is vendor-oriented, whereas CCITT is carrier-oriented), not to mention the equivalent functions in proprietary architectures like SNA and DNA. The Arpanet protocols do not bother with a session layer at all.

The ISO has tried reconcile the various viewpoints, but the result may be so complex that nobody will accept it, especially in view of the fact that many applications have been written directly over the transport layer. The following discussion applies to the CCITT version.

The session layer resides in the host computer, and provides structured, reliable communication between two cooperating processes on the end systems. It is the lowest layer that is unconcerned with the movement of data between machines. It interacts with its peer session layer as if there were no network between them at all, in fact as if they were two processes on the same machine. The session layer depends upon the transport layer to deliver information to its peer layer on the destination end system. There is a one-to-one correspondence between transport connections and session connections; the session layer does not multiplex.

This means it is the responsibility of the transport layer to deliver incoming data to the correct session. Since the transport layer can "see" all of the sessions, it is necessarily "privileged" and therefore must be protected from direct access by users. For this reason (and others) it usually resides in the operating system.

Since the session layer sees only its own user's data, it may reside in application program, though it may also be part of the operating system, or linked with the user program from a runtime library.

```
+-------------+   +-------------+   +-------------+   Application
| User        |   | User        |   | User        |   Programs
| Application |   | Application |   | Application |
+------+------+   +------+------+   +------+------+
       |                 |                 |
+------+------+   +------+------+   +------+------+
|   Session   |   |   Session   |   |   Session   |
|    Layer    |   |    Layer    |   |    Layer    |
+------+------+   +------+------+   +------+------+
       |                 |                 |
+------+-----------------+-----------------+------+   Operating System
|                Transport Layer                  |
+------------------------+------------------------+   Host Computer
                         |
 -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
                         |
+------------------------+------------------------+   Communication
|                  Network Layer                  |   Subnetwork
+-------------------------------------------------+
|                 Datalink Layer                  |
+-------------------------------------------------+
|                 Physical Layer                  |
+-------------------------------------------------+
```

But what does the session layer do? There are some aspects of interaction between two processes that are common to all connections, and which the session layer can take responsibility for. These include dialog management and checkpointing.

. Dialog Management:

The transport layer sends and receives data in a full duplex manner, without any regard for the structure or content of the data. But applications are not concerned with arbitrarily chopped-up pieces of messages arriving at unpredictable times; they expect whole commands, or whole responses. Furthermore, many computer applications that follow the command-response pattern of interaction are not prepared to accept new commands until they have finished "outputting" their response to the last one.

The session layer allows the applications to specify whether the dialog is one-way (OW, simplex), two-way-alternate (TWA, half-duplex), or two-way-simultaneous (TWS, full-duplex), and then enforces the selected style of dialog. In the TWS case, each side sends whenever it likes, and relies on the partner's buffering capability. This is analogous to typeahead on a full-duplex terminal connection.

The TWA case is more complicated. It is an attempt to impose a half-duplex style of interaction on an intrinsically full-duplex medium, mainly to allow for the IBM style of dialog. There are those who say "why bother?" -- if side A sends data before side B has asked for it, then let side B's transport layer hold it in a buffer temporarily.

Since only one side can transmit at a time, there must be a mechanism for granting permission to send. This is called the "token". When a session connection is established in which TWA dialog has been negotiated, one side gets to be "first". That side transmits and then, when done, sends the token to the other. This is analogous to a half-duplex terminal connection at the physical level, in which each side gives the other permission to transmit using a "line-turnaround handshake" character (like CR or Ctrl-Q).

In a TWA session there must also be a way to send data out of turn, for instance to interrupt an application that is producing unwanted data (maybe because it was given the wrong command). The session layer provides this mechanism as "typed data" (data sent without permission). An example might be the BREAK signal sent by a half-duplex terminal -- "please give me the token".

Another facet of dialog management is called "quarantine". This is useful in applications where a batch of commands must be delivered all at once, or not at all. For example, when updating a remote database to transfer money from one account to another, you would want the debit and credit commands to be executed together so as not to leave a "window of vulnerability" during which the money was in two accounts at the same time. Quarantine service allows commands to be bracketed, and then delivered all at once to the application, or not at all. It may be buffered locally by the sending session, or remotely by the receiving one. It is not delivered to the application until the closing quarantine bracket ("release") is encountered.

. Checkpointing:

The session layer provides mechanisms for marking the beginning and end of an "activity", and for dialog units within an activity, and even for records within a dialog unit. There are "synchronization" commands that can be used to isolate these pieces of data from one another. In the event of failure or termination of a session, the activity can be resumed at a previously marked synchronization point using the resynchronization service.

The ISO Session Layer defines major and minor sync points. Major sync points separate "dialog units", in which all data is separated from all data in other dialog units. After defining a major sync point, a user may not transmit additional data until the sync is acknowledged. Once a dialog unit is acknowledged, all recovery information for it can be purged. For instance, if multiple files are being transmitted, each file can be a separate dialog unit. Recovery is only possible back to the last major sync point.

Minor sync points are similar, but need not be ack'd before further transmissions occur (acks to these must be explicitly requested). If synchronization is lost, it should be possible to back up to ANY minor sync in the dialog. The more minor sync points, the more efficient backup and recovery (at the expense of transmitting and saving frequent checkpoints). Sync points are marked with serial numbers. The session layer does not save the data itself; this is the responsibility of the USER of the session layer, which must transmit it again.

An "activity" is a logical unit of work, consisting of one or more dialog units, that can be interrupted and later resumed, e.g. if one of the systems goes down. The session layer automatically stores the serial number of the last sync point, so that the session user can resume from that point.

A session may consist of one or more activities in sequence, or one activity can span several sessions (as when it is interrupted and later resumed in a new session).

The checkpointing functions each have associated tokens: Set Major/Minor Sync Point, Start/Resume/Interrupt/Discard/End Activity. The use of these tokens is negotiated at the beginning of the session connection, and the tokens are exchanged using the "give token" and "please token" functions. The use of tokens is optional, and if their use has not been negotiated, then checkpointing cannot be done.

. Other Functions of the Session Layer:

Transport connection mapping. One-to-one, but... Several sessions can occur in sequence over a single transport connection, and one session can span multiple transport connections. The latter capability provides the ultimate in shielding the user from network failures -- if the network crashes and then comes up again, the application may continue to run as if nothing happened. But this could open a "window of vulnerability" in which an intruder can gain access to someone else's open session.

The ability to support several simultaneous sessions over a single transport connection is not yet defined in OSI. This could relieve the transport layer of the multiplexing function, but would introduce additional complications into the session layer, like flow control. And a multiplexing session layer would have to run in a secure environment, like in the OS, thus it would add size and complexity to the OS.

Quality of Service (QOS) -- Like all the other layers, the Session Layer may request a given quality of service from its inferior layer (Transport). The parameters may be either prearranged or negotiated, and include:

Performance:

- Session connection (SC) establishment delay
- SC establishment failure probability
- Throughput (overall rate)
- Transit delay (round trip message delay)
- Residual error rate (lost, damaged, or duplicated session data units)
- Transfer failure probability (percent of time Transport can fail)
- SC release delay
- SC release failure probability
- SC resilience (probability that Transport will "abort" the connection)

(All of the above are really QOS parameters of the Transport service. A desired as well as minimum acceptable value is specified for error rate, throughput, and transit delay.)

Protection:

- None
- Protect against passive monitoring
- Protect against modification, deletion, replay, insertion of data
- Both kinds of protection

SC priority (must use at least the requested priority):

- The order in which SCs have their QOS downgraded, if necessary
- The order in which SCs are broken to recover resources, if necessary

Extended control:

- Let user use resync, "abort", activity interrupt/discard services when normal flow is congested (desired or not desired)

Optimized dialog transfer:

- Concatenation of multiple session service requests into a single unit (desired or not desired).

The session entity requesting the connection will specify any QOS parameters that are not prearranged. The other session can refuse the connection, or accept it, possibly downgrading the QOS based on its own capabilities.

Since many features are defined for the session layer, it is not expected that any particular session layer will have them all. A full-blown implementation is not only complex but unnecessary for virtually all applications. So, four subsets have been defined, similar to the transport subsets T0-T4.

- The Session Kernel includes connection establishment, release and management, as well as data transfer, in other words transparent use of Transport connections with none of the special features of the Session Layer.
- The Basic Combined Subset (BCS) also includes options for expedited data, token exchange, and typed data, and is intended for use in TWA (half duplex) session connections, e.g. terminal-to-host connections.
- The Basic Synchronized Subset (BSS) adds to this the checkpointing features and negotiated release, for use in reliable file transfer and transaction processing.
- The Basic Activity Subset (BAS) is a specialized session protocol, which adds activity management, in which multiple activities can be operated, suspended, and resumed over a single transport connection, but lacks full duplex operation, minor synchronization, resynchronization, and negotiated release. Used in CCITT applications to message text from control information, similar to CCITT T.62 (Teletex and Group 4 Facsimile).

SERVICE	KERNEL	BCS	BSS	BAS	
Session Connection	X	X	X	X	
Normal Data Transfer	X	X	X	*	(1/2 duplex)
Expedited Data Transfer	-	-	-	-	
Typed Data Transfer	-	-	X	X	
Capability Data Transfer	-	-	-	X	
Give Token	-	X	X	X	
Please Token	-	X	X	X	
Give Control (All Tokens)	-	-	-	X	
Minor Sync Point	-	-	X	X	
Major Sync Point	-	-	X	-	
Resynchronize	-	-	X	-	
Exception Reporting	-	-	-	X	
Activity Management	-	-	-	X	
Orderly Release	X	X	*	X	(optional)
"Abort"	X	X	X	X	

Different applications may require different session layer subsets. For instance MAP requires BCS, whereas FTAM needs BSS, and X.400 must have BAS. If all these applications are to run on the same system, there should be a common session layer that supports all the required functions. Unfortunately, no one subset is a superset of the others.

. Session Protocol:

Like the other layers, session protocol proceeds through connection establishment, data transfer, and connection release phases, and it may do segmenting and blocking where permitted. But unlike all the other layers we've looked at so far, it does NOT do sequencing, flow control, or error checking (why not?).

An interesting sidelight... Data SPDUs may be of any negotiated length, but connect requests, etc, are limited to 512 bytes. This makes it tough for the application layer to piggyback an application connect request inside a Session connect request. A new version (2) of the Session layer standard (Feb 88?) will allow unlimited length data in all SPDUs. But how will an application that's designed to work with V2 know that it's connecting to a V1 session layer on the other end???

A session packet (SPDU) consists of unit identifier (packet type), some parameters, and user data (the SDU of the upper layer). The parameters are encoded using "PLV" (parameter-length-value) notation: a code specifying which parameter (precedence, security level, address, etc), the length of the parameter, and a parameter value of the given length. This means that only those parameters that will actually be used are transmitted.

```
 +------+-----+-----+-----+-----+-----------+
 | TYPE | LEN | PLV | PLV | ... | USER DATA |
 +------+-----+-----+-----+-----+-----------+
```

(Does this remind you of the encoding of a Kermit Generic Command packet?)

Parameters may be grouped, so that a single parameter field can include many related parameters, which can be easily skipped over.

Parameters thus encoded include session addresses, the QOS parameters, type of service (e.g. Teletex), etc.

Question: do we need a session layer? One experienced observer notes that in the original British proposal to ISO, there was no session layer at all. But the IBM SNA protocol has one, and so it was included after all. The idea of session control -- who is allowed to talk, who must listen -- fits with the whole IBM philosophy of communications, a kind of authoritarian view of the world. In this view, the few applications that need dialog control, checkpointing, and quarantine can provide it themselves.

8. THE PRESENTATION LAYER

The presentation layer transfers "logical chunks" of data (like commands and responses) between the application and the session layers, and worries about their syntax (format). But there are many kinds of data -- numbers, characters, strings, etc -- and many different ideas of what each should look like, and there are many competing proposed and de-facto standards.

```
Application:       Presentation Context     Presentation   Transfer Syntax
System-Dependent <-----------------------> Layer         <--------------->
Syntax
```

The basic problem is this: how do you transfer data between two computers that represent data in different ways? For instance, one computer might represent text in ASCII, another in EBCDIC. Two systems might also have different internal formats for storing numeric data.

At first glance, it would seem that if one computer knew the other computer's conventions, it could translate before transmitting. But when you consider that there are many kinds of computers with many more kinds of conventions, then each computer would have to embody the full knowledge of every other kind of computer's data formats, clearly an impractical approach.

The answer is a "common intermediate representation". For each application, a standard format is chosen for interchange between unlike systems. The sending system converts to it, the receiving system converts from it. Thus each system only needs to know two formats for each application: its own, and the common format. Otherwise, if there were N different kinds of systems that had to communicate, each system would have to be programmed to know n different formats for each application.

The presentation layer provides for a set of common intermediate representations for various "low-level" types of data, in order to make the application independent of syntax.

The most widely accepted presentation standard is CCITT X.409, part of the X.400 electronic mail standard. X.409 syntax includes notations for the following datatypes: Boolean (true or false), Integer, Bit string, Octet string, Sequence, Set, Numeric string, Printable string, T.61 string (Teletext), Videotex string, IA5 (International Alphabet 5) string, UTC time, Generalized time. Each of these data types has a specific ID code, like 2 for integer, 21 for Videotex string. A data item is represented by a triplet:

```
+------------------+--------+------+
| Datatype ID code | length | data |
+------------------+--------+------+
```

For example:

```
17  0B  3631303230383132303305A (hex)
23  11  6102081200Z
 |   |  |
 |   |  |Date in Coordinated Universal Time (UTC) format (8 Feb 61 12:00)
 |   |Length
 |Data type code for UTC
```

with the characters in the data portion represented in IA5 (mostly = ASCII). Characters are translated from the native code (ASCII, EBCDIC, etc) into IA5 during transmission.

The "Presentation Context" is the mapping between the Application's abstract syntax notation and the Presentation layer's transfer notation. The Abstract syntax is negotiated between the Applications, whereas the Transfer syntax is negotiated between the presentation layers.

Since the presentation layer is actively manipulating the syntax of the user data, it is also the place chosen for other data transformations, such as ENCRYPTION and COMPRESSION, although the ISO standards have little to say in these areas yet.

Kermit provides a simple illustration of the functions of the presentation layer. Kermit has two presentation contexts -- text and binary. In text mode, the transfer syntax is ASCII, with lines terminated by CRLF. In binary mode, the transfer syntax corresponds to the machine's internal representation. In both contexts, further transformations are also done:

1. Control characters are encoded as printable characters, like #A, always.

2. Characters with 8th bit = 1 are encoded as 7-bit characters, prefixed by &, negotiated.

3. Repeated characters are prefixed by a special flag and a repeat count, ~<n>X, negotiated.

While it may be argued that some of these functions might be assigned to the datalink layer because their intention is to allow the data to get through possibly non-transparent physical links, the transformations are actually done much at a much higher level, above the transport layer.

REFERENCES

Textbook, pp.24-28, sections on SNA (112-138) and DNA (165-172) if you're interested, 333-344 (OSI definition).

Rauch-Hindin, W., "Upper Level OSI Protocols Near Completion", Mini-Micro Systems, July 1986, pp.53-66.

Stallings, W., "Is There an OSI Session Protocol In Your Future?", Data Communications, November 1987, p.147-159.

CCITT Recommendation X.215, "Session Service Definition for OSI for CCITT Applications" (1984).

CCITT Recommendation X.225, "Session Protocol Specification for OSI for CCITT Applications" (1984).

ISO 8326 (CCITT X.215), Connection Oriented Session Service Definition

ISO DP 8822, Presentation Service ISO DP 8823, Presentation Protocol

CCITT X.409, Presentation Transfer Syntax

Parts of Kermit book

9. THE APPLICATION LAYER

Now that we can establish reliable communication between any two machines on a network, how can we use these connections to get real work done? For instance, how can we send mail or transfer files between two machines? When the machines are made by a single manufacturer, like DEC, IBM, or Wang, a proprietary, vendor-specific solution is available at a price -- DECnet, SNA, Wangnet, etc.

When the two machines are of different manufacture, often the only solution is a very expensive proprietary hardware/software packages, like an implementation of IBM SNA on the DEC VAX (or DEC DNA on the IBM mainframe). But then what happens when you want to bring a Hewlett-Packard mini or a CDC supercomputer into the picture? Must each variety of computer have a special connection to each other kind?

Public, open, standard protocols like TCP/IP and its applications, or ISO OSI applications, address this problem by providing a single standard, "open" network architecture, and set of applications and protocols, potentially common to all computers. Each computer must know only its own internal architecture and data representations, and those of the common network standard.

The Application layer is where the real work gets done -- the work of communicating meaningful messages between application processes on different machines. The lower layers "merely" convey data. While it may have seemed that some of the lower layers were quite complex, at least their jobs were bounded -- each has a circumscribed set of tasks to accomplish. The application layer has no such restrictions -- it can be as complex as the task at hand -- the application -- requires.

Typically, the application layer is responsible for (mainly the first two):

- Exchange of "meaningful messages" between two application processes
- Identification of the communicating partners by name, address, description
- Determination of the availability of addresses and necessary resources
- Establishment of authority to communicate
- Authentication of partners' identities.
- Passing user-selected qualify-of-service parameters to the lower layers.

In the TCP/IP world, there is no application "layer", any more than there is a session or presentation layer. Rather, application programs communicate with each other directly through the transport service, which is typically buried in the computer's operating system, using calls to the computer's file system (open, close, read, write). This approach is simple, and it allows applications that were not even written for network operation to use the network as though it were a terminal or a disk file.

In the ISO/OSI world, matters are a bit more complicated. "User elements" are the interface between the application program and the application layer, or "application entity" (AE). In each AE, the application process has to choose a particular application service element (ASE) to perform its task. Sets of user elements are defined for each application (file transfer, RJE, electronic mail, Videotex, etc). Within a particular computer, these would be called ASE1, ASE2, ASE3, etc. Special ASEs have already been defined for X.400 and FTAM. A single application can access one or more ASEs.

And there is a directory service ASE, so that, say, a file transfer client on system A can learn the address of the file transfer server on system B in order to make a connection. This is how applications connect with other in the first place.

The application is known by its presentation address. The Associate Control Service Element (ACSE) provides the mapping between the process and this address.

There is also talk of a Commitment, Concurrency, and Recovery (CCR) ASE. Concurrency refers to management of multiple simultaneous access to the same resource, e.g. a database. Commitment refers to one system assurring the other that the requested work has been done. blah blah... Does this sound familiar? It is the application layer's interface to the session layer's checkpoint and quarantine features, which somehow penetrates the presentation layer. Similarly, Reliable Transfer Service (RTS, not an ASE, exactly, but a "module") lets the application at the session layer's dialog control functions -- "give turn", "please turn", etc.

An important concept at the application layer is the relationship between "client" and "server". In a network, there are typically various kinds of servers ready and waiting for work to do -- directory service, file transfer, mail delivery, etc. They are activated when a user (a person) runs a "client" program, which finds and engages the server.

In the TCP/IP world, servers are accessible through "well-known socket numbers" -- special transport addresses that are publicized. In the ISO world, we have a sublayer of the application layer called Remote Operation Service (ROS), whose purpose is to split a distributed application into "modules" or "agents", which communicate through an asymmetrical "access protocol" (asymmetric because, e.g., only the client has the right to initiate a dialog).

ABSTRACT SYNTAX NOTATION

The application layer is concerned with the "semantics" (meaning) of the data being transferred -- files, commands, screens, mail messages, database queries, etc. In order for applications on different systems to communicate, messages are expressed in "abstract syntax". This is agreed-upon notation for high-level concepts, distinct from the low-level translations done at the presentation layer, which is concerned only with how numbers, characters, and strings are represented during transmission.

The OSI abstract syntax is called ASN.1 (Abstract Syntax Notation One). The Abstract Syntax Notation describes the data structures being exchanged, in machine-independent fashion.

(...find out about ASN.1...)

The MOST COMMON APPLICATIONS are FILE TRANSFER, NETWORK TERMINAL CONNECTION, and ELECTRONIC MAIL.

FILE TRANSFER AND MANAGEMENT

A file transfer application allows files to be moved between systems that possibly differ in how they store, name, and format data into files, and may also provide additional file management features including file deletion, renaming, appending, directory changing and access, etc.

A file transfer protocol must be able to transfer files correctly and completely, clearly marking the beginning and end, and supply the file's name and possibly other attributes to the receiving system, and, for certain types of files (e.g. text), perform any necessary transformations to make the file useful on the receiving system.

An important issue in file transfer is whether the file can be sent to a foreign system and then later retrieved so that the new copy is identical to the original. This property is called "invertibility".

Another issue concerns the transfer of files to systems whose file storage or naming conventions are more restrictive than the originating system's. For instance, if files on system A have 50-character-long names, but system B only allows 6-character names, special care must be taken when sending files from A to B to ensure that names are converted to legal form for B, and that duplicate names are avoided.

There is increasing stress on the appearance of text, and increasing demand to preserve its printed representation, including graphic effects like underlining, boldface, italics, different type sizes, etc. How can you transfer a document among Wordstar, Wordperfect, and MacWrite and have it look the same? Each vendor is developing its own "document description language" for formatted documents -- IBM's DCA (Document Content Architecture) and Document Interchange Architecture (DIA); DEC's DDIF (Digital Document Interchange Format); US Navy's DIF (Document [Data?] Interchange Format); Adobe's PostScript; Interleaf; Imagen's Impress, Xerox's (something) ... These organizations are vying with one another to have their particular design adopted as "the standard" in this area. For now, file transfer protocols simply transfer files in these formats just as if they were any other files, and rely on the users at each end to pre- and post-process them if necessary.

Out of these has arisen some ISO standards efforts:

- Office Document Architecture (ODA), ISO/DIS ..., structures for exchange of processible documents.
- Office Document Interchange Format (ODIF), ISO/DIS ..., ASN.1 encoding that constitutes a particular representation for ODA.

Similarly for databases & spreadsheet data (DIF, DBF, SYLK, WKS)...

Similarly, representation of computer-generated graphics in system-independent form has given rise to a set of competing standards -- GKS, Core graphics, CGM (ANSI), CGI, HP's HPGL, IBM's ADMGDF, Computer Associates DISSPOP, ANSI PHIGS (Programmers Hierarchical Interactive Graphics System), etc -- as well as standards for specific applications like CAD, like IGES. Again, files are stored in these formats and then post-processed after transfer.

In the continuing quest for "interoperability", we come across a significant contradiction. Vendors are intrinsically motivated to develop bizarre and byzantine file systems -- files with all sorts of attributes and peculiarities -- in order to tie their customers to their equipment. (...examples -- FILES-11, RMS, MVS, CMS, Macintosh, ...) These vendors have no incentive to make their file systems conform to some kind of "standard". Yet, if we are to have a truly interoperable, heterogeneous computing environment, there has to be a way of fitting ALL files everywhere into a manageable set of categories.

One approach is to classify all files as either "text" or "binary". That is, as either convertible, or not. A text file is typically restricted to the common graphic characters -- letters, digits, and a few punctuation marks, as defined in US 7-bit ASCII. Such files can be represented on most computers (some exceptions include CDC supercomputers that have only 6-bit character sets). All others are considered "binary", and are transferred as-is, with no attempt at conversion. This is the approach of "basic" Kermit, and of Arpanet FTP.

At the other extreme, we have efforts to represent every conceivable attribute of every manufacturer's file system. This kind of thing gets out of control pretty fast, even within one vendor's circumscribed world -- for example, DEC, in its Data Access Protocol (DAP) supports 42 different "generic system capabilities" (like whether files can be preallocated, appended to, accessed randomly, etc), 8 data types (ASCII, EBCDIC, executable, etc), 4 organizations (sequential, relative, indexed, hashed), 5 record formats (fixed, variable, etc), 8 record attributes (for format control), 14 file allocation attributes (byte size, record size, block size, etc), 28 access options (supersede, update, append, rewind, etc), 26 device characteristics (terminal, directory structured, shared, spooled, etc), various access options (new, old, rename, password, etc), in addition to the better known file attributes like name, creation date, protection code, and so on. All this was deemed necessary even when the designers had only a small number of machines to worry about, all from a single vendor. And with all that, it's still optimistic to expect to transfer anything but ASCII stream files between two DEC machines of different architecture (say VAX/VMS and DECSYSTEM-20). What, then, when we bring IBM and CDC and Apple into the picture???

.. Xmodem:

Xmodem is a bare-bones protocol for transferring the contents of a single file. No conversions are done at the presentation, application, or any other level, and in most variants, the filename is not transmitted separately, or at all. Xmodem might be viewed as a datalink protocol used for file transfer. When Xmodem works at all (as it often does not, due to datalink transparency problems), file transfer is invertible.

An Xmodem connection is initiated "manually" -- the user uses terminal emulation on one computer to connect to another, starts up an Xmodem program on the remote computer, puts in the send or receive mode, "escapes back" to the local computer, and puts it in the opposite (receive or send) mode.

.. Kermit:

Kermit is a simple example of a file transfer protocol, and illustrates some of the basic issues. Like Xmodem, Kermit is a manual operation -- the user must establish the connection and start Kermit programs on both ends. Once started, however, the remote Kermit may be put in server mode (like a network file server) so that the Kermit client can control the Kermit server by means of protocol messages.

The Kermit file transfer protocol identifies the file by name, unambiguously marks its beginning and end, and, unless instructed otherwise, converts the contents of the file to a common intermediate representation during transmission, so that it may be converted and stored in useful form on the target system. Such conversion is normally done on "text files" and is avoided for "binary files".

Text files consist of "records" or "lines", each consisting of a string of characters in a particular character set like ASCII or EBCDIC. Character set conversion is done at the presentation layer, and record format conversion takes place at the application layer. For instance, an IBM mainframe might store text in 80-column EBCDIC "card image" format, and a UNIX system uses an ASCII stream, with records delimited by LF. Kermit's common intermediate representation for text is ASCII with CRLF after each line, so an IBM mainframe sending a file would translate from EBCDIC to ASCII at presentation level, and from fixed-length 80-character blank-padded records to CRLF-terminated lines at application level. The UNIX system would convert from CRLF-terminated lines to LF-terminated lines upon receipt. MS-DOS systems need no conversion, since their text storage format coincides with Kermit's transmission format.

The Kermit protocol defines a wide range of file attributes, such as length, creation date, format, record length, and so forth, which may be transmitted along with the file, in hopes that the target system can preserve them. The attributes are defined in a system-independent way, and thus provide a common intermediate representation for these parameters. To date, few Kermit programs actually make use of this mechanism.

Transfer of binary files is generally invertible (the major difficulty is in EOF indication), and text files also tend to be invertible (ASCII/EBCDIC conversion is the major stumbling block, since EBCDIC-to-ASCII translation itself is not necessarily invertible). Kermit does not address higher-level concerns, such as the appearance of formatted or typeset text, nor does it allow for checkpointing.

The Kermit protocol defines mechanisms for file management as well as transfer, including file deletion, renaming, copying, directory changing and space inquiry, etc; thus Kermit is really a file transfer and management protocol.

.. Arpanet FTP:

The Arpanet File Transfer Protocol (FTP) is very similar to Kermit, except that the file transfer server is a dedicated network resource, always available, rather than a process that must be started by the user on the remote end. The user runs an FTP "client" program and specifies the host name, user ID, and password of the remote system. The connection is established and the access permission checked, and then the user is allowed to send or get files, and to perform file management functions according to the remote system's access permissions.

As in Kermit, the user must specify whether files are text or binary so that the application can decide whether to do conversions. Text files are further classified as print or non-print files, so that simple printer control conversions can be done (ASA or Fortran-style vs LF, FF, VT, etc).

Although conversions are not done on binary files, options are required to let the user decide how to store binary data when the target system has a different word length (e.g. 32 bit data sent to a 36-bit word machine, or vice versa). In addition, the "client" and "server" programs let each other know what kind of system they're running on, and if the systems are the same, then special liberties can be taken (for instance, conversion to CIR can be skipped, or files can be transferred in chunks that correspond to the computer's disk block size for efficiency). Random access ("paged") files are also allowed, with each page preceded by a page header, specifying the position and length of each page.

FTP defines a Network Virtual File System (NVFS) with standard commands and file and directory naming conventions. There are separate control and data connections. The control connection (used for establishing connections, requesting files, setting file types, etc) uses the TELNET virtual terminal protocol (see below) and NVFS syntax. The data connection sends file data, converted (if necessary) to ASCII, with lines terminated by CRLF, unless otherwise specified.

.. ISO FTAM:

The ISO File Transfer, Access and Management protocol is the most ambitious file transfer scheme to date. To resolve differences among computer systems of different manufacture, operating systems, etc, it defines a common intermediate representation of an ideal file system, called a "virtual filestore", which embodies all the concepts of directory structure, file formats, attributes, naming conventions, access rules, etc. The file transfer application layer converts to and from virtual filestore conventions during transmission.

FTAM uses various ASEs for connection establishment, data transfer, and connection release. Data syntax conversion is negotiated during connection establishment and is performed by the presentation layer. The session layer, accessed via CCR, allows long file transfers to be checkpointed.

(...here's another crack in the layered structure...)

When an FTAM PDU is given to the presentation layer for transmission, the latter has no way of distinguishing FTAM PCI from data to be converted. Obviously, PCI should not be converted. Therefore, FTAM transfers PCI and data separately, within separate presentation contexts.

(remind you of Kermit S and A packets?)

Data transfer may occur on a whole-file basis, or on pieces of files; for instance, FTAM may be used to update a remote database. The abstract syntax of records within a sequential file are according to ASN.1, and FTAM defines its own mechanisms for more complex (e.g. hierarchical tree-structured) files, or for unstructured binary files.

In addition to data transfer, FTAM includes the following functions:

- Access control
- Accounting and charging
- Concurrency control via locks (multiple writes, updates)
- Checkpointing
- A comprehensive set of error recovery mechanisms

.. Exchange of Business Documents:

The ANSI X.12 suite of standards apply to a special instance of file transfer: the exchange of business documents like purchase orders and invoices. An application-level language called BDI (Business Data Interchange) has been defined to describe business transactions in terms of document headers (company name, address, date, PO number, terms, etc), line items (description, quantity, price), summary (total amount), etc. (Also EDI...)

.. Distributed File Systems:

When a file transfer protocol is embedded within a computer's file system, then remote files may be accessed as though they were local, and all the file operations (create, copy, rename, delete, etc) may be performed transparently, regardless of the physical location of the file. SUN's Network File System (NFS) is the best-known example.

.. Disk servers:

A disk server is a file-oriented application, but not a file transfer protocol. Rather, it is a network application that replaces an operating system's disk device driver. Rather than treating files as logical units, it allows applications that call upon the OS's disk service (read sector, write sector) to access remote disks as though they were local. The difference is that operations occur on the sector level rather than the file level, so that no conversions are done, and in fact no knowledge of the file system is required in the network application; all of this knowledge is embodied in the OS's higher level file service, which is ignorant of whether the file is on a local disk or "in the network". Disk servers are commonly found on local area or PC networks.

.. Distributed Databases:

Distributed databases are another variation on file transfer, but in this case multiple users are accessing a database that may be spread across several systems, querying and updating records in a random way. A distributed database is therefore a special case of a distributed file system, in which many physically separate files may be logically mapped into a single file, which many users on many different systems may be accessing simultaneously. There are at least two special problems to be solved: locating the physical file that contains the desired record or table, and prevention of uncoordinated queries or updates. The former requires a network-wide directory or index service (which is not easy when the database is relational), and the latter must be handled by a distributed lock manager.

. Virtual Terminal Service:

And what about terminal sessions? How do you log in from a Data General terminal over a network to a host that only knows how to control DEC terminals? How do you transfer structured documents between applications that represent them in different ways?

Since there are many different terminals on the market, each with its own peculiar set of screen control sequences and function keys, it is impractical to expect each computer to know the details of every kind of terminal. Your computer may know how to interact with your particular terminal (or your PC which is programmed to emulate a particular terminal), but that does not mean that other computers on the network can do so too. Virtual terminal service defines a common intermediate representation (CIR) for terminal control sequences, so that each system need understand only its own terminals plus the CIR, or "virtual terminal". Incoming CIR directives are translated to control sequences appropriate to the particular terminal, and terminal function or interrupt keystrokes are converted to CIR for transmission. Conversely, the remote host virtual terminal server translates an application's outbound terminal control sequences into CIR, and incoming CIR into interrupt characters or function-key codes.

(...here we could discuss X.3, X.28, and X.29...)

The ARPANET Network Virtual Terminal (NVT) specification (TELNET, RFC854) provides a very simple virtual terminal, with CIR only for those keys that effect how the terminal can control the computer -- BREAK, halt process, discard output, erase line, erase character, etc. For the remote host to control the local terminal's screen, the specific terminal type must still be supported by the remote host.

(contrast with MIT SUPDUP or UNIX termcap...)

The ISO virtual terminal protocol (VTP, ISO/DIS 9040-9041), on the other hand, includes abstract representations for every conceivable terminal function -- erasure, emphasis, fore- and background color, 2- or 3-dimensional display, windows, multiple character sets and fonts, cursor addressing, character attributes, block mode, etc etc. E.g. you can log into a

VAX as a VT100 from an IBM mainframe 3270 block-mode terminal. VTP is very, very, very complicated.

. Electronic Mail:

Electronic mail is a special kind of file transfer, in which "files" are really messages addressed to a particular user on the network. The message generally consists of a group of "headers" (like From, To, Date, and Subject) and a message body (text). Messages are transferred from the originator to the "mailbox" of the addressee(s) over the network (or on the local system). Most mail protocols provide for the authentication of the sender and verification of the receiver via special protocol messages outside of the text, allowing mail to be transferred reliably and securely. That is, the messages headers are for use only by the user program, not the network.

Delivery is usually accomplished not by user programs, but by privileged system processes, to prevent forged mail and unauthorized file access. Most mail systems provide immediate delivery when possible (i.e. when a network connection to the target system can be established), and also queued service (so that messages can be delivered later if the remote system is down).

.. ARPANET SMTP (RFC 821) and Text Message Format (RFC 822):

The ARPANET protocols include a standard for mail delivery (Simple Mail Transfer Protocol, or SMTP), and for message format. SMTP, as its name implies, is a simple mechanism allowing one host to deliver mail to another. The typical sequence is "Here comes a message for users A, B, and C on your system" "OK, but I don't have a user B" "OK, here's the message" "OK, I got it".

The message format standard defines what header items (like To, From, Date, and Subject) are required, what other ones are optional, and in what format they should appear. This standard allows a user program (mail manager) to be written that will accept and understand mail from any conforming system. The user program may be as powerful as desired. For instance, a user might be able to give a command like "forward to user christine at host xxx all messages from users at host yyy which contain the words 'kermit' or 'frog' and were received between April 13 and April 17". In this case, the mail program acts almost like a database language, extracting the desired messages (based upon the contents of their From, Date, and Text fields) and forwarding them as indicated.

A major issue in ARPANET mail, as the network grows by the interconnection of hundreds of local area networks, is addressing. At first, when there were only a couple hundred hosts on ARPANET, it was sufficient for all mail to be addressed to user@host. This meant that each host's mail system needed a host table containing the names and corresponding network addresses of all the nodes. With nodes now numbering in the many thousands, this is no longer practical -- host tables exceed the memory size of many computers. Thus, network host names are now expressed in a hierarchical manner. Instead of mailing to "christine@cucca" we now mail to "christine@cucca.cc.columbia.edu". When someone sends mail to us from outside, it proceeds through a hierarchy of "which network?" (edu), "which site on the network?" (columbia), "which department at the site?" (cc), and "which computer in the department?" (cucca). Thus, any given system need only know the addresses of sites on its own network, and the addresses of gateways to other networks. When a message arrives at a gateway, it is inspected for a site address and forwarded appropriately. (This discussion applies to addressing in general, and is not specific to mail, though it is currently most visible to mail users).

.. CCITT X.400 Message handling system (MHS)

The X.400 series of recommendations specifies an application for interpersonal electronic mail. As in ARPANET, there is a user program (or "user agent"), which is concerned with the message itself (heading and text, or "content"), and a message transfer agent that executes the delivery protocol ("envelope"). These are two separate sublayers of the OSI application layer. X.400 implementations have already started to appear on the market.

X.400 is very similar to ARPANET mail, but (naturally) much more complicated (the X.400 series of recommendations is about 300 pages long). Much attention is given to address formats, since they are designed to span all users on all networks in the world. A full-blown X.400 address may have many parts, including a country name, administrative domain name, private domain name, personal name, organization name, department name, X.121 address, telematic terminal identifier, etc etc. There are long lists of valid formats and entries in these catagories.

Protocol is given for access management, content type indication, encoding, time stamping, message identification, non-delivery notification (and prevention thereof), recipient lists, alternate recipients, deferred delivery (and cancellation thereof), delivery notification, (non-)disclosure of other recipients (bcc's), return of contents, delivery query, auto-forwarding, primary and copy recipient indication, expiration dates, cross-reference indication, indication of importance, sensitivity, obsolescence, etc; reply request, encryption, and much more.

There are already more than 20 X.400 products on the market. It has become a relatively mature standard. But to be truly universal, it will need a DIRECTORY SERVICE, so that users can find out the e-mail addresses of people they want to send mail to. When you consider that the number of worldwide computer users may soon number in the billions, this is not an easy function to design or implement. The CCITT is working on recommendation X.500 to "address" this issue. Think of lookups in a billion-entry database... responses could take hours. What does that do to timeouts? Breaking the problem into domains (country, network, organization, person) delegates the responsibility, but adds complexity...

There is a Version 2 of X.400 on the way, that will allow for greater security, mailboxes, physical delivery, etc.

Incidentally, you would think that X.400, being an international standard, would make some allowances for transmission of messages in different character sets. But it does not, no more than do the Arpanet protocols. X.400 allows only International Alphabet 5 (IA5 = ISO646 = US 7-bit ASCII with 9 positions left open for "national characters").

References

Roux, E., "OSI's Final Frontier: The Application Layer", Data Communications, Jan 88, p.137-145.

Clark, D.D., "Modularity and Efficiency in Protocol Implementations", RFC817 (1982). An iconoclastic view of layering.

ISO/DIS 8649, Common Application Service Elements (CASE)

ISO/DIS 8824, Specification of Abstract Syntax Notation One (ASN.1). ISO/DIS 8825, Specification of Basic Encoding Rules for ASN.1.

Postel, J., File Transfer Protocol, RFC959 (Oct 1985). ISO/DIS 8571/1-4, File Transfer, Access and Management (FTAM).

Postel, J., TELNET Protocol Specification, RFC854 (May 1983). ISO/DIS 9040, Virtual Terminal Service - Basic Class (1986). ISO/DIS 9041, Virtual Terminal Protocol - Basic Class (1986).

HIGH LEVEL ISSUES

. Performance:

Assuming that systems can be connected over a network, does the connection provide adequate performance? A set of criteria has been developed by the US Government in FED-STD 1033 to measure 26 different digital communication system performance parameters in the categories of efficiency (access time, bit rate, etc), accuracy (bit error rate, extra bit probability), and reliability (bit loss probability, etc), as well as the common metrics MTBF (Mean Time Between Failures) and MTTR (Mean Time To Repair).

Measurements like these are important in establishing system design goals, procurement requirements, and test specifications. Depending on the network technology, measurements will vary considerably under different conditions of load, etc. For instance, a contention-based network like Ethernet will perform well until it reaches a point of saturation, and then performance will degrade rapidly. The performance of datagram networks will also vary according to congestion.

Heated debates continue to rage over the performance of connectionless vs connection-oriented networks and protocols. blah blah...

. Conformance Testing:

When a vendor announces a network product that is compatible with OSI or some other networking architecture, customers must be prepared to test wheter the product works as claimed under both normal and abnormal conditions. Techniques are required for introduction of failures and errors at all levels, and for determining whether data is delivered correctly. Standards organizations like ISO, CCITT, ANSI, and NBS are all involved in this area.

The NBS has developed a formal notation for description of protocols, and programs for interpreting this notation, along with a fault generator to insert errors and programs to measure the performance of the model network. This technique cannot "prove" the correctness of a protocol specification, but it can evaluate its performance statistically. It also cannot evaluate an off-the-shelf vendor product. Organizations do exist, however, to validate implementations of specific protocols, like X.25.

. Security:

In the days of standalone computers, security was an issue that could be dealt with in relatively simple ways -- locks, etc. As soon as modems were connected to computers and dialins were allowed, the potential for unauthorized use escalated dramatically, and breakins continue to this day.

Networks open systems up even further to security violations. Before there were networks, "hackers" had to dial hundred of numbers at random, hoping to get a carrier tone. Now they can dial a single number (like their local Telenet or Tymnet node), and from there every system they attempt to access is guaranteed to be a computer. Furthermore, host-to-host networks put the power of large computers at the disposal of hackers trying to break into other computers.

The pendulum swings... The 1960s and 70s saw increasing need for distributed computing, access from home, host-host connections, etc, and during this period the hard problems of data communication were solved -- flow control, error detection and recovery, routing, congestion control, multiplexing, etc etc. But once reliable communication was established, the need to restrict its use became paramount.

A special problem is associated with PC networks, and particularly with PCs attached to host-host networks. In this context, a "host" is a multiuser computer in which users authenticate their access with a username and a password, and where they have their own file storage area which they can protect from other users. Most PCs (like MS-DOS systems, the Apple Macintosh, etc) do not have usernames and passwords, but rather, assume that anyone who has access to the PC itself also has unlimited access to all its files.

To put PCs on a network, then, opens the network up to "anonymous" access, in which the identity of the PC user cannot be authenticated, and it opens the PC file system up to uncontrolled incursions. To address these problems, vendors of PC networks provide special "file server" machines which have user IDs and passwords, just like hosts. Shared files are on the file server, subject to access controls specified by their owners.

Any network link is subject to tapping, either directly, or by monitoring electromagnetic emissions, etc (optical fiber might be an exception). In a complicated network connection, there's no guarantee against monitoring. The only practical solution is encryption. But where can it be done?

- Datalink layer - This can be quite effective. It means that every single bit that is transmitted can be encrypted, and therefore presumably immune to monitoring. But either the datalink software itself must be modified to do this, or special scrambler/unscrambler boxes must be built which sit physically between the network interface and the transmission medium. Such boxes are starting to appear on the market (e.g. recent DEC Ethernet encryption device). Complications on shared media when some but not all nodes have these boxes...

- Network layer - Not much can be done here, as the network layer must have its protocol information (addresses) intact in order to function. Some argument can be made that the IP approach lends itself well to security, because the IP packets can take different routes, so a spy on a single link cannot see the whole conversation. If the IP packets are encrypted, then the spy can't even see which ones go with which connection.

- Transport layer - End-to-end encryption also makes sense -- it can be host-based, leaves the communication subnetwork alone to do its job, and can therefore be done all in host-resident software. The entire transport PDU can be encrypted.

- Presentation layer - The ISO says it belongs here, where user data is transformed into and out of "transfer syntax". But this leaves the transport and session information in plain text, allowing spies to see who is talking to whom.

- Application layer - Can provide any and all security services.

However, even the contents of the messages (say transport layer and up) can be encrypted, it is still possible for spies to do network traffic analysis to find out, at least, what host talks to what other host, and how much.

ISO TC 97 SC 21 WG 1 recently finalized ISO 7498/2, Security Architecture, the beginning of a framework for what kinds of security services can go in what layer. Such services include peer identity authentication, data origin authentication, access control, connection confidentiality, traffic flow confidentiality, etc. Methods used to provide these services at the various layers include encryption, digital signature, access control lists, traffic padding, routing control, "notarization", ...

. Outstanding Issues:

Major issues to be resolved in OSI include naming and addressing (how to address uniquely every application process in the world -- and beyond?) and data security, especially across national boundaries.

The higher OSI layers are very immature, and their boundaries and terminology are even less clear than those at the lower layers. What functions belong in the session, presentation, and application layers? Are all three really necessary? Or will the upper three layers tend to be combined into each application program?

As higher-level protocols proliferate, an unambiguous precise method for describing them (and even generating the software that executes them) is necessary. Many have been proposed, some are already in use, but there is no standard yet.

Will there ever be a standard "user interface"?

TRENDS

Networking is all proceeding in the ISO/OSI direction. Major vendors like IBM and DEC are gradually bringing their proprietary protocols into conformance with OSI standards, and the US government has mandated OSI networking in future contracts, and in particular, the ARPANET TCP/IP suite of protocols and associated applications will be "migrated" to OSI.

However, OSI software will be a long time in the making, and it will be complex and expensive (if not in dollars, then in computer resources). And as higher-level protocol definitions are being fleshed out in all their generality, they begin to strain the capabilities of today's common operating systems (imagine implementing full-blown FTAM for the MS-DOS file system).

Furthermore, the strict layering of the OSI model does not necessarily mesh well with operating system design. Because of performance or other considerations, particular layers (especially the upper four) may find themselves split between operating system modules, user programs, libraries, etc, or may be lacking altogether. For example, it is the transport layer's responsibility to break user data into network-size packets. But because of layering principles, it must do this in total ignorance of the application. But the packet size might be very important to the application -- file transfer works best with very long packets, but full-duplex virtual terminal performance could be very poor unless packets were very short.

While "waiting for ISO", many organizations are making do with TCP/IP; compared to OSI, the ARPANET protocols are simple, work well, and are available for many different systems (often at little or no cost), operating over both LANs and long-haul connections.

And for those systems that don't have network support available (or can't afford it), RS-232-based communications -- terminal emulation and asynchronous protocols like Kermit -- will continue to fill the "network gaps" for many years to come.

And let's not forget ISDN. Although there has been little evidence of it yet, there is a good chance that one day a OSI-based telephone-switched digital network capable of carrying data and video as well as voice will span the globe, potentially allowing anyone with a telephone to transfer data or pictures with anyone else. As the higher-layer OSI protocols are completed, and the communications infrastructure is laid, universal interconnection will become increasingly possible. (See Also FED-STD 1037, Glossary of Telecommunications Terms.)

3270 - A series of IBM synchronous, EBCDIC, block mode, half-duplex terminals, the preferred (by IBM) type for use with IBM System/370 mainframes, and the generic term for this type of user-system interaction.

Access Method - IBM's way of saying "device driver".

Accunet - Digital service from AT&T Communications, offering circuits at speeds from T1 (1.54Mbps) down to 2400 bps, as well as packet services.

ACK - An acknowledgement. ASCII character Control-F, or a message in a particular protocol that acknowledges successful receipt of another message.

ACU - Automatic calling unit, a computer controlled telephone dialer.

Adaptive routing - A type of network routing in which the path from one node to another can vary according to changing conditions.

ADCCP - Advanced Data Communications Control Protocol, the ANSI standard datalink protocol.

Address - A location in memory, on a disk, or in a network, expressed as a number ranging from 0 to the number of the highest location. A location in memory may be a byte or a word; a location on disk is a block. Memory addresses are sequential, disk or network addresses usually are field encoded (track-sector-block, net-subnet-host-socket, etc).

ANSI - The American National Standards Institute, a nonprofit nongovernmental organization supported by more than 1000 trade organizations, professional societies, and companies, which serves as the USA's representative to the Internaional Organization for Standardization (ISO). ANSI standards relevant to asynchronous data communication include the ASCII specification, the character structure and parity standard, and the bit-sequencing standard.

APPC - Advanced Program-to-Program Communications (IBM), a process-to-process communication protocol based on LU 6.2 and PU 2.1.

Application Layer - The highest layer in the ISO/OSI and most other networking schemes; the layer where the actual work ("data processing") is accomplished.

Application Program - A program that can be written or run by a user (as opposed to a privileged program, or operating system software).

ARPANET - Advanced Research Projects Agency Network, a research computer network whose development was sponsored by the US Dept of Defense. One of the first packet switched networks. Characterized by connectionless networking, uses the DoD TCP/IP transport/internetworking protocols.

ARQ - Automatic Retransmission (or Repeat) Request.

ASCII - American Standard Code for Information Interchange, ANSI X3.4-1977, a 128-character code used almost universally by computers for representing and transmitting character data, in which each character corresponds to a number between 0 and 127.

ASCII protocol - Transfer of ASCII data from one computer to another with no error detection or correction.

Asynchronous - Character- or byte-oriented data transmission in which no out-of-band coordination takes place between the sender and the receiver, where character boundaries must be deduced from the structure of the data itself. In serial communication, delimitation is accomplished by start and stop bits.

Backbone - A high-capacity cable or network used to interconnect lower-speed networks or devices.

Balanced - Said of an HDLC-like datalink protocol in which both stations may transmit both commands and responses. Also, a mode of physical transmission in which each signal occurs on two wires, as in RS-422 and RS-423. Opposite of Unbalanced.

Bandwidth - The frequence range available for signalling (highest minus lowest), or more informally, the capacity of a circuit.

Baseband - A medium that carries just one information channel.

Baud - Unit of signalling speed. Typically, but not necessarily, equivalent to bits per second.

Baudot - A 5-bit character code used in teletype communications.

BASIC - Beginners All-purpose Symbolic Instruction Code, an interactive programming language noted for its ease of use, convenience for program development, slowness of execution, and poor structure.

Batch - A mode of transmission or execution in which an entire "job" is executed from beginning to end. Also, any computer application that requires little or no interaction from the user.

BCC - Block Check Character, see Block Check.

BCD - Binary Coded Decimal

Binary - (a) Base-2 number system with the digits 0 and 1; (b) any selection or condition in which there are two possibilities; (c) non-textual.

Binary Coded Decimal (BCD) - A 6-bit character code used on older IBM computers.

Binary Synchronous Protocol (Bisync, BSC) - An early datalink protocol developed by IBM.

Bit - Binary Digit (0 or 1). Smallest unit of storage or transmission in a computer.

Bit stuffing - a technique for achieving datalink transparency by inserting and stripping bits into data whose patterns are reserved for control purposes.

Block - (a) A delimited, or fixed-length, sequence of bytes; (b) to wait for a requested action to complete.

Block Check - A quantity formed from all the data in a block, for instance, by adding up all the bytes (a checksum) or combining them in some other way (like CRC), and then appended to the block itself, so that the recipient of the block can determine whether it was corrupted in transit.

bps - Bits per second.

BREAK - In asynchronous serial communication, a spacing condition that lasts at least 250 milliseconds, thereby causing a framing error which can be detected by the receiving UART.

Bridge - A device that filters packets between two adjacent branches of a shared-medium (broadcast) local area network.

Broadband - A communication medium that can carry a greater frequency range than normal voice-grade transmission media. Typically split into many channels using frequency-division multiplexing.

Broadcast - A method of transmission in which all devices receive simultaneously.

Bus - A shared communication medium along which data signals travel to all attached devices simultaneously, with a method provided for arbitrating contention. The basis for bus-topology communication networks like Ethernet or token bus.

Buffer - A storage area for data. In data communications, a device driver puts arriving data into a buffer in "real time", and the application program takes the data out at its leisure, freeing the application program from time-dependent considerations.

Byte - A unit of storage intended to hold a character, usually 8 bits long, abbreviated B. 8-bit bytes are sometimes called "octets". Computer memory and disk capacity is often measured in thousands (K) or millions (M) of bytes, e.g. 256KB. In most computers, a byte is the smallest unit of information that may be moved into and out of memory.

Byte oriented - a communications device, medium, or protocol in which a character (or byte) is the smallest unit of information that can be transferred.

Byte stuffing - A datalink technique for "quoting" data bytes that are the same as those used for framing or synchronization.

CAD - Computer Aided Design.

Carrier - A steady signal capable of being modulated (in either amplitude or frequency) by a another signal representing binary data. Also, a supplier of communication services ("common carrier").

CASE - See Common Application Service Elements

CATV - Community Antenna Television ("cable TV"), a transmission system using 75-ohm coaxial cable, also used in broadband networks.

CBEMA - Computer Business Equipment Manufacturers Association.

CBMS - Computer Based Message Switching (Electronic Mail).

CC - Connect Confirm.

CCITT - Comite Consultatif International Telegraphique et Telephonique, a committee of the International Telecommunications Union (ITU), which in turn is an agency of the United Nations. The CCITT issues standards, called Recommendations, in the area of data communication. The X series of Recommendations (X.25, X.29, etc) deals with digital networking, and the V series (V.21, V.22bis, V.26ter, etc) addresses data transmission over the switched telephone network.

CD - Carrier Detect: The RS-232-C signal used by the DCE to inform the DTE that carrier is present. Also called DCD and RLSD.

Cellular Radio - A technique for transmitting data via radio broadcast, often involving a mobile station, which is serviced by different transmitters as it changes location.

Central Processing Unit - The part of a computer that executes programmed instructions, consisting of an instruction decoder, arithmetic and logical processing units, etc.

Channel - Strictly speaking, a one-way communication path between a receiver and a transmitter. A full-duplex channel is actually two paths, either two separate wires, or one wire carrying signals at two frequencies.

Character - A discrete unit of information, a byte corresponding to a member of a given character set, like ASCII or EBCDIC. A character may be printable (letter, digit, punctuation, etc) or nonprintable (used for control purposes).

Character oriented - Said of a datalink protocol in which characters from a particular character set (like ASCII or EBCDIC) are used for control purposes.

Checksum - A block check based on the arithmetic sum of all the bytes in a block.

Circuit - A communication path two points.

Circuit switching - A method of communication where an electrical connection is made between between the calling and called stations, for their exclusive use until the connection is released.

Clock - A device that controls a computer's or communication device's frequency, speed, etc. In data communications, the clock can be used to generate interrupts to be used for sampling, polling, or timeout.

Clocking - Time synchronizing of communications data.

Coaxial cable - Transmission medium with inner and outer conductors separated by insulator, allowing high bandwidth, low susceptibility to interference.

Code - (a) A set of symbols, such as ASCII or EBCDIC character codes; (b) In programming, another word for program, as in source code (the program text as typed by the author), object code (the machine-language output of the compiler).

Common Application Service Elements (CASE) - The part of the protocol in the OSI application layer that is common to all processes and interfaces with the presentation layer.

Common carrier - A supplier of communication services to the public, subject to regulation.

Compression - Transformation of data to reduce its size for storage or transmission, in a way that allows it to be reconstituted to its original form. Common types of compression are run-length encoding, Huffman encoding, and LZW encoding.

Connection-Oriented - Said of a protocol at any particular OSI layer in which there must be a connection establishment phase, a data transfer phase, and a connection release phase, and in which each message follows the path of previous messages during data transfer, so that messages arrive at the end system in order.

Connectionless - Said of a protocol at a given OSI layer in which there is no connection setup or teardown phase, and every message (frame, packet, etc) is independent from all others, so that messages can arrive at the end system out of order.

Connector - A physical interface, a plug, of either male or female gender, providing contacts for one or more wires within a cable, mating with a similar plug of opposite gender to provide the desired electrical circuits.

Console - The primary input/output device with which a person controls a personal computer or a timesharing session on a shared computer.

Contention - Multiple users competing for access to some shared resource, such as a transmission medium, or the read/write head of a shared disk.

Control Character - A nonprintable character in a particular character code, for instance an ASCII chararacter in the range 0 through 31 or ASCII character 127, contrasted with the printable, or graphic, characters, like those in ASCII range 32 through 126.

Control Program - Another word for operating system.

CPU - Central Processing Unit

CR - Abbreviation for Carriage Return (ASCII 13, Control-M). Also, in transport protocols, Connect Request.

CRC - Cyclic Redundancy Check, an error-detection technique in which all the bits in a message are divided by a special "generating polynomial" and the remainder is appended to the message.

CRLF - Abbreviation for Carriage Return, Linefeed, the sequence of ASCII characters (numbers 13 and 10) used on many systems to delimit lines in a text file.

CRT - Cathod Ray Tube, a terminal screen, a video terminal.

CSMA/CD - Carrier Sense Multiple Access with Collision Detection, the medium access and contention control method used with Ethernet.

CSU - Channel Service Unit. See DSU.

CTS - Clear To Send, the RS-232 signal that indicates readiness to accept data.

Cursor - The blob on your CRT screen that indicates the current character position.

Cyclic Redundancy Check - Bits calculated using binary polynomial division and added to a message block by the sender, then recalculated by the receiver to detect (but not correct) transmission errors.

Data - Information as it is stored in, or transmitted by, a computer or terminal. Plural of Latin datum but in common use as an English collective (singular) noun.

Data compression (see compression)

Data encryption (see encryption)

Datagram - A message that travels through a multinode network independently of related messages, possibly following different routes, with no assurrance of delivery in proper sequence, or at all.

Datalink Layer - The network layer that provides error-correcting point-to-point transmission.

Data Communication Equipment (DCE) - Devices whose only purpose is to connect Data Terminal Equipment together: modems, multiplexers, etc.

Data Terminal Equipment (DTE) - Data processing devices: computers, terminals, and printers, as opposed to Data Communication Equipment.

DCA - Document Content Architecture (IBM).

DCE - Data Communication Equipment, through which data terminal equipment (DTE) is connected to each other, or to a network.

DDCMP - Digital Data Communications Message Protocol, a byte-count oriented datalink protocol used by Digital Equipment Corporation.

DDM - Distributed Data Management (IBM).

DDN - Defense Data Network, another word for ARPANET.

Deadlock - A condition in which a pair of supposedly cooperating processes or devices are blocked from operating because each is waiting for the other to complete some action.

DECnet - Digital Equipment Corporation's networking product.

Dedicated Line - A communication line that is not dialed, i.e. that is always available, dedicated to a connection between a particular pair of devices.

Delay - The amount of extra time spent waiting for an expected response. For instance, the amount of time it takes for a message to be acknowledged.

Device Driver - A software component of a computer's operating system that controls or services an input/output device, such as a UART or a disk controller, in real time, providing a simple, buffered, time-independent interface to the application programmer.

DIA - Document Interchange Architecture (IBM).

Dialup - Establishment of a data communication circuit by dialing a telephone number.

Digital - Representation of data by discrete, rather than continuous, voltages or states. Opposite of analog.

DIS - Draft International Standard (as in ISO/DIS).

DMA - Direct Memory Access, between input/output devices and the computer's main memory without assistance from the CPU.

DNA - Digital Equipment Corporation Networking Architecture.

DOS - Disk Operating System. An operating system that uses a magnetic disk as its principle medium of permanent storage.

Download - Transfer data from a remote computer to a local one.

DSU - Data Service Unit. CSU/DSU combination serves in place of a modem when used over digital leased lines (DDS).

DTE - Data Terminal Equipment.

DTR - Data Terminal Ready, the RS-232-C signal used by the DTE to tell the DCE that it is operational.

Duplex - The degree to which a circuit permits two-way traffic. Half-duplex means traffic can go either way, but only one way at a time; full-duplex means traffic can go both ways at the same time.

Dynamic Routing - A type of routing in which each packet finds its way through a network independently, so that packets can arrive misordered at the end system.

E-Mail - Electronic Mail.

EBCDIC - Extended Binary Coded Decimal Interchange Code. The character code used on IBM mainframes.

Echo - The process by which a character typed at a terminal, or a device emulating a terminal, is sent to the screen. Local echo means the terminal itself copies the character to the screen; this is usually associated with half-duplex communication. Remote echo means the system to which the character is transmitted sends it back to be displayed, possibly modified.

Echoplex - Full duplex with remote echoing. Also, a technique for uploading files that uses the remote echo for error detection and correction.

ECMA - European Computer Manufacturers Association, one of the standards bodies of ISO.

EIA - The Electronic Industries Association. An organization of U.S. electronics manufacturers. Issues standards in the area of data transmission, such as the RS-232, RS-422, RS-423, and RS-449 standards. Some EIA standards are adopted by ANSI.

Encryption - Modification of data so as to make it unintelligible to anyone who does not possess the encryption key.

ESC - ASCII character 27, Control-[.

Escape Character - A character used to get the attention of an otherwise transparent device or program.

Escape Sequence - A sequence of characters opaque to an otherwise transparent device or program, which causes it to enter conversational mode, or to take some other action. For instance, the screen control sequences that are sent by a computer to a terminal to control the appearance of the screen.

Ethernet - A bus-topology baseband local area network using CSMA/CD medium access, originally developed by Xerox corporation.

FDX - Full-Duplex.

FEC - Forward Error Correction, a block check technique in which the receiver can not only detect errors, but also correct certain kinds of errors using information supplied in the block check.

FEP - Front End Processor.

FIFO - First-In-First-Out. A FIFO list is also called a queue. Items in the queue are serviced and removed in the order in which they arrived. Queues are used in timesharing systems for scheduling user jobs, and in networks for scheduling packets for transmission.

File - A named collection of data stored on a disk. A file group is a collection of files that can be referred to using a single file specification.

File server - A network node dedicated to providing file storage and access to other nodes.

FIPS - Federal Information Processing Standard, issued by the US Government, developed by the National Bureau of Standards, under the Dept of Commerce.

Fixed routing - A type of network routing in which there is only one predefined path from one node to another.

Flag - A variable that can have two possible values, often implemented as a single bit, used to control the behavior of a program, or to indicate the success or failure of an operation.

Flow Control - The process by which the flow of data in a particular direction is regulated by the receiver so that the arrival of data is coordinated with the capacity of the receiver to process it.

Forward Error Correction (FEC) - A frame check method that allows the receiver of the frame not only to detect errors, but also correct them.

Frame - A datalink-level message, clearly delimited and containing a block for error detection.

Framing - The method used to delimit characters in asynchronous serial communications. Each character is preceded by a start bit (space, 0) and followed by a stop bit (mark, 1), with a continous marking condition indicating no transmission. Also, the method used to delimit blocks of information (frames) at the datalink level.

Front End Processor - A communication processor for a host computer, which operates independently from it but is closely tied to it. The front end relieves the host from the burden of detailed control of multiple devices, and usually has direct access to the host's memory.

FTAM - File Transfer, Access and Management, an ISO application-level protocol, ISO/DIS 8571 (1986).

Full Duplex - A circuit that permits data transmission in both directions simultaneously.

Gateway - An interface between two different networks, capable of performing translations at a particular level.

GGP - Gateway-to-gateway protocol.

GKS - Graphics Kernel System, a standard for the common representation of simple 2-dimensional images.

Half Duplex - A circuit that permits data transmission in both directions, but only in one direction at a time.

Hamming Code - A forward error correction technique in which single-bit errors can be corrected in any byte in a message.

Handshake - A method for granting permission to transmit, usually on a half duplex channel ("line turnaround handshake").

HDLC - High Level Data Link Control, an ISO standard datalink protocol.

HDX - Half Duplex

Hexadecimal - Numeric notation in base 16, using the digits 0-9 and A-F to represent the numbers 0-15, with each hexadecimal digit corresponding to four bits.

Host - A computer capable of providing services to users.

IEEE - Institute of Electrical and Electronics Engineers, an organization that, among other things, issues standards involving local networks.

IFIP - International Federation of Information Processing.

Interface - Computer jargon for something that allows two otherwise incompatible components to work together by satisfying their respective physical and logical requirements and making any necessary conversions of format, timing, voltage, etc. A connectors and UARTs are examples of physical interfaces; a device driver is a software interface. The aspect of a software program that interacts with a person is sometimes called the user interface.

Internetwork Protocol (IP) - A protocol that allows packets to be routed through multiple interconnected networks, typically implemented as the upper sublayer of the network layer, and using connectionless (datagram) routing. IPs are defined for both ARPANET and OSI network architectures.

Interrupt - In computing, an event that occurs at an unpredictable time, which a program might take special action to service, after which it returns to what it was doing before. Most device drivers and communication programs are "interrupt driven," allowing them to respond rapidly (in "real time") to arriving data, even if they're in the middle of doing something else, like transmitting.

Interrupt Service Routine - A program in the computer's memory that handles a particular interrupt, e.g. the arrival of a character at the serial port. Typically does a very small task, like copying a character from the device register to a memory buffer, then dismisses itself, allowing the user-level software to continue operation where it left off.

Interrupt Vector - A table in the computer's memory that tells it where to find the interrupt service routine associated with each kind of interrupt.

IP - Internetwork Protocol.

ISDN - Integrated Services Digital Network, a CCITT project for standardization of a network allowing voice, video, and data.

ISO - The International Organization for Standardization, a voluntary international group of national standards organizations, including ANSI, that issues standards in all areas, including computers and information processing, and whose technical committee also maintains liaison with CCITT.

ISO/DIS - ISO Draft International Standard.

ISO DP - ISO Draft Proposal.

JTAM (or JTM) - Job Transfer and Manipulation, an evolving ISO standard for network batch job submission (RJE).

Kernel - The most central part of a piece of software. For instance, the resident portion of an operating system. Also called "nucleus".

LAN - Local Area Network.

LAP - Link Access Procedure, a datalink protocol used in X.25 networks.

LAPB - A "Balanced" variation of LAP, now preferred over LAP.

Layer - One level of a hierarchy of network functions or protocols.

Leased Line - A permanent, dedicated communication line rented from the telephone or other company, usually used in conjunction with synchronous modems at speeds in the 2400-56000 baud range.

LEN - Low-Entry Networking (IBM).

Line - (1) A physical communication path, such as a telephone cable; (2) A computer's interface to or designation for such a path; (3) a sequence of characters in a text file intended to print on one line of a page or screen.

Line Turnaround - The amount of time it takes to switch the directionality of a half duplex connection, or the mechanism used for doing so, such as XON handshake, RTS/CTS RS-232 signals, etc.

LLC - Logical Link Control - The upper sublayer of the IEEE 802 Ethernet datalink layer (802.2).

Local Area Network - A data communication network allowing computing devices in a building or on a campus to communicate, usually over a shared medium, at higher speeds than are possible with telecommunications.

Local Echo - Immediate display on the local screen, by a local agent, of characters sent to a remote computer. Associated with half-duplex communication.

Long Haul - Long distance, applied to connections, modems, or networks. Opposite of short haul. Also, wide area (opposite of local area).

LU - Logical Unit, access port for users in IBM SNA networks.

LU 6.2 - IBM Logical Unit 6.2, an emerging standard for process-to-process network communication.

MAC - Medium Access Control, the lower sublayer of the IEEE 802 datalink layer that specifies how to access a shared transmission medium: Ethernet CSMA/CD (802.3), token bus (802.4), token ring (802.5).

Machine - A computer.

Manchester Encoding - A binary encoding in which a bit is represented as a positive/negative voltage transition rather than a constant voltage level, used on Ethernet.

MAP - General Motors Manufacturing Automation Protocol, an evolving implementation of the 7-layer ISO OSI protocol suite.

Mark - (1) The voltage level used to express a binary 1 on a communication line; (2) A kind of character parity in which the parity bit of all characters is set to 1.

Medium - That through which data is transmitted -- copper wire, coaxial cable, optical fiber, empty space, etc -- or which it is stored upon -- magnetic disk, diskette, tape, etc.

Memory - The internal, volatile, high-speed, solid state storage of a computer, as distinguished from external, permanent, lower speed, rotating mechanical memories (e.g. disks, tapes) used for bulk storage.

Message - A unit of information, usually consisting of multiple bytes or characters, cast into some specified format for transmission.

Message switching - Transfer of entire messages through a network, storing them in their entirety at each node before relay to the next node. Also called store and forward.

Modem - Modulator/Demodulator, a device that converts between serial digital data as output from a UART and analog waveforms suitable for transmission on a telephone line.

Modulation - In data communication, impressing data upon a carrier wave by changing its amplitude, frequency, or phase.

Modulo - A maximum number to be used in counting, at which counting begins over again at zero. Any number modulo n is the remainder after dividing that number by n.

MTBF - Mean Time Between Failures.

MTTR - Mean Time To Repair.

Multiprocessing - Said of a computer system that allows multiple processes to be active simultaneously, by allocating small time slices to each one in some scheduled fashion, as in timesharing.

NAK - (1) ASCII character 25, Control-U. (2) In communication protocols, a negative acknowledgment, an indication that a message was not received correctly.

NAPLPS - North American Presentation Layer Protocol Syntax.

NAU - Network Addressable Unit (IBM).

NBS - National Bureau of Standards (US).

Network - A permanent arrangement allowing two or more computers or devices to communicate with each other conveniently and reliably at high speeds, over dedicated media, typically requiring special hardware and operating-system level software.

Network layer - In the ISO OSI reference model, the layer that is concerned with routing packets through a network to their final destination. May also include a sublayer conerned with routing messages between networks.

NFS - Network File System, a network file-sharing protocol developed by SUN Microsystems.

Node - A device or computer on a network.

Noise - Interference on a communication medium that corrupts data during transmission.

Nucleus - The most central part of a piece of software. For instance, the resident portion of an operating system. Also called "kernel".

Null Modem - A pair of connectors, possibly with a length of cable between them, allowing two DTEs (computers or terminals) to be directly connected without intervening DCEs (modems or multiplexers), supplying the required RS-232 signals by means of cross-connections and jumpers. An asynchronous null modem consists only of wires and connectors, a synchronous null modem also provides a clock signal.

Object code - Machine instructions, either directly executable by the computer, or requiring relocation by a loader prior to execution. Compare with source code.

Octet - A group of 8 bits. Another word for 8-bit byte.

Operating System - The software program that controls a computer at the most basic level, consisting of a collection of device drivers, a scheduler, memory manager, etc. Operating system functions, particularly device drivers, operate in real time, as distinguished from user programs, which are scheduled and managed by the operating system, and which must call upon the operating system to perform critical functions like device input/output.

OS - Operating System.

OSI - The Open System Interconnection reference model of the ISO.

Overhead - Information, such as control, timing, sequencing, routing, and error checking, added to data during transmission. Also, the computer resources consumed processing this extra information.

Overrun - The overwriting of data in a buffer with new data before the old data has been retrieved for use, e.g. in a UART's holding register, or a device driver's buffer.

Pacing - Another word for flow control.

Packet - A message consisting of fields whose locations and interpretation are agreed upon by the sending and receiving entities, to be transmitted (and possibly switched) as a whole, and typically containing sequencing, routing, error checking, and other control information as well as data.

Packet Switching - A technique, typically used in computer networks, to allow multiple users and hosts to share the same set of transmission media by breaking their data up into discrete packets, which may be intermixed and routed arbitrarily and still arrive at their various destinations in sequence and intact.

PAD - Packet Assembler/Disassembler, a device connecting one or more terminals or computers to a packet-switched network, providing conversion from the unguarded asynchronous communication that occurs between itself and the terminal to packet-switched communication between itself and the host selected by the user.

Padding - A method for allowing a receiving device to keep up with sustained transmission, by including extra characters at critical points to tie up the transmission line while the device is busy servicing the data received so far. For instance, certain kinds of printers need padding after a carriage return character to give them time to move the printing head back to the left margin. Used in lieu of full duplex flow control.

Parallel - All at once. In data communication, the transmission of all the bits in a byte (or word) together, each on its own wire, usually done only over very short distances. Compare with serial.

Parameter - A symbolic value, standing for, or to be replaced by, a real value.

Parity - An error detection method in which one bit is set aside to indicate some property of the remaining bits in a byte or word. Usually, it is the number, modulo 2, of 1-bits in the quantity. Odd parity means the parity bit is set to make the overall number of 1 bits odd, Even makes the overall number of 1 bits even. Mark parity means the parity bit is always set to 1, space parity means it's always set to zero. No parity means the bit that would otherwise be used for parity may be used for data.

PBX - Private Branch Exchange, a telephone system serving the internal needs of an organization and providing connection to the external phone system. Often used for data transmission as well as voice within the organization. May be digital or analog.

PC - (1) Personal Computer; (2) Program Counter (the address of the current program instruction being executed by a computer); (3) Phase Corrector (in synchronous modems); (4) Pacing Count (in SNA).

PDN - Public Data Network.

PDU - Protocol Data Unit: the data and control information associated with a particular OSI protocol layer, e.g. a TDPU for the Transport layer.

P/F - The Poll/Final bit in HDLC and friends.

Physical Layer - The ISO OSI layer concerned with control of the communication device.

PLV - Parameter-Length-Value notation: a code specifying a parameter, followed by the length of the parameter, and then a parameter value of the given length.

Point to Point - Said of a transmission path that is direct, with no intermediate routing nodes involved, but possibly including transparent switches. For instance, a dialup phone connection is point to point, but a packet-switched network connection is not.

Polling - The process of asking each terminal or node in a prearranged sequence whether it has data to transmit.

Protocol - In data communication, a set of rules and formats for exchanging messages, generally incorporating methods of sequencing, timing, error detection and correction.

PSDN - Packet Switched Data Network.

PTT - Public Telephone and Telegraph Administration (Europe).

PU - The Physical Unit, in IBM SNA networks, that manages the communication resources at a given node.

PU 2.1 - The IBM Physical Unit associated with Logical Unit 6.2.

Public Data Network - A network, usually packet switched, providing access to the public on a subscription basis to potentially widely scattered and diverse services.

Queue - A list in which the first element entered is the first removed. Also called a First-In-First-Out (FIFO) list.

Real Time - Said of an environment in which events must be serviced promptly as they occur, rather than queued for later service.

Remote Job Entry (RJE) - Entering batch jobs from a remote "terminal", usually consisting of a card reader and line printer or device that emulates them.

Residual Error - An undetected transmission error.

Response Time - A measure of the interval between a stimulus and its response, for instance how long it takes a character to echo on a full duplex channel.

Retry - A second or subsequent attempt of the same operation, e.g. transmission of a packet.

RFC - Request For Comments, the ARPANET equivalent of a standard, e.g. RFC793 is the TCP standard.

RJE - Remote Job Entry.

Routing - The relaying of packets node-by-node through a network from the originating system to their final destination.

RS-232-C - An EIA standard that gives the electrical and functional specification for serial binary digital data transmission, the most commonly used interface between terminals (or computers) and modems (or multiplexers).

RTS - Request To Send, one of the RS-232-C signals, typically used by a terminal or computer to ask permission of a modem to transmit data to it.

SAA - IBM Systems Application Architecture, a specification of software applications, communications protocols, and user interfaces, allowing applications coded accordingly to be portable and interoperable among many different IBM and even non-IBM systems.

Satellite - In data communication, an object circling the earth in a relatively permanent, often geostationary orbit (always above the same spot), relaying data between earth stations (possibly through other satellites) usually via microwave, typically introducing delays in response time because of the great distances and possibly contention involved.

SSCP - System Services Control Point (IBM)

SDLC - Synchronous Data Link Control, a datalink protocol associated with IBM's System Network Architecture (SNA), similar to HDLC, and, like HDLC, a subset of ADCCP.

SDU - Service Data Unit, in OSI protocols, the next-higher layer's data, to be handled using the current layer's protocol.

Serial - In series, sequential, one after another. The dominant mode of transmission of binary data over distances greater than a few feet. Compare with parallel.

Server - A program, or intelligent device, that provides specified services to users, or "clients," in response to requests, usually over a communication line or network. Networks often supply file servers, print servers, name servers, etc.

Session - The period during which a user engages in dialog with a computer or a particular application.

Session Layer - The layer in the ISO OSI model which is responsible for communication between end-user processes.

Signal-to-Noise Ratio - The relative power of a signal to noise in a communication channel, measured in decibels (dB).

Signalled Error - A transmission error that is detected.

Simplex - Permitting data to travel in only one direction (like a radio broadcast).

Sliding Window - See Window.

SMTP - Simple Mail Transfer Protocol, the protocol used on the ARPANET and other networks for transferring electronic mail between networked systems.

SNA - IBM System Network Architecture.

SNADS - SNA Distribution Services (IBM).

Socket - In ARPANET, the network address of a process or application within a host.

Software - Computer program(s).

SOH - Start-Of-Header, ASCII and EBCDIC character number 1, Control-A. Intended to indicate the start of a packet or packet header.

Source Code - The text of a computer program as originally written by the programmer.

Source Routing - A kind of routing in which the user specifies each node in the path.

Space - (1) A binary 0 as represented on a transmission medium; (2) A blank, ASCII character 32; (3) A kind of parity in which the parity bit is always 0.

Splitting - Transmission of messages belonging to a particular layer over multiple next-lower-layer connections.

Star Network - A network in which each node is connected directly to a central hub.

Start Bit - In asynchronous serial transmission, the space (0-bit) that indicates that a character is starting to arrive, after one or more bit times of mark (1-bit) condition.

Stop Bit - In asynchronous serial transmission, the mark (1-bit) that terminates a character. It lasts for at least one bit time, and thereafter until the next character starts to arrive.

Store and Forward - See Message Switching.

Subnetwork - A subset of the layers of a network, usually the lower three (physical, datalink, network).

Sync (or SYN) - A bit pattern used by a synchronous line controller to achieve byte synchronization.

Synchronous - Having a constant time interval between successive bits or characters. A method of data communication in which characters (or arbitrary bit streams) may be transmitted without framing information (start and stop bits) to achieve greater throughput than possible with asynchronous communication, by using of out-of-band timing signals, but also requiring occasional resynchronization by means of in-band sync characters.

System - (1) A way of doing things; (2) A computer.

T1 - A digital long-haul medium capable of transmitting 1.544Mbps, typically multiplexed into 56Kbps or 64Kbps channels, originally used as telephone trunk lines, now seeing increasing use for data transmission.

TCAM - IBM mainframe Telecommunications Access Method, a device driver for asynchronous terminals.

TCP - Transmission Control Protocol, the Transport layer of the ARPANET protocol suite.

TDM - Time Division Multiplexing.

TDMA - Time Division Multiple Access.

Telecommunications - Asynchronous serial data communication, possibly (but not necessarily) involving dialup telephone connections and modems.

Telenet - A public packet switched network service.

Telnet - The Arpanet virtual terminal protocol, not be confused with Telenet.

Terminal - A device allowing a person to interact with a computer, with the person typing characters on a keyboard to send them to the computer, and with the computer's responses appearing on a screen or paper. Sometimes includes the ability to interpret special character sequences to accomplish screen formatting, but in general differing from a computer by not having local permanent memory or general-purpose programmability. Most terminals are ASCII, asynchronous, and character-oriented, but there are also other kinds, for example the IBM EBCDIC block mode 3270 series.

Terminal Emulation - Behaving like a terminal. Said of software that runs on PCs or other computers, which sends the user's typein out the serial port, and sends the port input to the screen. Sometimes includes the ability to interpret the same special command sequences (e.g. for screen formatting) that a specific real terminal would obey.

Terminal Server - A network device allowing ordinary terminals with no networking capabilities of their own to participate in a network, provided hosts share a common protocol with the terminal server.

Text - Computer data intended for a person to read, or typed by a person, consisting of only printable characters and those control characters necessary for format control (carriage return, linefeed, tab, etc). Text files can be transferred between unlike systems and still remain useful. Compare with binary file.

Throughput - A measure of how much data passes through a particular point per unit time.

Timeout - The process by which a program wakes up after waiting for some expected event (like input from a device) longer than a predetermined amount of time.

Timesharing - A style of computing in which multiple users simultaneously interact with the same computer, under scheduling control of the computer's operating system.

Token - A medium access technique used on local area networks, in which a "token" is passed from node to node, and only the node with the token is allowed to transmit. Suited to bus or ring topologies.

TOP - Boeing Technical and Office Protocol, an evolving implementation of the 7-layer ISO OSI protocol suite.

Topology - The layout nodes and physical connections in a network.

TPDU - Transport Protocol Data Unit.

Translation Table - A list of the numeric representations of characters in a given character set. The position in the list is the numeric value of a character in the set being translated from, the number located at that position is the value to be translated to. Also called translate table.

Transparent - Allowing data to pass through unmodified.

Transport Layer - The network layer responsible for end-to-end control of transmitted data.

TTY - Originally, Teletypewriter. Currently, any asynchronous ASCII terminal or computer that emulates one.

Turnaround - (1) Response Time; (2) Line Turnaround, i.e. the granting of permission to transmit on a half duplex connection.

TWA - Two-Way-Alternate, same as half duplex.

Twisted Pair - Pairs of insulated copper wire, 20-28 AWG, twisted around each other in helix fashion within an outer sleeve to minimize crosstalk interference during data transmission.

TWS - Two-Way-Simultaneous, same as full duplex.

Tymnet - A public packet switched network service.

Typeahead - The ability to send characters to a computer or device before it has requested them, possible only on full-duplex connections.

UART - Universal Asynchronous Receiver/Transmitter, the device that converts between parallel character data as stored in a computer's memory and asynchronous serial binary data as transmitted on a telecommunication line.

USART - Universal Synchronous/Asynchronous Receiver/Transmitter, like UART, but also handles synchronous communication.

USRT - Like UART, but only does synchronous.

Unattended - Referring to an operation that can proceed automatically, without human intervention.

Unbalanced - Said of an HDLC-like datalink protocol in which one station may transmit only commands and the other only responses. Opposite of Balanced. Also said of a transmission medium, like RS-232, in which all signals are measured against a common ground reference.

Unguarded - Said of data transmission in which no method of error detection and correction is employed.

Upload - Transfer data from the local computer to a remote computer.

User Program - A program that runs outside of the operating system's environment, whose scheduling is controlled by the operating system, and which must call upon the operating system to perform time-critical or privileged services.

V.24 - CCITT version of RS-232-C.

Virtual - Behaving as if it were a real (1) terminal, (2) circuit, (3) disk, (4) machine, ...

Virtual Circuit - A transmission path set up dynamically end-to-end, possibly shared by more than one user, with packets constrained to arrive in the same order in which they were sent, with no duplications or gaps.

Virtual Terminal - A common intermediate representation for a terminal and its control sequences and functions. Not the same as terminal emulation. The ARPANET TELNET protocol is a virtual terminal protocol. ISO/DIS 9040, Virtual Terminal Service, is another.

Voice Grade - Said of a telephone connection, either dialed or leased, intended for carrying voice rather than digital traffic; usually noisier, and with less bandwidth, than a digital or specially conditioned line.

VT100 - An asynchronous ASCII video terminal made by DEC and widely imitated. Uses the ANSI standard control sequences specified in ANSI X3.64-1978 and X3.41-1974. The VT100 is the basis for many later DEC models (VT102, etc, and the VT200 series), and for many PC terminal emulation programs.

VTAM - IBM mainframe Virtual Telecommunications Access Method; a device driver for asynchronous terminals and for network virtual terminals.

Window - The number of frames or packets that can be sent before requiring acknowledgement.

Word - A unit of storage in a computer's memory, usually the one used for numbers and addresses, directly addressable by the computer.

X.3 - A CCITT standard listing the functions of a PAD in a public data network.

X.21 - A CCITT standard for a synchronous physical layer for a public data network. X.21bis allows for asynchronous (e.g. RS-232) connections.

X.25 - A CCITT network-layer standard, the basis of many public packet-switched networks, included Telenet, Tymnet, Datapac, Transpac, Datex-P, etc etc.

X.28 - A CCITT recommendation that lists the commands that can be issued from a terminal to a PAD.

X.29 - A CCITT specification of the protocol between a PAD and a packet-mode host.

X.75 - A CCITT standard for interconnection of X.25 networks.

X.121 - A CCITT standard for network addressing.

X.200 - The CCITT version of the ISO OSI definition.

X.400 - A CCITT series of recommendations for Message Handling Systems (Electronic

Mail).

XID - Exchange Identification, an operation in HDLC-like datalink protocols allowing the two partners to configure themselves to one another by telling each other what capabilities they possess, and their receive-window size.

Xmodem - Asynchronous file transfer protocol based on the Ward Christensen protocol, MODEM, intended for use between microcomputers, widely found in commercial PC communication programs.

XNS - Xerox Network Systems

XON/XOFF - The most common in-band full duplex flow control method, in which the receiver sends an XOFF character when its input buffer is close to filling up, and an XOFF when it has made room for more data to arrive.

Table of Contents