The main task of the Internet and its TCP/IP protocol suite is to provide services for users. Although there are some specific client/server pro-grams that we discuss in future chapters, it would be impossible to write a specific client-server program for each demand. The better solution is a general-purpose client-server program that lets a user access any applica- tion program on a remote computer; in other words, allow the user to log on to a remote computer. After logging on, a user can use the services available on the remote computer and transfer the results back to the local computer. In this chapter, we discuss two of these application programs: TELNET and SSH.

**TELNET** is an abbreviation for *TErminaL NETwork.* It is the standard TCP/IP proto-

col for virtual terminal service as proposed by ISO. TELNET enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

**\*\*TELNET is a general-purpose client-server application program.**

**Concepts**

TELNET is related to several concepts that we briefly describe here.

**Time-Sharing Environment**

TELNET was designed at a time when most operating systems, such as UNIX, were operating in a **time-sharing** environment. In such an environment, a large computer supports multiple users. The interaction between a user and the computer occurs through a terminal, which is usually a combination of keyboard, monitor, and mouse. Even a microcomputer can simulate a terminal with a terminal emulator.

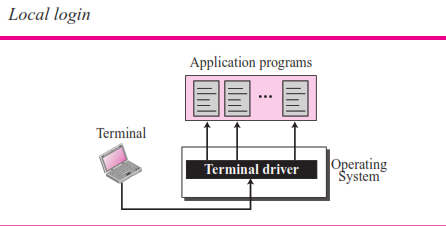
In a time-sharing environment, all of the processing must be done by the central computer. When a user types a character on the keyboard, the character is usually sent to the computer and echoed to the monitor. Time-sharing creates an environment in which each user has the illusion of a dedicated computer. The user can run a program, access the system resources, switch from one program to another, and so on.

***Login***

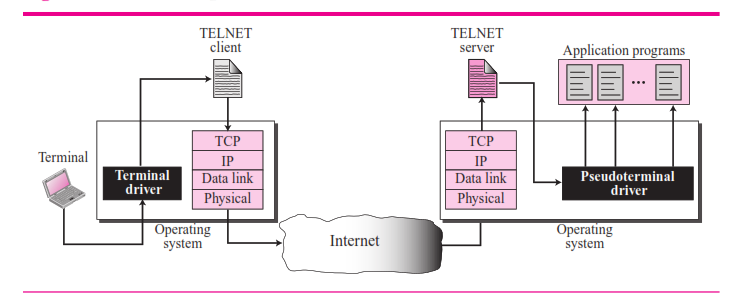
In a time-sharing environment, users are part of the system with some right to access resources. Each authorized user has an identification and probably a password. The user identification defines the user as part of the system. To access the system, the user logs into the system with a user id or login name. The system also includes password checking to prevent an unauthorized user from accessing the resources.

**Local Login** When a user logs into a local time-sharing system, it is called **local login*.*** As a user types at a terminal or at a workstation running a terminal emulator, the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system. The operating system, in turn, interprets the combination of characters and invokes the desired application program or utility (see Figure 20.1).

The mechanism, however, is not as simple as it seems because the operating sys- tem may assign special meanings to special characters. For example, in UNIX some combinations of characters have special meanings, such as the combination of the control character with the character z, which means suspend; the combination of the control char acter with the character c, which means abort; and so on. Whereas these special situations do not create any problem in local login because the terminal emulator and the terminal driver know the exact meaning of each character or combination of characters, they may create problems in remote login. Which process should interpret special characters? The client or the server? We will clarify this situation later in the chapter.



**Remote Login** When a user wants to access an application program or utility located on a remote machine, he or she performs **remote login.** Here the TELNET client and server programs come into use. The user sends the keystrokes to the terminal driver where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transforms the characters to a universal character set called *Network Virtual Terminal* (NVT) *characters* and delivers them to the local TCP/IP stack .



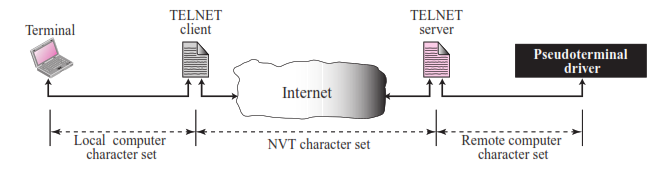
The commands or text, in NVT form, travel through the Internet and arrive at the TCP/IP stack at the remote machine. Here the characters are delivered to the operating system and passed to the TELNET server, which changes the characters to the corre- sponding characters understandable by the remote computer. However, the characters cannot be passed directly to the operating system because the remote operating system

is not designed to receive characters from a TELNET server: It is designed to receive characters from a terminal driver. The solution is to add a piece of software called a *pseudoterminal driver*, which pretends that the characters are coming from a terminal. The operating system then passes the characters to the appropriate application program.

**Network Virtual Terminal (NVT)**

The mechanism to access a remote computer is complex. This is because every com- puter and its operating system accepts a special combination of characters as tokens. For example, the end-of-file token in a computer running the DOS operating system is Ctrlz, while the UNIX operating system recognizes Ctrld.

We are dealing with heterogeneous systems. If we want to access any remote com- puter in the world, we must first know what type of computer we will be connected to, and we must also install the specific terminal emulator used by that computer. TELNET solves this problem by defining a universal interface called the **Network Virtual Terminal (NVT)** character set. Via this interface, the client TELNET translates charac- ters (data or commands) that come from the local terminal into NVT form and delivers them to the network. The server TELNET, on the other hand, translates data and com- mands from NVT form into the form acceptable by the remote computer. For an illus- tration of this concept, see Figure 20.3.



***NVT Character Set***

NVT uses two sets of characters, one for data and one for control. Both are 8-bit bytes

(Figure 20.4).

**Figure 20.4** *Format of data and control characters*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** |  |  |  |  |  |  |  |  | **1** |  |  |  |  |  |  |  |
| a. Data Character b. Control Character | | | | | | | | | | | | | | | | |

**Data Characters** For data, NVT normally uses what is called NVT ASCII. This is an

8-bit character set in which the seven lowest order bits are the same as US ASCII and the highest order bit is 0 (see Figure 20.4). Although it is possible to send an 8-bit ASCII (with the highest order bit set to be 0 or 1), this must first be agreed upon between the client and the server using option negotiation.

**Control Characters** To send **control characters** between computers (from client to server or vice versa), NVT uses an 8-bit character set in which the highest order bit is set to 1 (see Figure 20.4). Table 20.1 lists some of the control characters and their meanings. Later we will categorize these control characters on the basis of their functionalities.

**Table 20.1** *Some NVT control characters*

|  |  |  |  |
| --- | --- | --- | --- |
| *Character* | *Decimal* | *Binary* | *Meaning* |
| EOF | 236 | 11101100 | End of file |
| EOR | 239 | 11101111 | End of record |
| SE | 240 | 11110000 | Suboption end |
| NOP | 241 | 11110001 | No operation |
| DM | 242 | 11110010 | Data mark |
| BRK | 243 | 11110011 | Break |
| IP | 244 | 11110100 | Interrupt process |
| AO | 245 | 11110101 | Abort output |
| AYT | 246 | 11110110 | Are you there? |
| EC | 247 | 11110111 | Erase character |
| EL | 248 | 11111000 | Erase line |
| GA | 249 | 11111001 | Go ahead |
| SB | 250 | 11111010 | Suboption begin |
| WILL | 251 | 11111011 | Agreement to enable option |
| WONT | 252 | 11111100 | Refusal to enable option |
| DO | 253 | 11111101 | Approval to option request |
| DON’T | 254 | 11111110 | Denial of option request |
| IAC | 255 | 11111111 | Interpret (the next character) as control |

**Embedding**

TELNET uses only one TCP connection. The server uses the well-known port 23 and the client uses an ephemeral port. The same connection is used for sending both data and control characters. TELNET accomplishes this by embedding the control characters in the data stream. However, to distinguish data from control characters, each sequence of con- trol characters is preceded by a special control character called *interpret as control* (IAC). For example, imagine a user wants a server to display a file ( *file1*) on a remote server. She types: **cat file1**

in which *cat* is a Unix command that displays the content of the file on the screen. However, the name of the file has been mistyped ( *filea* instead of *file1*). The user uses the backspace key to correct this situation.

**cat filea<backspace>1**

However, in the default implementation of TELNET, the user cannot edit locally; the editing is done at the remote server. The backspace character is translated into two remote characters (IAC EC), which is embedded in the data and sent to the remote server.

**Options**

TELNET lets the client and server negotiate options before or during the use of the ser- vice. Options are extra features available to a user with a more sophisticated terminal. Users with simpler terminals can use default features. Some control characters dis- cussed previously are used to define options. Table 20.2 shows some common options.

**Table 20.2** *Options*

|  |  |  |
| --- | --- | --- |
| *Code* | *Option* | *Meaning* |
| 0 | Binary | Interpret as 8-bit binary transmission |
| 1 | Echo | Echo the data received on one side to the other |
| 3 | Suppress go-ahead | Suppress go-ahead signals after data |
| 5 | Status | Request the status of TELNET |
| 6 | Timing mark | Define the timing marks |
| 24 | Terminal type | Set the terminal type |
| 32 | Terminal speed | Set the terminal speed |
| 34 | Line mode | Change to line mode |

The option descriptions are as follows:

❑ **Binary.** This option allows the receiver to interpret every 8-bit character received, except IAC, as binary data. When IAC is received, the next character or characters are interpreted as commands. However, if two consecutive IAC characters are received, the first is discarded and the second is interpreted as data.

❑ **Echo.** This option allows the server to echo data received from the client. This means that every character sent by the client to the sender will be echoed back to the screen of the client terminal. In this case, the user terminal usually does not echo characters when they are typed but waits until it receives them from the server.

❑ **Suppress go-ahead.** This option suppresses the go-ahead (GA) character (see section on Modes of Operation).

❑ **Status.** This option allows the user or the process running on the client machine to get the status of the options being enabled at the server site. ❑ **Timing mark.** This option allows one party to issue a timing mark that indicates all previously received data has been processed.

❑ **Terminal type.** This option allows the client to send its terminal type.

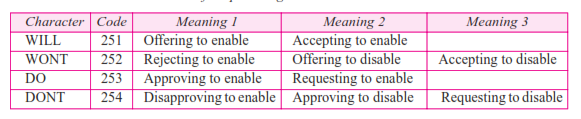
❑ **Terminal speed.** This option allows the client to send its terminal speed.

❑ **Line mode.** This option allows the client to switch to the line mode. We will discuss the line mode later.

***Option Negotiation***

To use any of the options mentioned in the previous section first requires **option nego- tiation** between the client and the server. Four control characters are used for this pur- pose; these are shown in Table 20.3.

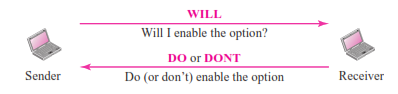
**Table 20.3** *NVT character set for option negotiation*



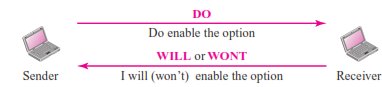
***Enabling an Option***

Some options can only be enabled by the server, some only by the client, and some by both. An option is enabled either through an *offer* or a *request*.

**Offer to Enable** A party can offer to enable an option if it has the right to do so. The offering can be approved or disapproved by the other party. The offering party sends the *WILL* command, which means “Will I enable the option?” The other party sends either the *DO* command, which means “Please do,” or the *DONT* command, which means “Please don’t.”.



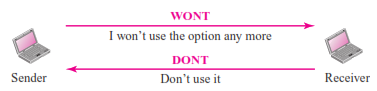
**Request to Enable** A party can request from the other party the enabling of an option. The request can be accepted or refused by the other party. The requesting party sends the *DO* command, which means “Please do enable the option.” The other party sends either the *WILL* command, which means “I will,” or the *WONT* command, which means “I won’t.”



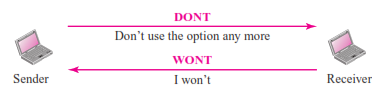
***Disabling an Option***

An option that has been enabled can be disabled by one of the parties. An option is dis- abled either through an *offer* or a *request*.

**Offer to Disable** A party can offer to disable an option. The other party must approve the offering; it cannot be disapproved. The offering party sends the *WONT* command, which means “I won’t use this option any more.” The answer must be the *DONT* com- mand*,* which means “Don’t use it anymore.” Figure 20.8 shows an offer to disable an option.



**Request to Disable** A party can request from another party the disabling of an option. The other party must accept the request; it cannot be rejected. The requesting party sends the *DONT* command, which means “Please don’t use this option anymore.” The answer must be the *WONT* command, which means “I won’t use it anymore.” shows a request to disable an option.

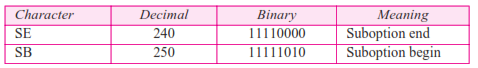


**Symmetry**

One interesting feature of TELNET is its symmetric option negotiation in which the client and server are given equal opportunity.

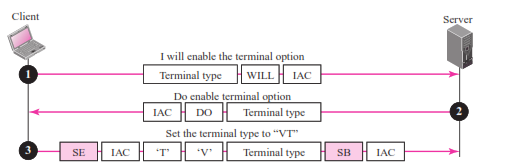
**Suboption Negotiation**

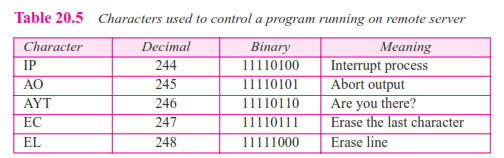
Some options require additional information. For example, to define the type or speed of a terminal, the negotiation includes a string or a number to define the type or speed.



**Controlling the Server**

Some control characters can be used to control the remote server. When an application program is running on the local computer, special characters are used to interrupt (abort) the program (for example, Ctrlc), or erase the last character typed (for example, delete key or backspace key), and so on. However, when a program is running on a remote computer, these control characters are sent to the remote machine. The user still types the same sequences, but they are changed to special characters and sent to the server.





**Out-of-Band Signaling**

To make control characters effective in special situations, TELNET uses **out-of-band signaling.** In out-of-band signaling, the control characters are preceded by IAC and are sent to the remote process.

**Modes of Operation**

Most TELNET implementations operate in one of three modes: default mode, character mode, or line mode.

***Default Mode***

The **default mode** is used if no other modes are invoked through option negotiation. In this mode, the echoing is done by the client. The user types a character and the client echoes the character on the screen (or printer) but does not send it until a whole line is completed. After sending the whole line to the server, the client waits for the GA (go ahead) command from the server before accepting a new line from the user. The opera- tion is half-duplex. Half-duplex operation is not efficient when the TCP connection itself is full-duplex, and so this mode is becoming obsolete.

***Character Mode***

In the **character mode,** each character typed is sent by the client to the server. The server normally echoes the character back to be displayed on the client screen. In this mode the echoing of the character can be delayed if the transmission time is long (such as in a satellite connection). It also creates overhead (traffic) for the network because three TCP segments must be sent for each character of data:

**1.** The user enters a character that is sent to the server.

**2.** The server acknowledges the received character and echoes the character back (in one segment).

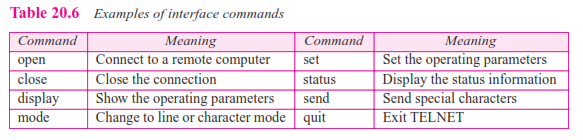
**3.** The client acknowledges the receipt of the echoed character.

***Line Mode***

A new mode has been proposed to compensate for the deficiencies of the default mode and the character mode. In this mode, called the **line mode,** line editing (echoing, character erasing, line erasing, and so on) is done by the client. The client then sends the whole line to the server. Although the line mode looks like the default mode, it is not. The default mode operates in the half-duplex mode; the line mode is full-duplex with the client sending one line after another, without the need for an intervening GA (go ahead) character from the server.

**User Interface**

The normal user does not use TELNET commands as defined above. Usually, the oper- ating system (UNIX, for example) defines an interface with user-friendly commands. An example of such a set of commands can be found in Table 20.6. Note that the inter- face is responsible for translating the user-friendly commands to the previously defined commands in the protocol.



**Security Issue**

TELNET suffers from security problems. Although TELNET requires a login name and password (when exchanging text), often this is not enough. A microcomputer con- nected to a broadcast LAN can easily eavesdrop using snooper software and capture a login name and the corresponding password (even if it is encrypted).

Another popular remote login application program is **Secure Shell (SSH)**. SSH, like

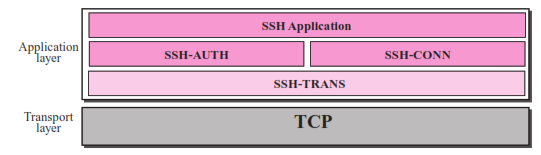
TELNET, uses TCP as the underlying transport protocol, but SSH is more secure and provides more services than TELNET.

**Versions**

There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1 is now deprecated because of security flaws in it. In this section, we discuss only SSH-2.

**Components**

SSH is a proposed application-layer protocol with four components



***SSH Transport-Layer Protocol (SSH-TRANS)***

Since TCP is not a secured transport layer protocol, SSH first uses a protocol that cre- ates a secured channel on the top of TCP. This new layer is an independent protocol referred to as SSH-TRANS. When the software implementing this protocol is called, the client and server first use the TCP protocol to establish an insecure proconnection. Then they exchange several security parameters to establish a secure channel on the top of the TCP. We discuss network security in Chapter 29, but we briefly list the services provided by this protocol:

**1.** Privacy or confidentiality of the message exchanged.

**2.** Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder.

**3.** Server authentication, which means that the client is now sure that the server is the one that it claims to be.

**4.** Compression of the messages that improve the efficiency of the system and makes attack more difficult.

***SSH Authentication Protocol (SSH-AUTH)***

After a secure channel is established between the client and the server and the server is authenticated for the client, SSH can call another software that can authenticate the cli- ent for the server.

***SSH Connection Protocol (SSH-CONN)***

After the secured channel is established and both server and client are authenticated for each other, SHH can call a piece of software that implements the third protocol, SSH- CONN. One of the services provided by the SSH-CONN protocol is to do multiplex- ing. SSH-CONN takes the secure channel established by the two previous protocols and lets the client create multiple logical channels over it.

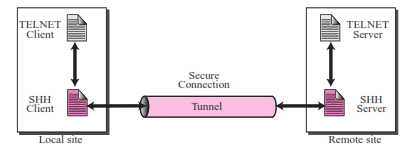
***SSH Applications***

After the connection phase is completed, SSH allows several application programs to use the connection. Each application can create a logical channel as described above and then benefit from the secured connection. In other words, remote login is one of the services that can use the SSH-CONN protocols; other applications, such as a file trans- fer application can use one of the logical channels for this purpose. In the next chapter, we show how SSH can be used for secure file transfer.

**Port Forwarding**

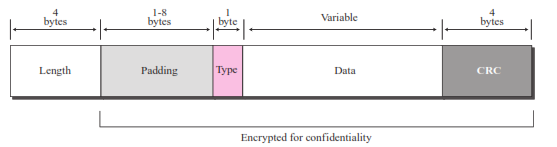
One of the interesting services provided by the SSH protocol is to provide **port forwarding**. We can use the secured channels available in SSH to access an application pro- gram that does not provide security services. Application such as TELNET (see Chapter 20) and SMTP (see Chapter 23) can use the services of SSH using port forwarding mechanism. SSH port forwarding mechanism creates a tunnel through which the messages belonging to other protocol can travel. For this reason, this mechanism is sometimes referred to as SSH **tunneling**. Figure 20.18 shows the concept of port forwarding.

Fig: port forwarding



We can change a direct, but insecure, connection between the TELNET client and the TELNET server by port forwarding. The TELNET client can use the SHH client on the local site to make a secure connection with the SSH server on the remote site. Any request from the TELNET client to the TELNET server is carried through the tunnel provided by the SSH client and server. Any response from the TELNET server to the TELNET client is also carried through the tunnel provided by the SSH client and server. We talk more about tunneling in Chapter 30.

SSH packet format:



The following is the brief description of each field:

**Length.** This 4-byte field defines the length of the packet including the type, the data, and the CRC field, but not the padding and the length field.

**Padding.** One to eight bytes of padding is added to the packet to make the attack on the security provision more difficult.

**Type.** This one-byte field defines the type of the packet used by SSH protocols.

**Data.** This field is of variable length. The length of the data can be found by deducting the five bytes from the value of the length field.

**CRC.** The cyclic redundancy check filed is used for error detection