Understanding TCP/IP

TCP/IP, the ubiquitous network protocol, is actually a four-layer suite of protocols and is well worth gaining an understanding of. The third instalment of our four-part article.

By Julian Moss

In the previous instalment of this article [*PCNA 88, File C04100*] we looked at the transport layer protocols of the TCP / IP suite: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). We saw that UDP is an unreliable, connectionless protocol suitable for transferring small amounts of data and for broadcast and multicast applications, and we saw that TCP implements reliability mechanisms and requires clients to establish a connection with a server before data can be transferred. This month we will examine some of the application-layer protocols, how they work, and how they exploit the characteristics of UDP and TCP.

Time

A network time service is one of the simplest possible Internet applications. It tells you the time as a 32-bit value, giving the number of seconds that have elapsed since midnight on 1st January 1900.

Time servers use the well-known port number 37. When your time client opens UDP port 37 on the server, the server responds by sending the four bytes of time information.

For such a simple transaction UDP is perfectly adequate, though as it happens many time servers do support connections using TCP as well. TCP's built in reliability is of little use in this application, because by the time the protocol decides that the message may have been lost and re-sends it, the information it contained will be out of date. UDP is the most suitable protocol for real-time applications like this, and others like audio, video and network gaming.

SNMP

A slightly more complex UDP application is Simple Network Management Protocol (SNMP). It allows applications to glean information about how various elements of the network are performing, and to control the network by means of commands sent over it rather than by physical configuration of equipment.

In SNMP there are two distinct components, the SNMP manager and SNMP agents. A manager can communicate with many agents. Typically, the SNMP manager would be an application running on the network manager's console, and agents will run on user workstations, in hubs, routers and other pieces of network hardware.

All communication is between the manager and an agent. Agents don't communicate with each other. Communication may be infrequent and sporadic, and the amount of information exchanged small. Usually a command sent by the manager will generate just a single response.

SNMP uses UDP. This avoids the overhead of having to maintain connections between the SNMP manager and each agent. Because the communication protocol consists essentially of a request for data and a reply containing the data requested, UDP's lack of reliability is not a problem. Reliability is easily implemented within the SNMP manager by re-sending a request if no response is received within a certain period.

The main function of SNMP is to allow the manager to get information from tables maintained by the agents. The tables are known as the Management Information Base (MIB). The MIB is divided into groups, each containing information about a different aspect of the network. Examples of the information that the MIB may contain include the name, type and speed of a network interface, a component's physical location and the contact person for it, and statistics such as the number of packets sent and the number that were undeliverable.

"A network time service is one of the simplest possible Internet applications. It tells you the time as a 32-bit value, giving the number of seconds that have elapsed since midnight on 1st January 1900."

Object IDs

Data is addressed using object IDs. These are written as sequences of numbers separated by periods, rather like long IP addresses. Each number going from left to right represents a node in a tree structure, with related information being grouped in one branch of the tree. There are standardised object IDs for commonly used items of information, and also a section for vendor-specific information. The assignment of object IDs is controlled by the Internet Assigned Numbers Authority (IANA).

"Most SNMP messages have a fixed format. In a typical transaction, an SNMP manager will send a UDP datagram to port 161 on a host running an SNMP agent."

Most SNMP messages have a fixed format. In a typical transaction, an SNMP manager will send a UDP datagram to port 161 on a host running an SNMP agent. The datagram has fields for the type of message (in this case a get-request message), the transaction ID (which will be echoed in the response so that the manager can match up requests with the data received), and a list of object ID / value pairs. In the get-request message the object IDs specify the information requested and the value fields are empty.

The agent will respond with a datagram in which the message type field is get-response. An error status field will indicate whether the request has been fulfilled, or whether an error such as a request for a non-existent object ID occurred. The same list of object ID / value pairs as in the get-request message will be returned, but with the value fields filled in.

There are five types of message in SNMP version 1. Apart from get-request and get-response there is set-request, used by the SNMP manager to initialise a value, and get-next-request. The latter is a bit like listing a directory with a wildcard file spec, in that it returns a list of all the available object IDs in a particular group.

The fifth message type, trap, is used by SNMP agents to signal events to the SNMP manager. These messages are sent to UDP port 162. Trap messages have a format of their own. This includes a trap type field which indicates the type of event being signalled: for example, the agent initialising itself or the network device being turned off. There is a vendor-specific trap type which allows vendors to define traps for events of their own choosing.

Message Types

One problem with SNMP version 1 is that the maximum size of a message is 512 bytes. This limit was chosen so that the UDP datagram in which it is sent falls within the limit (576 bytes) that all TCP / IP transports are guaranteed to pass. The error status value will indicate if the information requested is too big. Typically, this can occur when asking for text-based information, which is returned as strings of up to 255 characters in length.

SNMP version 2 adds two new message types. Get-bulk-request provides a way to retrieve larger amounts of data than version 1 can handle, and inform-request allows SNMP managers to communicate with one another. SNMP 2 also adds security features which can be used to help ensure that information is passed only to agents authorised to receive it.

Telnet

Telnet is a terminal emulation application that enables a workstation to connect to a host using a TCP / IP link and interact with it as if it was a directly connected terminal. It is a client / server application. The server runs on a host on which applications are running, and passes information between the applications and the Telnet clients. The well-known port number for Telnet servers is TCP port 23.

Telnet clients must convert the user data between the form in which it is transmitted and the form in which it is displayed. This is the difficult part of the application, the terminal emulation, and has little to do with the Telnet protocol itself. Telnet protocol commands are principally used to allow the client and server to negotiate the display options, because Telnet clients and servers don't make assumptions about each other's capabilities.

TCP provides the reliability for Telnet, so neither the client nor the server need be concerned about re-sending data that is lost, nor about error checking. This makes the Telnet protocol very simple. There is no special format for TCP segments that contain commands - they simply form part of the data stream.

Data is sent, usually as 7-bit ASCII, in TCP packets (which you may recall are called segments). A byte value of 255, "interpret as command" (IAC), means that the bytes which follow are to be treated as Telnet commands and not user data. This is immediately followed by a byte that identifies the command itself, and then a value. Many commands are fixed length, so the byte after that, if not another IAC, would be treated as user data. To send the byte 255 as data, two consecutive bytes of value 255 are used.

Some commands, such as those that include text values, are variable length. These are implemented using the sub-

"Telnet clients must convert the user data between the form in which it is transmitted and the form in which it is displayed. This is the difficult part of the application."

TCP/IP

"Telnet allows you to interact with an application running on a remote computer, but it has no facility for enabling you to copy a file from that computer's hard disk to yours."

option begin (SB) and sub-option end (SE) command bytes. These command bytes enclose the variable length data like parentheses.

The principal Telnet commands used to negotiate the display options when a client connects to a server are WILL (sender wants to enable this option), WONT (sender wants to disable this option), DO (sender wants the receiver to enable this option) and DONT (sender wants the receiver to disable this option).

To see how this works, consider an example. You start your Telnet client, which is configured to emulate a VT 220 terminal, and connect to a host. The client sends WILL <terminal-type> (where <terminal-type> is the byte value representing the terminal type display option) to say that it wants to control what terminal type to use. The server will respond with DO <terminal-type> to show that it is happy for the client to control this option.

Next the server will send SB <terminal-type> <send> SE. This is an invitation to the client to tell the server what its terminal type is: <send> is a byte that means "send the information". The client responds with SB <terminal-type> <is> VT 220 SE (<is> is a byte that indicates that the requested information follows) and so the server is informed of the terminal emulation that the client will be using.

Client and server will negotiate various other options at the start of a connection. Certain options may also be changed during the Telnet session. The echo option determines whether or not characters that are sent by the client are echoed on the display, and by which end. If characters that are typed at the terminal are to be echoed back by the host application the Telnet server will send WILL <echo> to the client, which will agree to this by sending DO <echo>. This option can be changed during a session to suppress the display of password characters.

Another Telnet option to be negotiated is the transmission mode. The usual mode is character-at-a-time mode, where each character typed at the terminal is echoed back by the server unless the host application specifically turns echoing off. You can tell when character-at-a-time mode is being used because there is a delay between a key being pressed and a character appearing in the terminal window.

The main alternative to character-at-a-time mode is line mode. In this mode, the client displays the characters typed and provides line editing capabilities for the user. Only completed lines are sent to the server. Line mode is used by some mainframe terminal emulations. Again, it is possible to switch modes during a Telnet session if it is required to interact with an application running on the host that responds to single keystrokes rather than whole lines of input.

The urgent flag and urgent pointer in a TCP segment come into use when a Telnet terminal user presses the Break key to interrupt a process on the host. Break is converted by the Telnet client into two Telnet commands which are sent to the server: IP (interrupt process) followed by DO <timing mark> (again, we use angle brackets to indicate a byte representing an option). The server responds to the latter with WILL <timing mark> followed by a DM (data mark) command. The urgent pointer is set to point to the DM command byte, so even if flow control has halted the transmission of normal data this command will still be received. Data mark is a synchronisation marker which causes any queued data up to that point to be discarded.

Most of the data that passes between client and server during a Telnet session is user input and application data. The important thing to realise is that Telnet does not package up this data with additional headers or control information: it is simply passed directly to TCP. One side effect of this is that you can use a Telnet client to talk to other TCP applications that use ASCII-based protocols simply by connecting to the appropriate port. Though it might not normally be sensible to do this, it can be a useful troubleshooting tool.

Finger

Finger is a simple example of a TCP/IP application that uses an ASCII-based protocol. A Finger server is

"The well-known Finger port is TCP port 79. A Finger client opens this port and then sends a request, which is either a null string or a user name. The server responds by sending some text and closing the connection."

a program that supplies information to a requesting client. The information supplied usually relates to the user accounts on a host, though many ISPs use Finger servers to provide status information.

The well-known Finger port is TCP port 79. A Finger client opens this port and then sends a request, which is either a null string or a user name. The server responds by sending some text and closing the connection. If a null string was sent you may receive information about all users known to the system; a user name will return information about that specific user.

For security reasons many organisations do not run Finger servers, or have them reply with a standard message whatever the request. From our perspective the point of interest is that the protocol is pure ASCII text, as you can verify by connecting to a Finger server using a Telnet client.

File Transfer Protocol

Telnet allows you to interact with an application running on a remote computer, but it has no facility for enabling you to copy a file from that computer's hard disk to yours, nor for you to upload files to the remote system. That function is carried out using File Transfer Protocol (FTP).

The FTP specification caters for several different file types, structures and transfer modes, but in practice FTP implementations recognise either text files or binary files. Text files are converted from their native format to 7-bit ASCII with each line terminated by a carriage-return, line-feed pair for transmission. They are converted back to the native text file format by the FTP client. FTP therefore provides a cross-platform transfer mechanism for text files. Binary files are transmitted exactly as-is.

Data is transferred as a continuous stream of bytes. The TCP transport protocol provides all the reliability, making sure that data that is lost is re-sent and checking that it is received correctly. It is worth noting that error detection uses a simple 16-bit checksum so the probability of undetected errors is high compared to a file transfer protocol like Zmodem which uses a 32-bit CRC.

FTP is unusual compared to other TCP applications in that it uses two TCP connections. A control connection is made to the well-known FTP port number 21, and this is used to send FTP commands and receive replies. A separate data connection is established whenever a file or other information is to be transferred, and closed when the data transfer has finished. Keeping data and commands separate makes life easier for the client software, and means that the control connection is always free to send an ABOR (abort) command to terminate a lengthy data transfer.

FTP commands are sent in plain 7-bit ASCII, and consist of a command of up to 4 characters followed by zero or more parameters (those familiar with text mode FTP clients like that supplied with Microsoft TCP/IP may find it curious that FTP commands are not the same as the commands given to the FTP client). The replies consist of a three digit number followed by an optional text explanation, for example, “250 CWD command successful”. The numbers are for easy interpretation by FTP client software, the explanations are for the benefit of the user.

It is instructive to see what happens during a simple FTP session. When you connect to the FTP server (TCP port 21) it sends its welcome message prefixed by the numeric code 220. The FTP client prompts you for your username, which it then sends using the FTP command “USER username”. The server may respond with “331 Need password for username”. The client detects this, prompts you for the password and sends this to the server using the command “PASS password”. If the password is correct the client will receive the response “230 Access granted”.

The next thing you might do is type DIR, to list the current directory on the server. This command to the client results in two FTP commands being issued to the server. The first, “PORT x,x,x,x,y1,y2” tells the server the IP address (x.x.x.x) and port number (y1 * 256 + y2) to use for the data connection. The port number is one in the range 1024 to 4999, a range used for ephemeral connections (those that are used briefly for some specific purpose). The second, LIST, causes the server to open the specified port, send the directory list, and close it again.

The sequence for downloading a file is very similar to that for obtaining a directory list. First, a PORT command is used to specify the data connection port, and then the command “RETR filename” is sent to specify the file to be retrieved. The server opens the data port and sends the data, which the client writes to the hard disk. The server closes the TCP connection to the data port when the file transfer has finished, which is the signal to the client to close the newly-created file.

Conclusion

Since you are unlikely to be asked to write your own client or server there is little to be gained from looking at these application protocols in more detail. However, it is hoped that some useful insights into the working of Internet applications can be gained from these brief descriptions of how a few of them work.

Perhaps the most striking thing about the protocols that use TCP is how simple they are. Because the lower protocol levels take care of reliability, routing and physical transfer matters, the application protocol need concern itself only with things relating to the application. This, of course, is the whole point of using a layered protocol stack.

Click here for the final part of this article

PCNA

The Author

Julian Moss is a freelance writer and software developer. The URL of his Web site is http://www.jm-tech.com/.

Recent Reviews from Tech Support Alert

Reviews of the Best Windows Backup Software
In this detailed comparative review, we checked out eighteen backup software utilities designed for home or SOHO use. Many of the products reviewed were disappointing. However 6 products passed our tests with flying colors and 2 of these were so impressive, they were awarded our “Editor’s Choice.”

Suppliers of Cheap Inkjet Printer Cartridges Reviewed and Rated
With hundreds of companies all claiming to have the *“cheapest and best inkjet printer cartridges*,” our editors decided to put their claims to the test. Not unexpectedly, many suppliers flunked but we did manage to come up with a number of web sites that sell good quality inkjet printer cartridges at heavily discounted prices.

The Best Anti Trojan Software
Our editors took a close look at the 6 leading anti-trojan/trojan remover software utilities. Unfortunately, they found only 2 products that were effective in their ability to detect and remove dangerous modern polymorphic and process injecting trojans.

The 46 Best Ever Freeware Utilities
This is our Editor, Ian “Gizmo” Richards, personal selection of the best freeware utilities. He’s hunted down some real gems, many of which perform better than expensive commercial products.