CHAPTER 22
DISTRIBUTED APPLICATIONS

ANSWERS TO QUESTIONS

22.1 RFC 821 defines SMTP which is the protocol for exchanging email messages. RFC 822 describes the format of those messages.

22.2 The Simple Mail Transfer Protocol (SMTP) is the standard protocol in the TCP/IP protocol suite for transferring mail between hosts. Multipurpose Internet Mail Extension (MIME) is an extension of SMTP to address some of its problems and limitations. MIME addresses SMTP’s inability to directly transmit executables and other binary files and provides support for national language characters that SMTP does not directly support.

22.3 Content-Type: Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to present the data to the user or otherwise deal with the data in an appropriate manner. Transfer-Encoding: Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.

22.4 R64 converts a raw 8-bit binary stream to a stream of printable ASCII characters. Each group of three octets of binary data is mapped into four ASCII characters.

ANSWERS TO PROBLEMS

22.3 Six round trips: The HELO command, MAIL, RCPT, DATA, body of the message, and QUIT. Source: [STEV94]
22.4 Consider the first five network round trips from Problem 22.3. Each is a small command (probably a single segment) that places little load on the network. If all five make it through to the server without retransmission, the congestion window could be six segments when the body is sent. If the body is large, the client could send the first six segment at once, which the network might not be able to handle. Source: [STEV94]
23.1 The Domain Name System (DNS) is a directory lookup service that provides a mapping between the name of a host on the Internet and its numerical address.

23.2 **Name servers:** These are server programs that hold information about a portion of the domain name tree structure and the associated RRs. **Resolvers:** These are programs that extract information from name servers in response to client requests. A typical client request is for an IP address corresponding to a given domain name.

23.3 DNS is based on a hierarchical database containing **resource records (RRs)** that include the name, IP address, and other information about hosts.

23.4 DNS operation typically includes the following steps (Figure 23.3):

1. A user program requests for an IP address for a domain name.
2. A resolver module in the local host or local ISP formulates a query for a local name server in the same domain as the resolver.
3. The local name server checks to see if the name is in its local database or cache, and, if so, returns the IP address to the requestor. Otherwise, the name server queries other available name servers, starting down from the root of the DNS tree or as high up the tree as possible.
4. When a response is received at the local name server, it stores the name/address mapping in its local cache and may maintain this entry for the amount of time specified in the time to live field of the retrieved RR.
5. The user program is given the IP address or an error message.
23.5 A **domain** refers to a group of networks that are under the administrative control of a single entity, such as a company or government agency. Domains are organized hierarchically, so that a given domain may consist of a number of subordinate domains. Names are assigned to domains and reflect this hierarchical organization. Each name server is configured with a subset of the domain name space, known as a **zone**, which is a collection of one or more (or all) subdomains within a domain, along with the associated RRs.

23.6 There are two methods by which queries are forwarded and results returned. Suppose a name server (A) forwards a DNS request to another name server (B). If B has the name/address in its local cache or local database, it can return the IP address to A. If not, then B can do either of the following:

1. Query another name server for the desired result and then send the result back to A. This is known as a **recursive** technique.

## ANSWERS TO PROBLEMS

23.2 Every UDP datagram has an associated length. A process that receives a UDP datagram is told what its length is. When a resolver issues a query using TCP instead of UDP, what is received is a stream of bytes without any record markers. Thus there is no obvious way for the application to know how much data is returned. Also there is no length field in the DNS header. The solution is provided in RFC 1035, Section 4.2.2, which specifies that a 2-byte length value precede the actual DNS message. Source: [STEV94]

23.12 Most resolvers sort the resource records returned by nameservers so that addresses on the same network that the host issuing the query appear first in the list returned to the application that called the resolver library function, on the premise that they will lend to a more efficient communication.
3.1 An NVT is an imaginary device with a well-defined set of characteristics. Using the VTP, a connection is set up between a terminal user and a remote host. Both sides generate data and control signals in their native language. Each side translates its native data and control signals into those of the NVT and translates incoming NVT traffic into its native data and control signals.

3.2 Connection management: Includes connection request and termination. For Telnet, TCP is used to set up a connection. Negotiation: Used to determine a mutually agreeable set of characteristics between the two correspondents. Whereas the NVT has a wide range of capabilities and features, a particular real terminal may be more limited. Furthermore, the NVT has a number of options, such as line length, to be negotiated. Control: Exchange of control information and commands (e.g., end of line, interrupt process). Data: Transfer of data between two correspondents.

3.3 Telnet can be used between two terminals, two processes, or a terminal and a process, using a TCP connection. If the communicating entity is a process, a server Telnet module is needed to convert between the NVT representation and the process representation. If the communicating entity is a terminal, a user Telnet module is needed to map the characteristics of the terminal into those of the NVT.
3.4 The Telnet synch signal is designed to allow the user to communicate some urgent command to a server process, such as an Interrupt Process (IP) or Abort Output (AO) command. The Telnet synch signal consists of the Telnet command Data Mark (DM) transmitted in a TCP segment with TCP urgent notification. To send an urgent command, such as AO, or IP, Telnet sends the urgent command followed by the DM sequence (IAC DM) as urgent data. When the destination Telnet receives a TCP urgent notification, it should immediately scan the data stream for Telnet commands as normal, but discard all data. Telnet commands are handled normally. This continues until the DM is found, when processing returns to normal.

3.5 The first category includes options that change, enhance, or refine the characteristics of the NVT. The first category also includes options that define a new virtual terminal to replace the NVT. Options in the second category change the transfer protocol. Options that define new commands or control features of the transfer protocol are also included in the second category. The third category of options allows other information that is not part of the user data or the transfer protocol to be defined and passed over the connection.

3.6 The interpretation of these commands depends on the context of the negotiation, as illustrated in Figure 3.4.

3.7 FTP involves a User FTP entity and a Server FTP entity. The host that initiates the transfer is the user. The user chooses the name of the file and the options to be used in the transfer. The server accepts or rejects the transfer request based on its file system protection criteria and on the options requested. If the transfer request is accepted, the server is responsible for establishing and managing the transfer.

3.8 There are four defined data types: ASCII, EBCDIC, image, and logical byte size.
3.9 **File structure:** This type assumes that the file is simply a string of bytes (defined by the data type option) that terminates in an end of file marker. **Record structure:** This type is used when it is more convenient to treat the file as a sequence of records. The record structure type causes transmission of individual records, separated by the standard End of Record marker for the specified data type. **Page structure:** This type is used for files that are not stored in a contiguous fashion on the disk and in which this page structure needs to be preserved across the transfer.

3.10 The **stream mode** is the simplest and is the default mode. In stream mode, the unmodified, raw data are sent over the data connection. The **block mode** provides for restarting a failed or interrupted transfer. If a fault or interrupt occurs, a transfer can be picked up where it left off, rather than having to retransmit an entire file. When this mode is accepted, the source encapsulates the data for transfer into blocks. The **compressed mode** provides a way to improve efficiency of the transfer by allowing the source to squeeze sequences of the same character into a shorter coded sequence.

3.11 The **restart marker** in a block indicates that this block consists of a string of printable characters used by the source to mark a checkpoint in the data stream. The receiver marks the corresponding position in the data stream and returns this information to the sender. The source and destination may, if interrupted, restart the transfer from the last correctly received restart marker. The specification does not establish any specific policy for how checkpointing and restart operations should be conducted.
ANSWERS TO PROBLEMS

3.1 This problem is covered in RFC 1143. An example is used between "us" and "him", in which both sides know that the option is ON.

On his side:
1 He decides to disable. He sends DONT and disables the option.
2 He decides to reenable. He sends DO and remembers he is negotiating.
3 He receives WONT and gives up on negotiation.
4 He decides to try once again to reenable. He sends DO and remembers he is negotiating.
5 He receives WONT and gives up on negotiation. For whatever reason, he decides to agree with future requests.
6 He receives WILL and agrees. He responds DO and enables the option.
7 He receives DONT and sighs. He responds WONT and disables the option.
(repeat 10 and then 11, forever)

On our side:
3 We receive DONT and sigh. We respond WONT and disable the option.
4 We receive DO but disagree. We respond WONT.
5 We decide to disable. We send WONT and disable the option. For whatever reason, we decide to agree with future requests.
6 We receive DO and agree. We send WILL and enable the option.
7 We receive DONT and sigh. We send WONT and disable the option.
(repeat 12 and then 13, forever)

Both sides have followed RFC 854; but we end in an option negotiation loop, as DONT DO DO and then DO DONT forever travel through the network one way, and WONT WONT followed by WILL WONT forever travel through the network the other way. The behavior in steps 1 and 9 is responsible for this loop.
3.2 This case is also covered in RFC 1143, which states: The other side agrees with DO DONT DO. We receive the first DO, enable the option, and forget we have negotiated. Now DONT DO is coming through the network and both sides have forgotten they are negotiating; consequently we loop.