UNIT V

Simple Mail Transfer Protocol (SMTP) is an Internet standard for electronic mail (e-mail) transmission across Internet Protocol (IP) networks.It includes the extended SMTP (ESMTP) additions, and is the protocol in widespread use today. SMTP is specified for outgoing mail transport and uses TCP port 25.

While electronic mail servers and other mail transfer agents use SMTP to send and receive mail messages, user-level client mail applications typically only use SMTP for sending messages to a mail server for relaying. For receiving messages, client applications usually use either the Post Office Protocol (POP) or the Internet Message Access Protocol (IMAP) or a proprietary system (such as Microsoft Exchange or Lotus Notes/Domino) to access their mail box accounts on a mail server.

Email is submitted by a mail client (MUA, message user agent) to a mail server (MSA, message submission agent) using SMTP on TCP port 587. Most mailbox providers still allow submission on traditional port 25. From there, the MSA delivers the mail to its MTA. Often, these two agents are just different instances of the same software launched with different options on the same machine. Local processing can be done either on a single machine, or split among various appliances; in the former case, involved processes can share files; in the latter case, SMTP is used to transfer the message internally, with each host configured to use the next appliance as a smart host. Each process is an MTA in its own right; that is, an SMTP server.

The boundary MTA has to locate the target host. It uses the in the Domain name system (DNS) to look up the the mail exchanger record (MX record) for the recipient's domain (the part of the address on the right of @). The returned MX record contains the name of the target host. The MTA next looks up the A record for that name in order to get the IP address and connect to such host as an SMTP client. (The

article on MX record discusses many factors in determining which server the sending MTA connects to.)

Once the MX target accepts the incoming message, it hands it to a mail delivery agent (MDA) for local mail delivery. An MDA is able to save messages in the relevant mailbox format]]. Again, mail reception can be done using many computers or just one —the picture displays two nearby boxes in either case. An MDA may deliver messages directly to storage, or forward them over a network using SMTP, or any other means, including the Local Mail Transfer Protocol (LMTP), a derivative of SMTP designed for this purpose.

Once delivered to the local mail server, the mail is stored for batch retrieval by authenticated mail clients (MUAs). Mail is retrieved by end-user applications, called email clients, using Internet Message Access Protocol (IMAP), a protocol that both facilitates access to mail and manages stored mail, or the Post Office Protocol (POP) which typically uses the traditional mbox mail file format or a proprietary system such as Microsoft Exchange/Outlook or Lotus Notes/Domino. Webmail clients may use either method, but the retrieval protocol is often not a formal standard.

SMTP defines message *transport*, not the message *content*. Thus, it defines the mail *envelope* and its parameters, such as the envelope sender, but not the header or the body of the message itself. STD 10 and RFC 5321 define SMTP (the envelope), while STD 11 and RFC 5322 define the message (header and body), formally referred to as the Internet Message Format.

SMTP Protocols

SMTP is a text-based protocol, in which a mail sender communicates with a mail receiver by issuing command strings and supplying necessary data over a reliable ordered data stream channel, typically a Transmission Control Protocol (TCP)

connection. An *SMTP session* consists of commands originated by an SMTP client (the initiating agent, sender, or transmitter) and corresponding responses from the SMTP server (the listening agent, or receiver) so that the session is opened, and session parameters are exchanged. A session may include zero or more SMTP transactions. An *SMTP transaction* consists of three command/reply sequences (see example below.) They are:

1. **MAIL** command, to establish the return address, a.k.a. Return-Path, 5321.From, mfrom, or envelope sender. This is the address for bounce messages.
2. **RCPT** command, to establish a recipient of this message. This command can be issued multiple times, one for each recipient. These addresses are also part of the envelope.
3. **DATA** to send the *message text*. This is the content of the message, as opposed to its envelope. It consists of a *message header* and a *message body* separated by an empty line. DATA is actually a group of commands, and the server replies twice: once to the *DATA command* proper, to acknowledge that it is ready to receive the text, and the second time after the end-of-data sequence, to either accept or reject the entire message.

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of e-mail to support:

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets

MIME defines mechanisms for sending other kinds of information in e-mail. These include text in languages other than English using character encodings other than ASCII, and 8-bit binary content such as files containing images, sounds, movies, and computer programs. MIME is also a fundamental component of communication protocols such as HTTP, which requires that data be transmitted in the context of e-mail-like messages even though the data might not (and usually doesn't) actually have anything to do with e-mail. Mapping messages into and out of MIME format is typically done automatically by an e-mail client or by mail servers when sending or receiving Internet (SMTP/MIME) e-mail.

The goals of the MIME definition included requiring no changes to existent e-mail servers and allowing plain text e-mail to function in both directions with existing clients. These goals were achieved by using additional RFC 822-style headers for all MIME message attributes and by making the MIME headers optional with default values ensuring a non-MIME message is interpreted correctly by a MIME-capable client. A simple MIME text message is therefore likely to be interpreted correctly by a non-MIME client although if it has e-mail headers the non-MIME client won't know how to interpret. Similarly, if the quoted printable transfer encoding (see below) is used, the ASCII part of the message will be intelligible to users with non-MIME clients

Multipart subtypes

The MIME standard defines various multipart-message subtypes, which specify the nature of the message parts and their relationship to one another. The subtype is specified in the "Content-Type" header of the overall message. For example, a multipart MIME message using the digest subtype would have its Content-Type set as "multipart/digest".

The RFC initially defined 4 subtypes: mixed, digest, alternative and parallel. A minimally compliant application must support mixed and digest; other subtypes are optional. Additional subtypes, such as signed and form-data, have since been separately defined in other RFCs.

IMAP:

The **Internet Message Access Protocol** (**IMAP**) is one of the two most prevalent Internet standard protocols for e-mail retrieval, the other being the Post Office Protocol (POP).Virtually all modern e-mail clients and mail servers support both protocols as a means of transferring e-mail messages from a server.

The Internet Message Access Protocol (commonly known as IMAP, and previously called *Internet Mail Access Protocol*, *Interactive Mail Access Protocol* is an Application Layer Internet protocol that allows an e-mail client to access e-mail on a remote mail server.

IMAP supports both on-line and off-line modes of operation. E-mail clients using IMAP generally leave messages on the server until the user explicitly deletes them. This and other characteristics of IMAP operation allow multiple clients to manage the same mailbox. Most e-mail clients support IMAP in addition to POP to retrieve messages; however, fewer email services support IMAP IMAP offers access to the mail store. Clients may store local copies of the messages, but these are considered to be a temporary cache.

E-mail messages are sent to an e-mail server that stores messages in the recipient's email box. The user retrieves messages with an e-mail client that uses one of a number of e-mail retrieval protocols. Some clients and servers preferentially use vendor-specific, proprietary protocols, but most support the Internet standard protocols,

SMTP for sending e-mail and POP and IMAP for retrieving e-mail, allowing interoperability with other servers and clients.).

The Internet Message Access Protocol (commonly known as IMAP, and previously called *Internet Mail Access Protocol*, *Interactive Mail Access Protocol* is an Application Layer Internet protocol that allows an e-mail client to access e-mail on a remote mail server.IMAP supports both on-line and off-line modes of operation. E-mail clients using IMAP generally leave messages on the server until the user explicitly deletes them. This and other characteristics of IMAP operation allow multiple clients to manage the same mailbox. Most e-mail clients support IMAP in addition to POP to retrieve messages; however, fewer email services support IMAP. IMAP offers access to the mail store. Clients may store local copies of the messages, but these are considered to be a temporary cache.

E-mail messages are sent to an e-mail server that stores messages in the recipient's email box. The user retrieves messages with an e-mail client that uses one of a number of e-mail retrieval protocols. Some clients and servers preferentially use vendor-specific, proprietary protocols, but most support the Internet standard protocols, SMTP for sending e-mail and POP and IMAP for retrieving e-mail, allowing interoperability with other servers and clients

ADVANTAGES OS IMAP:

Connected and disconnected modes of operation

When using POP, clients typically connect to the e-mail server briefly, only as long as it takes to download new messages. When using IMAP4, clients often stay connected as long as the user interface is active and download message content on demand. For users with many or large messages, this IMAP4 usage pattern can result in faster response times.

Multiple clients simultaneously connected to the same mailbox

The POP protocol requires the currently connected client to be the only client connected to the mailbox. In contrast, the IMAP protocol specifically allows simultaneous access by multiple clients and provides mechanisms for clients to detect changes made to the mailbox by other, concurrently connected, clients.

Access to MIME message parts and partial fetch

Usually all Internet e-mail is transmitted in MIME format, allowing messages to have a tree structure where the leaf nodes are any of a variety of single part content types and the non-leaf nodes are any of a variety of multipart types. The IMAP4 protocol allows clients to separately retrieve any of the individual MIME parts and also to retrieve portions of either individual parts or the entire message. These mechanisms allow clients to retrieve the text portion of a message without retrieving attached files or to stream content as it is being fetched.

Message state information

Through the use of flags defined in the IMAP4 protocol, clients can keep track of message state; for example, whether or not the message has been read, replied to, or deleted. These flags are stored on the server, so different clients accessing the same mailbox at different times can detect state changes made by other clients. POP provides no mechanism for clients to store such state information on the server so if a single user accesses a mailbox with two different POP clients, state information—such as whether a message has been accessed—cannot be synchronized between the clients. The IMAP4 protocol supports both pre-defined system flags and client defined keywords. System flags indicate state information such as whether a message has been read. Keywords, which are not supported by all IMAP servers, allow messages to be given one or more tags whose meaning is up to the client. Adding user created

tags to messages is an operation supported by some web-based email services, such as Gmail.

Multiple mailboxes on the server

IMAP4 clients can create, rename, and/or delete mailboxes (usually presented to the user as folders) on the server, and move messages between mailboxes. Multiple mailbox support also allows servers to provide access to shared and public folders.

Server-side searches

IMAP4 provides a mechanism for a client to ask the server to search for messages meeting a variety of criteria. This mechanism avoids requiring clients to download every message in the mailbox in order to perform these searches.

Built-in extension mechanism

Reflecting the experience of earlier Internet protocols, IMAP4 defines an explicit mechanism by which it may be extended. Many extensions to the base protocol have been proposed .

DISADVANTAGES OF IMAP:

While IMAP remedies many of the shortcomings of POP, this inherently introduces additional complexity. Much of this complexity (e.g., multiple clients accessing the same mailbox at the same time) is compensated for by server-side workarounds such as Maildir or database backends.

Unless the mail store and searching algorithms on the server are carefully implemented, a client can potentially consume large amounts of server resources when searching massive mailboxes.

IMAP4 clients need to maintain a TCP/IP connection to the IMAP server in order to be notified of the arrival of new mail. Notification of mail arrival is done through in-band signaling, which contributes to the complexity of client-side IMAP protocol handling. A private proposal, push IMAP, would extend IMAP to implement push e-mail by sending the entire message instead of just a notification. However, push IMAP has not been generally accepted and current IETF work has addressed the problem in other ways (see the Lemonade Profile for more information).

Unlike some proprietary protocols which combine sending and retrieval operations, sending a message and saving a copy in a server-side folder with a base-level IMAP client requires transmitting the message content twice, once to SMTP for delivery and a second time to IMAP to store in a sent mail folder.

POP3

In computing, the **Post Office Protocol** (**POP**) is an application-layer Internet standard protocol used by local e-mail clients to retrieve e-mail from a remote server over a TCP/IP connection. POP and IMAP (Internet Message Access Protocol) are the two most prevalent Internet standard protocols for e-mail retrieval. Virtually all modern e-mail clients and servers support both. The POP protocol has been developed through several versions, with version 3 (POP3) being the current standard. POP3 is used for most mail clients such as gmail and yahoo.

POP supports simple download-and-delete requirements for access to remote mailboxes (termed maildrop in the POP RFC's). Although most POP clients have an option to leave mail on server after download, e-mail clients using POP generally connect, retrieve all messages, store them on the user's PC as new messages, delete them from the server, and then disconnect. Other protocols, notably IMAP, (Internet Message Access Protocol) provide more complete and complex remote access to

typical mailbox operations. Many e-mail clients support POP as well as IMAP to retrieve messages; however, fewer Internet Service Providers (ISPs) support IMAP.

A POP3 server listens on well-known port 110. Encrypted communication for POP3 is either requested after protocol initiation, using the STLS command, if supported, or by POP3S, which connects to the server using Transport Layer Security (TLS) or Secure Sockets Layer (SSL) on well-known TCP port 995 (e.g. Google Gmail).

Available messages to the client are fixed when a POP session opens the maildrop, and are identified by message-number local to that session or, optionally, by a unique identifier assigned to the message by the POP server. This unique identifier is permanent and unique to the maildrop and allows a client to access the same message in different POP sessions. Mail is retrieved and marked for deletion by message-number. When the client exits the session, the mail marked for deletion is removed from the maildrop.

PROTOCOLS USED IN POP3:

STLS

POP uses the Transmission Control Protocol on port number 110. Transmission may be encrypted using Transport Layer Security (TLS) or Secure Sockets Layer (SSL). This is negotiated in the POP3 protocol using the *STLS* command. Some clients and servers, such as Google Gmail, instead use the deprecated alternate-port method, which uses TCP port 995 (POP3S).

XTND XMIT

Irrespective of the mail retrieval protocol, e-mail clients typically use the Message Submission flavor of the Simple Mail Transfer Protocol (SMTP) to send messages.

POP3 contains a protocol extension, known as "XTND XMIT", that allows clients to transmit outbound mail. The Qualcomm qpopper and CommuniGate Pro servers and Eudora clients are examples of systems that optionally utilize the XTND XMIT methods of authenticated client-to-server e-mail transmission.

SDPS

Demon Internet introduced extensions to POP3 that allow multiple accounts per domain, and has become known as *Standard Dial-up POP3 Service* (SDPS) To access each account, the username includes the hostname, as *john@hostname* or *john+hostname*.

Comparison with IMAP

Clients which leave mail on server generally use the UIDL command get the current association of message-numbers to message identified by its unique identifier. The unique identifier is arbitrary, and might be repeated if the mailbox contains identical messages. In contrast, IMAP uses a 32-bit unique identifier (UID) that is assigned to messages in ascending (although not necessarily consecutive) order as they are received. When retrieving new messages, an IMAP client requests the UIDs greater than the highest UID among all previously-retrieved messages, whereas a POP client must fetch the entire UIDL map. For large mailboxes, this can require significant processing.

MIME serves as the standard for attachments and non-ASCII text in e-mail. Although neither POP3 nor SMTP require MIME-formatted e-mail, essentially all Internet e-mail comes MIME-formatted, so POP clients must also understand and use MIME. IMAP, by design, assumes MIME-formatted e-mail.

HTTP:

The **Hypertext Transfer Protocol** (**HTTP**) is an Application Layer protocol for distributed, collaborative, hypermedia information systems

HTTP is a request-response protocol standard for client-server computing. In HTTP, a web browser, for example, acts as a *client*, while an application running on a computer hosting the web site acts as a *server*. The client submits *HTTP requests* to the responding server by sending messages to it. The server, which stores content (or *resources*) such as HTML files and images, or generates such content on the fly, sends messages back to the client in response. These returned messages may contain the content requested by the client or may contain other kinds of response indications. A client is also referred to as a *user agent* (or 'UA'). A web crawler is another example of a common type of client or user agent.

In between the client and server there may be several intermediaries, such as proxies, web caches or gateways. In such a case, the client communicates with the server indirectly, and only converses directly with the first intermediary in the chain. A server may be called the *origin server* to reflect the fact that this is where content ultimately originates from.

HTTP is not constrained in principle to using TCP/IP, although this is its most popular implementation platform. Indeed HTTP can be "implemented on top of any other protocol on the Internet, or on other networks." HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used.[2]

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs)—or, more specifically, Uniform Resource Locators (URLs)—using the `http` or `https` URI schemes.

Its use for retrieving inter-linked resources, called hypertext documents.

HTTP Session:

An HTTP session is a sequence of network request-response transactions. An HTTP client initiates a request. It establishes a Transmission Control Protocol (TCP) connection to a particular port on a host . An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested resource, an error message, or some other information.

REQUEST METHODS:

HTTP defines nine methods (sometimes referred to as "verbs") indicating the desired action to be performed on the identified **resource**. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server.

HEAD

Asks for the response identical to the one that would correspond to a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

GET

Requests a representation of the specified resource. Note that GET should not be used for operations that cause side-effects, such as using it for taking actions in web applications. One reason for this is that GET may be used arbitrarily by robots or crawlers, which should not need to consider the side effects that a request should cause. See safe methods below.

POST

Submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both.

PUT

Uploads a representation of the specified resource.

DELETE

Deletes the specified resource.

TRACE

Echoes back the received request, so that a client can see what (if any) changes or additions have been made by intermediate servers.

OPTIONS

Returns the HTTP methods that the server supports for specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

CONNECT

Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

PATCH

Is used to apply partial modifications to a resource

HTTP servers are required to implement at least the GET and HEAD methods and, whenever possible, also the OPTIONS method.

DNS

The **Domain Name System** (**DNS**) is a hierarchical naming system for computers, services, or any resource connected to the Internet or a private network. It associates various information with domain names assigned to each of the participants. Most importantly, it translates domain names meaningful to humans into the numerical (binary) identifiers associated with networking equipment for the purpose of locating and addressing these devices worldwide. An often-used analogy to explain the Domain Name System is that it serves as the "phone book" for the Internet by translating human-friendly computer hostnames into IP addresses. For example, *www.example.com* translates to *192.0.32.10*.

The Domain Name System makes it possible to assign domain names to groups of Internet users in a meaningful way, independent of each user's physical location. Because of this, World Wide Web (WWW) hyperlinks and Internet contact information can remain consistent and constant even if the current Internet routing arrangements change or the participant uses a mobile device. Internet domain names are easier to remember than IP addresses such as `208.77.188.166` (IPv4) or `2001:db8::1f70:6e8` (IPv6). People take advantage of this when they recite meaningful URLs and e-mail addresses without having to know how the machine will actually locate them.

The Domain Name System distributes the responsibility of assigning domain names and mapping those names to IP addresses by designating authoritative name servers for each domain. Authoritative name servers are assigned to be responsible for their particular domains, and in turn can assign other authoritative name servers for their sub-domains. This mechanism has made the DNS distributed and fault tolerant and has helped avoid the need for a single central register to be continually consulted and updated.

In general, the Domain Name System also stores other types of information, such as the list of mail servers that accept email for a given Internet domain. By providing a worldwide, distributed keyword-based redirection service, the Domain Name System is an essential component of the functionality of the Internet.

Name servers

The Domain Name System is maintained by a distributed database system, which uses the client-server model. The nodes of this database are the name servers. Each domain has at least one authoritative DNS server that publishes information about that domain and the name servers of any domains subordinate to it. The top of the hierarchy is served by the root nameservers, the servers to query when looking up (*resolving*) a top-level domain name (TLD).

Authoritative name server

An *authoritative* name server is a name server that gives answers that have been configured by an original source, for example, the domain administrator or by dynamic DNS methods, in contrast to answers that were obtained via a regular DNS query to another name server. An authoritative-only name server only returns answers to queries about domain names that have been specifically configured by the administrator.

An authoritative name server can either be a *master* server or a *slave* server. A master server is a server that stores the original (*master*) copies of all zone records. A slave server uses an automatic updating mechanism of the DNS protocol in communication with its master to maintain an identical copy of the master records.

Every DNS zone must be assigned a set of authoritative name servers that are installed in NS records in the parent zone.

When domain names are registered with a domain name registrar their installation at the domain registry of a top level domain requires the assignment of a *primary* name server and at least one *secondary* name server. The requirement of multiple name servers aims to make the domain still functional even if one name server becomes inaccessible or inoperable. The designation of a primary name server is solely determined by the priority given to the domain name registrar. For this purpose generally only the fully qualified domain name of the name server is required, unless the servers are contained in the registered domain, in which case the corresponding IP address is needed as well

SNMP:

Simple Network Management Protocol (**SNMP**) is a UDP-based network protocol. It is used mostly in network management systems to monitor network-attached devices for conditions that warrant administrative attention. SNMP is a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF). It consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects

An SNMP-managed network consists of three key components:

- Managed device
- Agent — software which runs on managed devices
- Network management system (NMS) — software which runs on the manager

A *managed device* is a network node that implements an SNMP interface that allows unidirectional (read-only) or bidirectional access to node-specific information. Managed devices exchange node-specific information with the NMSs. Sometimes called network elements, the managed devices can be any type of device, including,

but not limited to, routers, access servers, switches, bridges, hubs, IP telephones, IP video cameras, computer hosts, and printers.

An *agent* is a network-management software module that resides on a managed device. An agent has local knowledge of management information and translates that information to or from an SNMP specific form.

A *network management system* (NMS) executes applications that monitor and control managed devices. NMSs provide the bulk of the processing and memory resources required for network management. One or more NMSs may exist on any managed network.

SNMP itself does not define which information (which variables) a managed system should offer. Rather, SNMP uses an extensible design, where the available information is defined by management information bases (MIBs). MIBs describe the structure of the management data of a device subsystem; they use a hierarchical namespace containing object identifiers (OID). Each OID identifies a variable that can be read or set via SNMP.

Protocol details

SNMP operates in the Application Layer of the Internet Protocol Suite (Layer 7 of the OSI model). The SNMP agent receives requests on UDP port 161. The manager may send requests from any available source port to port 161 in the agent. The agent response will be sent back to the source port on the manager.

All SNMP PDUs are constructed as follows:

IP	UDP	version	commun	PDU-	request-	error-	error-	variable

header header ity type id status index bindings

The seven SNMP protocol data units (PDUs) are as follows:

GetRequest

Retrieve the value of a variable or list of variables. Desired variables are specified in variable bindings (values are not used). Retrieval of the specified variable values is to be done as an atomic operation by the agent. A *Response* with current values is returned.

SetRequest

Change the value of a variable or list of variables. Variable bindings specified in the body of the request. Changes to all specified variables are to be made as an atomic operation by the agent. A *Response* with (current) new values for the variables is returned.

GetNextRequest

Returns a *Response* with variable binding for the lexicographically next variable in the MIB. The entire MIB of an agent can be walked by iterative application of *GetNextRequest* starting at OID 0. Rows of a table can be read by specifying column OIDs in the variable bindings of the request.

GetBulkRequest

Optimized version of *GetNextRequest*. Requests multiple iterations of *GetNextRequest* and returns a *Response* with multiple variable bindings walked from the variable binding or bindings in the request. PDU specific *non-repeaters* and *max-repetitions* fields are used to control response behavior. *GetBulkRequest* was introduced in SNMPv2.

Response

Returns variable bindings and acknowledgement for *GetRequest*, *SetRequest*, *GetNextRequest*, *GetBulkRequest* and *InformRequest*. Error reporting is provided by *error-status* and *error-index* fields. Although it was used as a response to both gets and sets, this PDU was called *GetResponse* in SNMPv1.

Trap

Asynchronous notification from agent to manager. Includes *sysUpTime*, an OID identifying the type of trap and optional variable bindings. Destination addressing for traps is determined in an application specific manner typically through trap configuration variables in the MIB. The format of the trap message was changed in SNMPv2 and the PDU was renamed *SNMPv2-Trap*.

InformRequest

This is acknowledged asynchronous notification from manager to manager. This PDU use the same format as the SNMPv2 version of *Trap*. Manager-to-manager notifications were already possible in SNMPv1 (using a *Trap*), but as SNMP commonly runs over UDP where delivery is not assured and dropped packets are not reported, delivery of a *Trap* was not guaranteed. *InformRequest* fixes this by sending back an acknowledgement on receipt. Receiver replies with *Response* parroting all information in the *InformRequest*. This PDU was introduced in SNMPv2.

TELNET (TELE NETwork) is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communications facility via a virtual terminal connection. User data is interspersed in-band with TELNET control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP).

Telnet was developed in 1969 beginning with RFC 15, extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard STD 8, one of the first Internet standards.

Historically, telnet provided access to a command-line interface (usually, of an operating system) on a remote host. Most network equipment and operating systems with a TCP/IP stack support a Telnet service for remote configuration (including systems based on Windows NT). Because of security issues with Telnet, its use for this purpose has waned in favor of SSH.

The term *telnet* may also refer to the software that implements the client part of the protocol. Telnet client applications are available for virtually all computer platforms. *Telnet* is also used as a verb. *To telnet* means to establish a connection with the Telnet protocol, either with command line client or with a programmatic interface. For example, a common directive might be: "*To change your password, telnet to the server, login and run the passwd command.*" Most often, a user will be *telnetting* to a Unix-like server system or a network device such as a router and obtain a login prompt to a command line text interface or a character-based full-screen manager.

Telnet is a client-server protocol, based on a reliable connection-oriented transport. Typically this protocol is used to establish a connection to Transmission Control Protocol (TCP) port number 23, where a Telnet server application (telnetd) is listening. Telnet, however, predates TCP/IP and was originally run over Network Control Program (NCP) protocols.

Transport Layer Security (**TLS**) and its predecessor, **Secure Socket Layer** (**SSL**), are cryptographic protocols that provide security for communications over networks such as the Internet. TLS and SSL encrypt the segments of network connections at the Application Layer to ensure secure end-to-end transit at the Transport Layer

TELNET Protocols:

The Telnet protocol is often thought of as simply providing a facility for remote logins to computer via the Internet. This was its original purpose although it can be used for many other purposes.

It is best understood in the context of a user with a simple terminal using the local telnet program (known as the client program) to run a login session on a remote computer where his communications needs are handled by a telnet server program. It should be emphasised that the telnet server can pass on the data it has received from the client to many other types of process including a remote login server. It is described in RFC854 and was first published in 1983.

The Network Virtual Terminal:

Communication is established using the TCP/IP protocols and communication is based on a set of facilities known as a **Network Virtual Terminal** (NVT). At the user or client end the telnet client program is responsible for mapping incoming NVT codes to the actual codes needed to operate the user's display device and is also responsible for mapping user generated keyboard sequences into NVT sequences.

The NVT uses 7 bit codes for characters, the display device, referred to as a printer in the RFC, is only required to display the "standard" printing ASCII characters represented by 7 bit codes and to recognise and process certain control codes. The 7 bit characters are transmitted as 8 bit bytes with most significant bit set to zero. An end-of-line is transmitted as the character sequence CR (carriage return) followed by LF (line feed). If it is desired to transmit an actual carriage return this is transmitted as a carriage return followed by a NUL (all bits zero) character.

File Transfer Protocol (**FTP**) is a standard network protocol used to copy a file from one host to another over a TCP/IP-based network, such as the Internet. FTP is built on a client-server architecture and utilizes separate control and data connections between the client and server applications, which solves the problem of different end host configurations (i.e., Operating System, file names) FTP is used with user-based password authentication or with anonymous user access

FTP can be run in *active mode* or *passive mode*, which control how the second connection is opened. In active mode the client sends the server the IP address port number that the client will use for the data connection, and the server opens the connection. Passive mode was devised for use where the client is behind a firewall and unable to accept incoming TCP connections. The server sends the client an IP address and port number and the client opens the connection to the server.

While transferring data over the network, four data representations can be used:

- ASCII mode: used for text. Data is converted, if needed, from the sending host's character representation to "8-bit ASCII" before transmission, and (again, if necessary) to the receiving host's character representation. As a consequence, this mode is inappropriate for files that contain numeric data in binary, floating point or binary coded decimal form.
- Image mode (commonly called Binary mode): the sending machine sends each file byte for byte and as such the recipient stores the bytestream as it receives it. (Image mode support has been recommended for all implementations of FTP).
- EBCDIC mode: use for plain text between hosts using the EBCDIC character set. This mode is otherwise like ASCII mode.
- Local mode: Allows two computers with identical setups to send data in a proprietary format without the need to convert it to ASCII

For text files, different format control and record structure options are provided. These features were designed to facilitate files containing Telnet or ASA formatting.

Data transfer can be done in any of three modes:

- Stream mode: Data is sent as a continuous stream, relieving FTP from doing any processing. Rather, all processing is left up to TCP. No End-of-file indicator is needed, unless the data is divided into records.
- Block mode: FTP breaks the data into several blocks (block header, byte count, and data field) and then passes it on to TCP.
- Compressed mode: Data is compressed using a single algorithm (usually Run-length encoding).

The original FTP specification has many security concerns.

- Bounce Attacks
- Spoof Attacks
- Brute Force Attacks
- Sniffing
- Username Protection
- Port Stealing

FTP Protocols:

The File Transfer Protocol use with the scientific and research network, ARPANET. Access to the ARPANET during this time was limited to a small number of military sites and universities, negating the need for data security or privacy requirements within the protocol.

As the ARPANET gave way to the Internet, data began to traverse increasingly longer paths from client to server. The opportunity for unauthorized third parties to eavesdrop on data transmissions proportionally increased.

The Internet browser company Netscape developed and released the application layer wrapper, Secure Sockets Layer.This protocol enables applications to communicate across a network in a private and secure fashion, discouraging eavesdropping, tampering, and message forgery. While it can add security to any protocol that uses reliable connections (such as TCP), it was most commonly used by Netscape with HTTP to form HTTPS.

The SSL protocol was eventually applied to FTP, with a draft Request for Comments (RFC. An official IANA port was registered shortly thereafter.

Two separate methods were developed to invoke client security for use with FTP clients: *Explicit* or *Implicit*. The *explicit* method is a legacy compatible implementation where FTPS aware clients can invoke security with an FTPS aware server without breaking overall FTP functionality with non-FTPS aware clients. The *implicit* method requires that all clients of the FTPS server be aware that SSL is to be used on the session, and thus is incompatible with non-FTPS-aware clients.

Explicit

In explicit mode (also known as **FTPES**), an FTPS client must "explicitly request" security from an FTPS server and then step-up to a mutually agreed encryption method. If a client does not request security, the FTPS server can either allow the client to continue insecure or refuse/limit the connection.

The mechanism for negotiating authentication and security with FTP was added , which included the new FTP command **AUTH**. While this RFC does not explicitly

define any required security mechanisms (i.e., SSL or TLS), it does require that the FTPS client will challenge the FTPS server with a mutually known mechanism. If the FTPS client challenges the FTPS server with an unknown security mechanism, the FTPS server will respond to the **AUTH** command with error code *504 (not supported)*. Clients could determine which mechanisms were supported by querying the FTPS server with the **FEAT** command, although it should be noted that servers are not necessarily required to be honest in disclosing what levels of security they support. Common methods of invoking FTPS security included: **AUTH TLS** and **AUTH SSL**.FTPS compliance required that clients always negotiate using the **AUTH TLS** method. The RFC also recommends FTPS servers to accept the draft mechanism **AUTH TLS-C**.

Implicit

Negotiation is not allowed with implicit FTPS configurations. A client is immediately expected to challenge the FTPS server with a TLS/SSL **ClientHello** message. If such a message is not received by the FTPS server, the server should drop the connection.

In order to maintain compatibility with existing non-TLS/SSL aware FTP clients, implicit FTPS was expected to listen on the IANA Well . This allowed administrators to retain legacy compatible services on the original 21/TCP FTP control channel.

SECURITY SERVICES:

SSH:

Secure Shell or **SSH** is a network protocol that allows data to be exchanged using a secure channel between two networked devices. Used primarily on Linux and Unix based systems to access shell accounts, SSH was designed as a replacement for Telnet and other insecure remote shells, which send information, notably passwords,

in plaintext, rendering them susceptible to packet analysis. The encryption used by SSH provides confidentiality and integrity of data over an insecure network, such as the Internet

SSH uses public-key cryptography to authenticate the remote computer and allow the remote computer to authenticate the user, if necessary.

SSH is typically used to log into a remote machine and execute commands, but it also supports tunneling, forwarding TCP ports and X11 connections; it can transfer files using the associated SFTP or SCP protocols. SSH uses the client-server model.

The standard TCP port has been assigned for contacting SSH servers.

An SSH client program is typically used for establishing connections to an SSH daemon accepting remote connections. Both are commonly present on most modern operating systems, including Mac OS X, Linux, FreeBSD, Solaris and OpenVMS. Proprietary, freeware and open source versions of various levels of complexity and completeness exist.

The SSH-2 protocol has an internal architecture with well-separated layers. These are:

- The *transport* layer : This layer handles initial key exchange and server authentication and sets up encryption, compression and integrity verification. It exposes to the upper layer an interface for sending and receiving plaintext packets of up to 32,768 bytes each (more can be allowed by the implementation). The transport layer also arranges for key re-exchange, usually after 1 GB of data has been transferred or after 1 hour has passed, whichever is sooner.
- The *user authentication* layer: This layer handles client authentication and provides a number of authentication methods. Authentication is *client-driven*:

when one is prompted for a password, it may be the SSH client prompting, not the server. The server merely responds to client's authentication requests. Widely used user authentication methods include the following:

- *password*: a method for straightforward password authentication, including a facility allowing a password to be changed. This method is not implemented by all programs.
- *publickey*: a method for public key-based authentication, usually supporting at least DSA or RSA keypairs
- *keyboard-interactive* :a versatile method where the server sends one or more prompts to enter information and the client displays them and sends back responses keyed-in by the user. Used to provide one-time password authentication such as S/Key or SecurID. Used by some OpenSSH configurations when PAM is the underlying host authentication provider to effectively provide password authentication, sometimes leading to inability to log in with a client that supports just the plain *password* authentication method.
- GSSAPI authentication methods which provide an extensible scheme to perform SSH authentication using external mechanisms such as Kerberos 5 or NTLM, providing single sign on capability to SSH sessions. These methods are usually implemented by commercial SSH implementations for use in organizations, though OpenSSH does have a working GSSAPI implementation.

- The *connection* layer: This layer defines the concept of channels, channel requests and global requests using which SSH services are provided. A single SSH connection can host multiple channels simultaneously, each transferring data in both directions. Channel requests are used to relay out-of-band channel specific data, such as the changed size of a terminal window or the exit code of

a server-side process. The SSH client requests a server-side port to be forwarded using a global request. Standard channel types include:

  - *shell* for terminal shells, SFTP and exec requests (including SCP transfers)
  - *direct-tcpip* for client-to-server forwarded connections
  - *forwarded-tcpip* for server-to-client forwarded connections
- The SSHFP DNS record provides the public host key fingerprints in order to aid in verifying the authenticity of the host.

This open architecture provides considerable flexibility, allowing SSH to be used for a variety of purposes beyond secure shell. The functionality of the transport layer alone is comparable to Transport Layer Security (TLS); the user authentication layer is highly extensible with custom authentication methods; and the connection layer provides the ability to multiplex many secondary sessions into a single SSH connection, a feature comparable to BEEP and not available in TLS.

Since SSH-1 has inherent design flaws which make it vulnerable (e.g., man-in-the-middle attacks), it is now generally considered obsolete and should be avoided by explicitly disabling fallback to SSH-1. While most modern servers and clients support SSH-2, some organizations still use software with no support for SSH-2, and thus SSH-1 cannot always be avoided.

Security Issues:

In all versions of SSH, it is important to verify unknown public keys before accepting them as valid. Accepting an attacker's public key as a valid public key has the effect of disclosing the transmitted password and allowing man-in-the-middle attacks

Pretty Good Privacy (**PGP**) is a data encryption and decryption computer program that provides cryptographic privacy and authentication for data communication. PGP is often used for signing, encrypting and decrypting e-mails to increase the security of e-mail communications.

PGP and similar products follow the OpenPGP standard for encrypting and decrypting data.

PGP encryption uses a serial combination of hashing, data compression, symmetric-key cryptography, and, finally, public-key cryptography; each step uses one of several supported algorithms. Each public key is bound to a user name and/or an e-mail address. which uses a hierarchical approach based on certificate authority and which was added to PGP implementations later. Current versions of PGP encryption include both options through an automated key management server.

Compatibility

As PGP evolves, PGP systems that support newer features and algorithms are able to create encrypted messages that older PGP systems cannot decrypt, even with a valid private key. Thus, it is essential that partners in PGP communication understand each other's capabilities or at least agree on PGP settings.

Digital signatures

PGP supports message authentication and integrity checking. The latter is used to detect whether a message has been altered since it was completed (the *message integrity* property), and the former to determine whether it was actually sent by the person/entity claimed to be the sender (a *digital signature*). In PGP, these are used by default in conjunction with encryption, but can be applied to plaintext as well. The sender uses PGP to create a digital signature for the message with either the RSA or

DSA signature algorithms. To do so, PGP computes a hash (also called a message digest) from the plaintext, and then creates the digital signature from that hash using the sender's private keys.

Security quality

To the best of publicly available information, there is no known method which will allow a person or group to break PGP encryption by cryptographic or computational means. , cryptographer Bruce Schneier characterized an early version as being "the closest you're likely to get to military-grade encryption." Early versions of PGP have been found to have theoretical vulnerabilities and so current versions are recommended. In addition to protecting data in transit over a network, PGP encryption can also be used to protect data in long-term data storage such as disk files.

The cryptographic security of PGP encryption depends on the assumption that the algorithms used are unbreakable by direct cryptanalysis with current equipment and techniques. For instance, in the original version, the RSA algorithm was used to encrypt session keys; RSA's security depends upon the one-way function nature of mathematical integer factoring. Likewise, the secret key algorithm used in PGP version 2 was IDEA, which might, at some future time, be found to have a previously unsuspected cryptanalytic flaw. Specific instances of current PGP, or IDEA, insecurities—if they exist—are not publicly known. As current versions of PGP have added additional encryption algorithms, the degree of their cryptographic vulnerability varies with the algorithm used. In practice, each of the algorithms in current use is not publicly known to have cryptanalytic weaknesses.

New versions of PGP are released periodically and vulnerabilities that developers are aware of are progressively fixed. Any agency wanting to read PGP messages would probably use easier means than standard cryptanalysis, e.g. rubber-hose cryptanalysis or black-bag cryptanalysis i.e. installing some form of trojan horse or keystroke logging software/hardware on the target computer to capture encrypted keyrings and their passwords. The FBI has already used this attack against PGP in its investigations. However, any such vulnerabilities apply not just to PGP, but to all encryption software.