Network Working Group
Request for Comments: 1123

Internet Engineering Task Force
R. Braden, Editor
October 1989

Requirements for Internet Hosts -- Application and Support

Status of This Memo

This RFC is an official specification for the Internet community. It incorporates by reference, amends, corrects, and supplements the primary protocol standards documents relating to hosts. Distribution of this document is unlimited.

Summary

This RFC is one of a pair that defines and discusses the requirements for Internet host software. This RFC covers the application and support protocols; its companion RFC-1122 covers the communication protocol layers: link layer, IP layer, and transport layer.

Table of Contents

1. INTRODUCTION

This document is one of a pair that defines and discusses the requirements for host system implementations of the Internet protocol suite. This RFC covers the applications layer and support protocols. Its companion RFC, "Requirements for Internet Hosts -- Communications Layers" [INTRO:1] covers the lower layer protocols: transport layer, IP layer, and link layer.

These documents are intended to provide guidance for vendors, implementors, and users of Internet communication software. They represent the consensus of a large body of technical experience and wisdom, contributed by members of the Internet research and vendor communities.

This RFC enumerates standard protocols that a host connected to the Internet must use, and it incorporates by reference the RFCs and other documents describing the current specifications for these protocols. It corrects errors in the referenced documents and adds additional discussion and guidance for an implementor.

For each protocol, this document also contains an explicit set of requirements, recommendations, and options. The reader must understand that the list of requirements in this document is incomplete by itself; the complete set of requirements for an Internet host is primarily defined in the standard protocol specification documents, with the corrections, amendments, and supplements contained in this RFC.

A good-faith implementation of the protocols that was produced after careful reading of the RFC's and with some interaction with the Internet technical community, and that followed good communications software engineering practices, should differ from the requirements of this document in only minor ways. Thus, in many cases, the "requirements" in this RFC are already stated or implied in the standard protocol documents, so that their inclusion here is, in a sense, redundant. However, they were included because some past implementation has made the wrong choice, causing problems of interoperability, performance, and/or robustness.

This document includes discussion and explanation of many of the requirements and recommendations. A simple list of requirements would be dangerous, because:

- Some required features are more important than others, and some features are optional.
- There may be valid reasons why particular vendor products that

are designed for restricted contexts might choose to use different specifications.

However, the specifications of this document must be followed to meet the general goal of arbitrary host interoperation across the diversity and complexity of the Internet system. Although most current implementations fail to meet these requirements in various ways, some minor and some major, this specification is the ideal towards which we need to move.

These requirements are based on the current level of Internet architecture. This document will be updated as required to provide additional clarifications or to include additional information in those areas in which specifications are still evolving.

This introductory section begins with general advice to host software vendors, and then gives some guidance on reading the rest of the document. Section 2 contains general requirements that may be applicable to all application and support protocols. Sections 3, 4, and 5 contain the requirements on protocols for the three major applications: Telnet, file transfer, and electronic mail, respectively. Section 6 covers the support applications: the domain name system, system initialization, and management. Finally, all references will be found in Section 7.

1.1 The Internet Architecture

For a brief introduction to the Internet architecture from a host viewpoint, see Section 1.1 of [INTRO:1]. That section also contains recommended references for general background on the Internet architecture.

1.2 General Considerations

There are two important lessons that vendors of Internet host software have learned and which a new vendor should consider seriously.

1.2.1 Continuing Internet Evolution

The enormous growth of the Internet has revealed problems of management and scaling in a large datagram-based packet communication system. These problems are being addressed, and as a result there will be continuing evolution of the specifications described in this document. These changes will be carefully planned and controlled, since there is extensive participation in this planning by the vendors and by the organizations responsible for operations of the networks.

Development, evolution, and revision are characteristic of computer network protocols today, and this situation will persist for some years. A vendor who develops computer communication software for the Internet protocol suite (or any other protocol suite!) and then fails to maintain and update that software for changing specifications is going to leave a trail of unhappy customers. The Internet is a large communication network, and the users are in constant contact through it. Experience has shown that knowledge of deficiencies in vendor software propagates quickly through the Internet technical community.

1.2.2 Robustness Principle

At every layer of the protocols, there is a general rule whose application can lead to enormous benefits in robustness and interoperability:

> "Be liberal in what you accept, and
> conservative in what you send"

Software should be written to deal with every conceivable error, no matter how unlikely; sooner or later a packet will come in with that particular combination of errors and attributes, and unless the software is prepared, chaos can ensue. In general, it is best to assume that the network is filled with malevolent entities that will send in packets designed to have the worst possible effect. This assumption will lead to suitable protective design, although the most serious problems in the Internet have been caused by unenvisaged mechanisms triggered by low-probability events; mere human malice would never have taken so devious a course!

Adaptability to change must be designed into all levels of Internet host software. As a simple example, consider a protocol specification that contains an enumeration of values for a particular header field -- e.g., a type field, a port number, or an error code; this enumeration must be assumed to be incomplete. Thus, if a protocol specification defines four possible error codes, the software must not break when a fifth code shows up. An undefined code might be logged (see below), but it must not cause a failure.

The second part of the principle is almost as important: software on other hosts may contain deficiencies that make it unwise to exploit legal but obscure protocol features. It is unwise to stray far from the obvious and simple, lest untoward effects result elsewhere. A corollary of this is "watch out

for misbehaving hosts"; host software should be prepared, not just to survive other misbehaving hosts, but also to cooperate to limit the amount of disruption such hosts can cause to the shared communication facility.

1.2.3 Error Logging

The Internet includes a great variety of host and gateway systems, each implementing many protocols and protocol layers, and some of these contain bugs and mis-features in their Internet protocol software. As a result of complexity, diversity, and distribution of function, the diagnosis of user problems is often very difficult.

Problem diagnosis will be aided if host implementations include a carefully designed facility for logging erroneous or "strange" protocol events. It is important to include as much diagnostic information as possible when an error is logged. In particular, it is often useful to record the header(s) of a packet that caused an error. However, care must be taken to ensure that error logging does not consume prohibitive amounts of resources or otherwise interfere with the operation of the host.

There is a tendency for abnormal but harmless protocol events to overflow error logging files; this can be avoided by using a "circular" log, or by enabling logging only while diagnosing a known failure. It may be useful to filter and count duplicate successive messages. One strategy that seems to work well is: (1) always count abnormalities and make such counts accessible through the management protocol (see Section 6.3); and (2) allow the logging of a great variety of events to be selectively enabled. For example, it might useful to be able to "log everything" or to "log everything for host X".

Note that different managements may have differing policies about the amount of error logging that they want normally enabled in a host. Some will say, "if it doesn't hurt me, I don't want to know about it", while others will want to take a more watchful and aggressive attitude about detecting and removing protocol abnormalities.

1.2.4 Configuration

It would be ideal if a host implementation of the Internet protocol suite could be entirely self-configuring. This would allow the whole suite to be implemented in ROM or cast into silicon, it would simplify diskless workstations, and it would

be an immense boon to harried LAN administrators as well as system vendors. We have not reached this ideal; in fact, we are not even close.

At many points in this document, you will find a requirement that a parameter be a configurable option. There are several different reasons behind such requirements. In a few cases, there is current uncertainty or disagreement about the best value, and it may be necessary to update the recommended value in the future. In other cases, the value really depends on external factors -- e.g., the size of the host and the distribution of its communication load, or the speeds and topology of nearby networks -- and self-tuning algorithms are unavailable and may be insufficient. In some cases, configurability is needed because of administrative requirements.

Finally, some configuration options are required to communicate with obsolete or incorrect implementations of the protocols, distributed without sources, that unfortunately persist in many parts of the Internet. To make correct systems coexist with these faulty systems, administrators often have to "mis-configure" the correct systems. This problem will correct itself gradually as the faulty systems are retired, but it cannot be ignored by vendors.

When we say that a parameter must be configurable, we do not intend to require that its value be explicitly read from a configuration file at every boot time. We recommend that implementors set up a default for each parameter, so a configuration file is only necessary to override those defaults that are inappropriate in a particular installation. Thus, the configurability requirement is an assurance that it will be POSSIBLE to override the default when necessary, even in a binary-only or ROM-based product.

This document requires a particular value for such defaults in some cases. The choice of default is a sensitive issue when the configuration item controls the accommodation to existing faulty systems. If the Internet is to converge successfully to complete interoperability, the default values built into implementations must implement the official protocol, not "mis-configurations" to accommodate faulty implementations. Although marketing considerations have led some vendors to choose mis-configuration defaults, we urge vendors to choose defaults that will conform to the standard.

Finally, we note that a vendor needs to provide adequate

documentation on all configuration parameters, their limits and effects.

1.3 Reading this Document

1.3.1 Organization

In general, each major section is organized into the following subsections:

(1) Introduction

(2) Protocol Walk-Through -- considers the protocol specification documents section-by-section, correcting errors, stating requirements that may be ambiguous or ill-defined, and providing further clarification or explanation.

(3) Specific Issues -- discusses protocol design and implementation issues that were not included in the walk-through.

(4) Interfaces -- discusses the service interface to the next higher layer.

(5) Summary -- contains a summary of the requirements of the section.

Under many of the individual topics in this document, there is parenthetical material labeled "DISCUSSION" or "IMPLEMENTATION". This material is intended to give clarification and explanation of the preceding requirements text. It also includes some suggestions on possible future directions or developments. The implementation material contains suggested approaches that an implementor may want to consider.

The summary sections are intended to be guides and indexes to the text, but are necessarily cryptic and incomplete. The summaries should never be used or referenced separately from the complete RFC.

1.3.2 Requirements

In this document, the words that are used to define the significance of each particular requirement are capitalized. These words are:

* "MUST"

  This word or the adjective "REQUIRED" means that the item is an absolute requirement of the specification.

* "SHOULD"

  This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

* "MAY"

  This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be "conditionally compliant".

1.3.3 Terminology

This document uses the following technical terms:

Segment
: A segment is the unit of end-to-end transmission in the TCP protocol. A segment consists of a TCP header followed by application data. A segment is transmitted by encapsulation in an IP datagram.

Message
: This term is used by some application layer protocols (particularly SMTP) for an application data unit.

Datagram
: A [UDP] datagram is the unit of end-to-end transmission in the UDP protocol.

Multihomed
A host is said to be multihomed if it has multiple IP addresses to connected networks.

1.4 Acknowledgments

This document incorporates contributions and comments from a large group of Internet protocol experts, including representatives of university and research labs, vendors, and government agencies. It was assembled primarily by the Host Requirements Working Group of the Internet Engineering Task Force (IETF).

The Editor would especially like to acknowledge the tireless dedication of the following people, who attended many long meetings and generated 3 million bytes of electronic mail over the past 18 months in pursuit of this document: Philip Almquist, Dave Borman (Cray Research), Noel Chiappa, Dave Crocker (DEC), Steve Deering (Stanford), Mike Karels (Berkeley), Phil Karn (Bellcore), John Lekashman (NASA), Charles Lynn (BBN), Keith McCloghrie (TWG), Paul Mockapetris (ISI), Thomas Narten (Purdue), Craig Partridge (BBN), Drew Perkins (CMU), and James Van Bokkelen (FTP Software).

In addition, the following people made major contributions to the effort: Bill Barns (Mitre), Steve Bellovin (AT&T), Mike Brescia (BBN), Ed Cain (DCA), Annette DeSchon (ISI), Martin Gross (DCA), Phill Gross (NRI), Charles Hedrick (Rutgers), Van Jacobson (LBL), John Klensin (MIT), Mark Lottor (SRI), Milo Medin (NASA), Bill Melohn (Sun Microsystems), Greg Minshall (Kinetics), Jeff Mogul (DEC), John Mullen (CMC), Jon Postel (ISI), John Romkey (Epilogue Technology), and Mike StJohns (DCA). The following also made significant contributions to particular areas: Eric Allman (Berkeley), Rob Austein (MIT), Art Berggreen (ACC), Keith Bostic (Berkeley), Vint Cerf (NRI), Wayne Hathaway (NASA), Matt Korn (IBM), Erik Naggum (Naggum Software, Norway), Robert Ullmann (Prime Computer), David Waitzman (BBN), Frank Wancho (USA), Arun Welch (Ohio State), Bill Westfield (Cisco), and Rayan Zachariassen (Toronto).

We are grateful to all, including any contributors who may have been inadvertently omitted from this list.

2. GENERAL ISSUES

This section contains general requirements that may be applicable to all application-layer protocols.

2.1 Host Names and Numbers

The syntax of a legal Internet host name was specified in RFC-952 [DNS:4]. One aspect of host name syntax is hereby changed: the restriction on the first character is relaxed to allow either a letter or a digit. Host software MUST support this more liberal syntax.

Host software MUST handle host names of up to 63 characters and SHOULD handle host names of up to 255 characters.

Whenever a user inputs the identity of an Internet host, it SHOULD be possible to enter either (1) a host domain name or (2) an IP address in dotted-decimal ("#.#.#.#") form. The host SHOULD check the string syntactically for a dotted-decimal number before looking it up in the Domain Name System.

DISCUSSION:

This last requirement is not intended to specify the complete syntactic form for entering a dotted-decimal host number; that is considered to be a user-interface issue. For example, a dotted-decimal number must be enclosed within "[ ]" brackets for SMTP mail (see Section 5.2.17). This notation could be made universal within a host system, simplifying the syntactic checking for a dotted-decimal number.

If a dotted-decimal number can be entered without such identifying delimiters, then a full syntactic check must be made, because a segment of a host domain name is now allowed to begin with a digit and could legally be entirely numeric (see Section 6.1.2.4). However, a valid host name can never have the dotted-decimal form #.#.#.#, since at least the highest-level component label will be alphabetic.

2.2 Using Domain Name Service

Host domain names MUST be translated to IP addresses as described in Section 6.1.

Applications using domain name services MUST be able to cope with soft error conditions. Applications MUST wait a reasonable interval between successive retries due to a soft error, and MUST

allow for the possibility that network problems may deny service for hours or even days.

An application SHOULD NOT rely on the ability to locate a WKS record containing an accurate listing of all services at a particular host address, since the WKS RR type is not often used by Internet sites. To confirm that a service is present, simply attempt to use it.

2.3 Applications on Multihomed hosts

When the remote host is multihomed, the name-to-address translation will return a list of alternative IP addresses. As specified in Section 6.1.3.4, this list should be in order of decreasing preference. Application protocol implementations SHOULD be prepared to try multiple addresses from the list until success is obtained. More specific requirements for SMTP are given in Section 5.3.4.

When the local host is multihomed, a UDP-based request/response application SHOULD send the response with an IP source address that is the same as the specific destination address of the UDP request datagram. The "specific destination address" is defined in the "IP Addressing" section of the companion RFC [INTRO:1].

Similarly, a server application that opens multiple TCP connections to the same client SHOULD use the same local IP address for all.

2.4 Type-of-Service

Applications MUST select appropriate TOS values when they invoke transport layer services, and these values MUST be configurable. Note that a TOS value contains 5 bits, of which only the most-significant 3 bits are currently defined; the other two bits MUST be zero.

DISCUSSION:

As gateway algorithms are developed to implement Type-of-Service, the recommended values for various application protocols may change. In addition, it is likely that particular combinations of users and Internet paths will want non-standard TOS values. For these reasons, the TOS values must be configurable.

See the latest version of the "Assigned Numbers" RFC [INTRO:5] for the recommended TOS values for the major application protocols.

2.5 GENERAL APPLICATION REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
User interfaces:							
Allow host name to begin with digit	2.1	x					
Host names of up to 635 characters	2.1	x					
Host names of up to 255 characters	2.1		x				
Support dotted-decimal host numbers	2.1		x				
Check syntactically for dotted-dec first	2.1		x				
Map domain names per Section 6.1	2.2	x					
Cope with soft DNS errors	2.2	x					
Reasonable interval between retries	2.2	x					
Allow for long outages	2.2	x					
Expect WKS records to be available	2.2				x		
Try multiple addr's for remote multihomed host	2.3		x				
UDP reply src addr is specific dest of request	2.3		x				
Use same IP addr for related TCP connections	2.3		x				
Specify appropriate TOS values	2.4	x					
TOS values configurable	2.4	x					
Unused TOS bits zero	2.4	x					

3. REMOTE LOGIN -- TELNET PROTOCOL

3.1 INTRODUCTION

Telnet is the standard Internet application protocol for remote login. It provides the encoding rules to link a user's keyboard/display on a client ("user") system with a command interpreter on a remote server system. A subset of the Telnet protocol is also incorporated within other application protocols, e.g., FTP and SMTP.

Telnet uses a single TCP connection, and its normal data stream ("Network Virtual Terminal" or "NVT" mode) is 7-bit ASCII with escape sequences to embed control functions. Telnet also allows the negotiation of many optional modes and functions.

The primary Telnet specification is to be found in RFC-854 [TELNET:1], while the options are defined in many other RFCs; see Section 7 for references.

3.2 PROTOCOL WALK-THROUGH

3.2.1 Option Negotiation: RFC-854, pp. 2-3

Every Telnet implementation MUST include option negotiation and subnegotiation machinery [TELNET:2].

A host MUST carefully follow the rules of RFC-854 to avoid option-negotiation loops. A host MUST refuse (i.e, reply WONT/DONT to a DO/WILL) an unsupported option. Option negotiation SHOULD continue to function (even if all requests are refused) throughout the lifetime of a Telnet connection.

If all option negotiations fail, a Telnet implementation MUST default to, and support, an NVT.

DISCUSSION:
Even though more sophisticated "terminals" and supporting option negotiations are becoming the norm, all implementations must be prepared to support an NVT for any user-server communication.

3.2.2 Telnet Go-Ahead Function: RFC-854, p. 5, and RFC-858

On a host that never sends the Telnet command Go Ahead (GA), the Telnet Server MUST attempt to negotiate the Suppress Go Ahead option (i.e., send "WILL Suppress Go Ahead"). A User or Server Telnet MUST always accept negotiation of the Suppress Go

Ahead option.

When it is driving a full-duplex terminal for which GA has no meaning, a User Telnet implementation MAY ignore GA commands.

DISCUSSION:
Half-duplex ("locked-keyboard") line-at-a-time terminals for which the Go-Ahead mechanism was designed have largely disappeared from the scene. It turned out to be difficult to implement sending the Go-Ahead signal in many operating systems, even some systems that support native half-duplex terminals. The difficulty is typically that the Telnet server code does not have access to information about whether the user process is blocked awaiting input from the Telnet connection, i.e., it cannot reliably determine when to send a GA command. Therefore, most Telnet Server hosts do not send GA commands.

The effect of the rules in this section is to allow either end of a Telnet connection to veto the use of GA commands.

There is a class of half-duplex terminals that is still commercially important: "data entry terminals," which interact in a full-screen manner. However, supporting data entry terminals using the Telnet protocol does not require the Go Ahead signal; see Section 3.3.2.

3.2.3 Control Functions: RFC-854, pp. 7-8

The list of Telnet commands has been extended to include EOR (End-of-Record), with code 239 [TELNET:9].

Both User and Server Telnets MAY support the control functions EOR, EC, EL, and Break, and MUST support AO, AYT, DM, IP, NOP, SB, and SE.

A host MUST be able to receive and ignore any Telnet control functions that it does not support.

DISCUSSION:
Note that a Server Telnet is required to support the Telnet IP (Interrupt Process) function, even if the server host has an equivalent in-stream function (e.g., Control-C in many systems). The Telnet IP function may be stronger than an in-stream interrupt command, because of the out-of-band effect of TCP urgent data.

The EOR control function may be used to delimit the

stream. An important application is data entry terminal support (see Section 3.3.2). There was concern that since EOR had not been defined in RFC-854, a host that was not prepared to correctly ignore unknown Telnet commands might crash if it received an EOR. To protect such hosts, the End-of-Record option [TELNET:9] was introduced; however, a properly implemented Telnet program will not require this protection.

3.2.4 Telnet "Synch" Signal: RFC-854, pp. 8-10

When it receives "urgent" TCP data, a User or Server Telnet MUST discard all data except Telnet commands until the DM (and end of urgent) is reached.

When it sends Telnet IP (Interrupt Process), a User Telnet SHOULD follow it by the Telnet "Synch" sequence, i.e., send as TCP urgent data the sequence "IAC IP IAC DM". The TCP urgent pointer points to the DM octet.

When it receives a Telnet IP command, a Server Telnet MAY send a Telnet "Synch" sequence back to the user, to flush the output stream. The choice ought to be consistent with the way the server operating system behaves when a local user interrupts a process.

When it receives a Telnet AO command, a Server Telnet MUST send a Telnet "Synch" sequence back to the user, to flush the output stream.

A User Telnet SHOULD have the capability of flushing output when it sends a Telnet IP; see also Section 3.4.5.

DISCUSSION:

There are three possible ways for a User Telnet to flush the stream of server output data:

(1) Send AO after IP.

This will cause the server host to send a "flush-buffered-output" signal to its operating system. However, the AO may not take effect locally, i.e., stop terminal output at the User Telnet end, until the Server Telnet has received and processed the AO and has sent back a "Synch".

(2) Send DO TIMING-MARK [TELNET:7] after IP, and discard all output locally until a WILL/WONT TIMING-MARK is

received from the Server Telnet.

Since the DO TIMING-MARK will be processed after the IP at the server, the reply to it should be in the right place in the output data stream. However, the TIMING-MARK will not send a "flush buffered output" signal to the server operating system. Whether or not this is needed is dependent upon the server system.

(3) Do both.

The best method is not entirely clear, since it must accommodate a number of existing server hosts that do not follow the Telnet standards in various ways. The safest approach is probably to provide a user-controllable option to select (1), (2), or (3).

3.2.5 NVT Printer and Keyboard: RFC-854, p. 11

In NVT mode, a Telnet SHOULD NOT send characters with the high-order bit 1, and MUST NOT send it as a parity bit. Implementations that pass the high-order bit to applications SHOULD negotiate binary mode (see Section 3.2.6).

DISCUSSION:

Implementors should be aware that a strict reading of RFC-854 allows a client or server expecting NVT ASCII to ignore characters with the high-order bit set. In general, binary mode is expected to be used for transmission of an extended (beyond 7-bit) character set with Telnet.

However, there exist applications that really need an 8-bit NVT mode, which is currently not defined, and these existing applications do set the high-order bit during part or all of the life of a Telnet connection. Note that binary mode is not the same as 8-bit NVT mode, since binary mode turns off end-of-line processing. For this reason, the requirements on the high-order bit are stated as SHOULD, not MUST.

RFC-854 defines a minimal set of properties of a "network virtual terminal" or NVT; this is not meant to preclude additional features in a real terminal. A Telnet connection is fully transparent to all 7-bit ASCII characters, including arbitrary ASCII control characters.

For example, a terminal might support full-screen commands coded as ASCII escape sequences; a Telnet implementation would pass these sequences as uninterpreted data. Thus, an NVT should not be conceived as a terminal type of a highly-restricted device.

3.2.6 Telnet Command Structure: RFC-854, p. 13

Since options may appear at any point in the data stream, a Telnet escape character (known as IAC, with the value 255) to be sent as data MUST be doubled.

3.2.7 Telnet Binary Option: RFC-856

When the Binary option has been successfully negotiated, arbitrary 8-bit characters are allowed. However, the data stream MUST still be scanned for IAC characters, any embedded Telnet commands MUST be obeyed, and data bytes equal to IAC MUST be doubled. Other character processing (e.g., replacing CR by CR NUL or by CR LF) MUST NOT be done. In particular, there is no end-of-line convention (see Section 3.3.1) in binary mode.

DISCUSSION:
The Binary option is normally negotiated in both directions, to change the Telnet connection from NVT mode to "binary mode".

The sequence IAC EOR can be used to delimit blocks of data within a binary-mode Telnet stream.

3.2.8 Telnet Terminal-Type Option: RFC-1091

The Terminal-Type option MUST use the terminal type names officially defined in the Assigned Numbers RFC [INTRO:5], when they are available for the particular terminal. However, the receiver of a Terminal-Type option MUST accept any name.

DISCUSSION:
RFC-1091 [TELNET:10] updates an earlier version of the Terminal-Type option defined in RFC-930. The earlier version allowed a server host capable of supporting multiple terminal types to learn the type of a particular client's terminal, assuming that each physical terminal had an intrinsic type. However, today a "terminal" is often really a terminal emulator program running in a PC, perhaps capable of emulating a range of terminal types. Therefore, RFC-1091 extends the specification to allow a

more general terminal-type negotiation between User and Server Telnets.

3.3 SPECIFIC ISSUES

3.3.1 Telnet End-of-Line Convention

The Telnet protocol defines the sequence CR LF to mean "end-of-line". For terminal input, this corresponds to a command-completion or "end-of-line" key being pressed on a user terminal; on an ASCII terminal, this is the CR key, but it may also be labelled "Return" or "Enter".

When a Server Telnet receives the Telnet end-of-line sequence CR LF as input from a remote terminal, the effect MUST be the same as if the user had pressed the "end-of-line" key on a local terminal. On server hosts that use ASCII, in particular, receipt of the Telnet sequence CR LF must cause the same effect as a local user pressing the CR key on a local terminal. Thus, CR LF and CR NUL MUST have the same effect on an ASCII server host when received as input over a Telnet connection.

A User Telnet MUST be able to send any of the forms: CR LF, CR NUL, and LF. A User Telnet on an ASCII host SHOULD have a user-controllable mode to send either CR LF or CR NUL when the user presses the "end-of-line" key, and CR LF SHOULD be the default.

The Telnet end-of-line sequence CR LF MUST be used to send Telnet data that is not terminal-to-computer (e.g., for Server Telnet sending output, or the Telnet protocol incorporated another application protocol).

DISCUSSION:

To allow interoperability between arbitrary Telnet clients and servers, the Telnet protocol defined a standard representation for a line terminator. Since the ASCII character set includes no explicit end-of-line character, systems have chosen various representations, e.g., CR, LF, and the sequence CR LF. The Telnet protocol chose the CR LF sequence as the standard for network transmission.

Unfortunately, the Telnet protocol specification in RFC-854 [TELNET:1] has turned out to be somewhat ambiguous on what character(s) should be sent from client to server for the "end-of-line" key. The result has been a massive and continuing interoperability headache, made worse by various faulty implementations of both User and Server

Telnets.

Although the Telnet protocol is based on a perfectly symmetric model, in a remote login session the role of the user at a terminal differs from the role of the server host. For example, RFC-854 defines the meaning of CR, LF, and CR LF as output from the server, but does not specify what the User Telnet should send when the user presses the "end-of-line" key on the terminal; this turns out to be the point at issue.

When a user presses the "end-of-line" key, some User Telnet implementations send CR LF, while others send CR NUL (based on a different interpretation of the same sentence in RFC-854). These will be equivalent for a correctly-implemented ASCII server host, as discussed above. For other servers, a mode in the User Telnet is needed.

The existence of User Telnets that send only CR NUL when CR is pressed creates a dilemma for non-ASCII hosts: they can either treat CR NUL as equivalent to CR LF in input, thus precluding the possibility of entering a "bare" CR, or else lose complete interworking.

Suppose a user on host A uses Telnet to log into a server host B, and then execute B's User Telnet program to log into server host C. It is desirable for the Server/User Telnet combination on B to be as transparent as possible, i.e., to appear as if A were connected directly to C. In particular, correct implementation will make B transparent to Telnet end-of-line sequences, except that CR LF may be translated to CR NUL or vice versa.

IMPLEMENTATION:

To understand Telnet end-of-line issues, one must have at least a general model of the relationship of Telnet to the local operating system. The Server Telnet process is typically coupled into the terminal driver software of the operating system as a pseudo-terminal. A Telnet end-of-line sequence received by the Server Telnet must have the same effect as pressing the end-of-line key on a real locally-connected terminal.

Operating systems that support interactive character-at-a-time applications (e.g., editors) typically have two internal modes for their terminal I/O: a formatted mode, in which local conventions for end-of-line and other

formatting rules have been applied to the data stream, and a "raw" mode, in which the application has direct access to every character as it was entered. A Server Telnet must be implemented in such a way that these modes have the same effect for remote as for local terminals. For example, suppose a CR LF or CR NUL is received by the Server Telnet on an ASCII host. In raw mode, a CR character is passed to the application; in formatted mode, the local system's end-of-line convention is used.

3.3.2 Data Entry Terminals

DISCUSSION:

In addition to the line-oriented and character-oriented ASCII terminals for which Telnet was designed, there are several families of video display terminals that are sometimes known as "data entry terminals" or DETs. The IBM 3270 family is a well-known example.

Two Internet protocols have been designed to support generic DETs: SUPDUP [TELNET:16, TELNET:17], and the DET option [TELNET:18, TELNET:19]. The DET option drives a data entry terminal over a Telnet connection using (sub-) negotiation. SUPDUP is a completely separate terminal protocol, which can be entered from Telnet by negotiation. Although both SUPDUP and the DET option have been used successfully in particular environments, neither has gained general acceptance or wide implementation.

A different approach to DET interaction has been developed for supporting the IBM 3270 family through Telnet, although the same approach would be applicable to any DET. The idea is to enter a "native DET" mode, in which the native DET input/output stream is sent as binary data. The Telnet EOR command is used to delimit logical records (e.g., "screens") within this binary stream.

IMPLEMENTATION:

The rules for entering and leaving native DET mode are as follows:

- The Server uses the Terminal-Type option [TELNET:10] to learn that the client is a DET.
- It is conventional, but not required, that both ends negotiate the EOR option [TELNET:9].
- Both ends negotiate the Binary option [TELNET:3] to

enter native DET mode.

- When either end negotiates out of binary mode, the other end does too, and the mode then reverts to normal NVT.

3.3.3 Option Requirements

Every Telnet implementation MUST support the Binary option [TELNET:3] and the Suppress Go Ahead option [TELNET:5], and SHOULD support the Echo [TELNET:4], Status [TELNET:6], End-of-Record [TELNET:9], and Extended Options List [TELNET:8] options.

A User or Server Telnet SHOULD support the Window Size Option [TELNET:12] if the local operating system provides the corresponding capability.

DISCUSSION:
Note that the End-of-Record option only signifies that a Telnet can receive a Telnet EOR without crashing; therefore, every Telnet ought to be willing to accept negotiation of the End-of-Record option. See also the discussion in Section 3.2.3.

3.3.4 Option Initiation

When the Telnet protocol is used in a client/server situation, the server SHOULD initiate negotiation of the terminal interaction mode it expects.

DISCUSSION:
The Telnet protocol was defined to be perfectly symmetrical, but its application is generally asymmetric. Remote login has been known to fail because NEITHER side initiated negotiation of the required non-default terminal modes. It is generally the server that determines the preferred mode, so the server needs to initiate the negotiation; since the negotiation is symmetric, the user can also initiate it.

A client (User Telnet) SHOULD provide a means for users to enable and disable the initiation of option negotiation.

DISCUSSION:
A user sometimes needs to connect to an application service (e.g., FTP or SMTP) that uses Telnet for its

control stream but does not support Telnet options. User Telnet may be used for this purpose if initiation of option negotiation is disabled.

3.3.5 Telnet Linemode Option

DISCUSSION:
An important new Telnet option, LINEMODE [TELNET:12], has been proposed. The LINEMODE option provides a standard way for a User Telnet and a Server Telnet to agree that the client rather than the server will perform terminal character processing. When the client has prepared a complete line of text, it will send it to the server in (usually) one TCP packet. This option will greatly decrease the packet cost of Telnet sessions and will also give much better user response over congested or long-delay networks.

The LINEMODE option allows dynamic switching between local and remote character processing. For example, the Telnet connection will automatically negotiate into single-character mode while a full screen editor is running, and then return to linemode when the editor is finished.

We expect that when this RFC is released, hosts should implement the client side of this option, and may implement the server side of this option. To properly implement the server side, the server needs to be able to tell the local system not to do any input character processing, but to remember its current terminal state and notify the Server Telnet process whenever the state changes. This will allow password echoing and full screen editors to be handled properly, for example.

3.4 TELNET/USER INTERFACE

3.4.1 Character Set Transparency

User Telnet implementations SHOULD be able to send or receive any 7-bit ASCII character. Where possible, any special character interpretations by the user host's operating system SHOULD be bypassed so that these characters can conveniently be sent and received on the connection.

Some character value MUST be reserved as "escape to command mode"; conventionally, doubling this character allows it to be entered as data. The specific character used SHOULD be user selectable.

On binary-mode connections, a User Telnet program MAY provide an escape mechanism for entering arbitrary 8-bit values, if the host operating system doesn't allow them to be entered directly from the keyboard.

IMPLEMENTATION:
The transparency issues are less pressing on servers, but implementors should take care in dealing with issues like: masking off parity bits (sent by an older, non-conforming client) before they reach programs that expect only NVT ASCII, and properly handling programs that request 8-bit data streams.

3.4.2 Telnet Commands

A User Telnet program MUST provide a user the capability of entering any of the Telnet control functions IP, AO, or AYT, and SHOULD provide the capability of entering EC, EL, and Break.

3.4.3 TCP Connection Errors

A User Telnet program SHOULD report to the user any TCP errors that are reported by the transport layer (see "TCP/Application Layer Interface" section in [INTRO:1]).

3.4.4 Non-Default Telnet Contact Port

A User Telnet program SHOULD allow the user to optionally specify a non-standard contact port number at the Server Telnet host.

3.4.5 Flushing Output

A User Telnet program SHOULD provide the user the ability to specify whether or not output should be flushed when an IP is sent; see Section 3.2.4.

For any output flushing scheme that causes the User Telnet to flush output locally until a Telnet signal is received from the Server, there SHOULD be a way for the user to manually restore normal output, in case the Server fails to send the expected signal.

3.5. TELNET REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Option Negotiation	3.2.1	x					
Avoid negotiation loops	3.2.1	x					
Refuse unsupported options	3.2.1	x					
Negotiation OK anytime on connection	3.2.1		x				
Default to NVT	3.2.1	x					
Send official name in Term-Type option	3.2.8	x					
Accept any name in Term-Type option	3.2.8	x					
Implement Binary, Suppress-GA options	3.3.3	x					
Echo, Status, EOL, Ext-Opt-List options	3.3.3		x				
Implement Window-Size option if appropriate	3.3.3		x				
Server initiate mode negotiations	3.3.4		x				
User can enable/disable init negotiations	3.3.4		x				
Go-Aheads							
Non-GA server negotiate SUPPRESS-GA option	3.2.2	x					
User or Server accept SUPPRESS-GA option	3.2.2	x					
User Telnet ignore GA's	3.2.2			x			
Control Functions							
Support SE NOP DM IP AO AYT SB	3.2.3	x					
Support EOR EC EL Break	3.2.3			x			
Ignore unsupported control functions	3.2.3	x					
User, Server discard urgent data up to DM	3.2.4	x					
User Telnet send "Synch" after IP, AO, AYT	3.2.4		x				
Server Telnet reply Synch to IP	3.2.4			x			
Server Telnet reply Synch to AO	3.2.4	x					
User Telnet can flush output when send IP	3.2.4		x				
Encoding							
Send high-order bit in NVT mode	3.2.5				x		
Send high-order bit as parity bit	3.2.5					x	
Negot. BINARY if pass high-ord. bit to applic	3.2.5		x				
Always double IAC data byte	3.2.6	x					

Double IAC data byte in binary mode	3.2.7	x				
Obey Telnet cmds in binary mode	3.2.7	x				
End-of-line, CR NUL in binary mode	3.2.7					x
End-of-Line						
EOL at Server same as local end-of-line	3.3.1	x				
ASCII Server accept CR LF or CR NUL for EOL	3.3.1	x				
User Telnet able to send CR LF, CR NUL, or LF	3.3.1	x				
ASCII user able to select CR LF/CR NUL	3.3.1		x			
User Telnet default mode is CR LF	3.3.1		x			
Non-interactive uses CR LF for EOL	3.3.1	x				
User Telnet interface						
Input & output all 7-bit characters	3.4.1		x			
Bypass local op sys interpretation	3.4.1		x			
Escape character	3.4.1	x				
User-settable escape character	3.4.1		x			
Escape to enter 8-bit values	3.4.1			x		
Can input IP, AO, AYT	3.4.2	x				
Can input EC, EL, Break	3.4.2		x			
Report TCP connection errors to user	3.4.3		x			
Optional non-default contact port	3.4.4		x			
Can spec: output flushed when IP sent	3.4.5		x			
Can manually restore output mode	3.4.5		x			

4. FILE TRANSFER

4.1 FILE TRANSFER PROTOCOL -- FTP

4.1.1 INTRODUCTION

The File Transfer Protocol FTP is the primary Internet standard for file transfer. The current specification is contained in RFC-959 [FTP:1].

FTP uses separate simultaneous TCP connections for control and for data transfer. The FTP protocol includes many features, some of which are not commonly implemented. However, for every feature in FTP, there exists at least one implementation. The minimum implementation defined in RFC-959 was too small, so a somewhat larger minimum implementation is defined here.

Internet users have been unnecessarily burdened for years by deficient FTP implementations. Protocol implementors have suffered from the erroneous opinion that implementing FTP ought to be a small and trivial task. This is wrong, because FTP has a user interface, because it has to deal (correctly) with the whole variety of communication and operating system errors that may occur, and because it has to handle the great diversity of real file systems in the world.

4.1.2. PROTOCOL WALK-THROUGH

4.1.2.1 LOCAL Type: RFC-959 Section 3.1.1.4

An FTP program MUST support TYPE I ("IMAGE" or binary type) as well as TYPE L 8 ("LOCAL" type with logical byte size 8). A machine whose memory is organized into m-bit words, where m is not a multiple of 8, MAY also support TYPE L m.

DISCUSSION:

The command "TYPE L 8" is often required to transfer binary data between a machine whose memory is organized into (e.g.) 36-bit words and a machine with an 8-bit byte organization. For an 8-bit byte machine, TYPE L 8 is equivalent to IMAGE.

"TYPE L m" is sometimes specified to the FTP programs on two m-bit word machines to ensure the correct transfer of a native-mode binary file from one machine to the other. However, this command should have the same effect on these machines as "TYPE I".

4.1.2.2 Telnet Format Control: RFC-959 Section 3.1.1.5.2

A host that makes no distinction between TYPE N and TYPE T SHOULD implement TYPE T to be identical to TYPE N.

DISCUSSION:

This provision should ease interoperation with hosts that do make this distinction.

Many hosts represent text files internally as strings of ASCII characters, using the embedded ASCII format effector characters (LF, BS, FF, ...) to control the format when a file is printed. For such hosts, there is no distinction between "print" files and other files. However, systems that use record structured files typically need a special format for printable files (e.g., ASA carriage control). For the latter hosts, FTP allows a choice of TYPE N or TYPE T.

4.1.2.3 Page Structure: RFC-959 Section 3.1.2.3 and Appendix I

Implementation of page structure is NOT RECOMMENDED in general. However, if a host system does need to implement FTP for "random access" or "holey" files, it MUST use the defined page structure format rather than define a new private FTP format.

4.1.2.4 Data Structure Transformations: RFC-959 Section 3.1.2

An FTP transformation between record-structure and file-structure SHOULD be invertible, to the extent possible while making the result useful on the target host.

DISCUSSION:

RFC-959 required strict invertibility between record-structure and file-structure, but in practice, efficiency and convenience often preclude it. Therefore, the requirement is being relaxed. There are two different objectives for transferring a file: processing it on the target host, or just storage. For storage, strict invertibility is important. For processing, the file created on the target host needs to be in the format expected by application programs on that host.

As an example of the conflict, imagine a record-oriented operating system that requires some data files to have exactly 80 bytes in each record. While STORing

a file on such a host, an FTP Server must be able to pad each line or record to 80 bytes; a later retrieval of such a file cannot be strictly invertible.

4.1.2.5 Data Connection Management: RFC-959 Section 3.3

A User-FTP that uses STREAM mode SHOULD send a PORT command to assign a non-default data port before each transfer command is issued.

DISCUSSION:
This is required because of the long delay after a TCP connection is closed until its socket pair can be reused, to allow multiple transfers during a single FTP session. Sending a port command can avoided if a transfer mode other than stream is used, by leaving the data transfer connection open between transfers.

4.1.2.6 PASV Command: RFC-959 Section 4.1.2

A server-FTP MUST implement the PASV command.

If multiple third-party transfers are to be executed during the same session, a new PASV command MUST be issued before each transfer command, to obtain a unique port pair.

IMPLEMENTATION:
The format of the 227 reply to a PASV command is not well standardized. In particular, an FTP client cannot assume that the parentheses shown on page 40 of RFC-959 will be present (and in fact, Figure 3 on page 43 omits them). Therefore, a User-FTP program that interprets the PASV reply must scan the reply for the first digit of the host and port numbers.

Note that the host number h1,h2,h3,h4 is the IP address of the server host that is sending the reply, and that p1,p2 is a non-default data transfer port that PASV has assigned.

4.1.2.7 LIST and NLST Commands: RFC-959 Section 4.1.3

The data returned by an NLST command MUST contain only a simple list of legal pathnames, such that the server can use them directly as the arguments of subsequent data transfer commands for the individual files.

The data returned by a LIST or NLST command SHOULD use an

implied TYPE AN, unless the current type is EBCDIC, in which case an implied TYPE EN SHOULD be used.

DISCUSSION:
Many FTP clients support macro-commands that will get or put files matching a wildcard specification, using NLST to obtain a list of pathnames. The expansion of "multiple-put" is local to the client, but "multiple-get" requires cooperation by the server.

The implied type for LIST and NLST is designed to provide compatibility with existing User-FTPs, and in particular with multiple-get commands.

4.1.2.8 SITE Command: RFC-959 Section 4.1.3

A Server-FTP SHOULD use the SITE command for non-standard features, rather than invent new private commands or unstandardized extensions to existing commands.

4.1.2.9 STOU Command: RFC-959 Section 4.1.3

The STOU command stores into a uniquely named file. When it receives an STOU command, a Server-FTP MUST return the actual file name in the "125 Transfer Starting" or the "150 Opening Data Connection" message that precedes the transfer (the 250 reply code mentioned in RFC-959 is incorrect). The exact format of these messages is hereby defined to be as follows:

```
125 FILE: pppp
150 FILE: pppp
```

where pppp represents the unique pathname of the file that will be written.

4.1.2.10 Telnet End-of-line Code: RFC-959, Page 34

Implementors MUST NOT assume any correspondence between READ boundaries on the control connection and the Telnet EOL sequences (CR LF).

DISCUSSION:
Thus, a server-FTP (or User-FTP) must continue reading characters from the control connection until a complete Telnet EOL sequence is encountered, before processing the command (or response, respectively). Conversely, a single READ from the control connection may include

more than one FTP command.

4.1.2.11 FTP Replies: RFC-959 Section 4.2, Page 35

A Server-FTP MUST send only correctly formatted replies on the control connection. Note that RFC-959 (unlike earlier versions of the FTP spec) contains no provision for a "spontaneous" reply message.

A Server-FTP SHOULD use the reply codes defined in RFC-959 whenever they apply. However, a server-FTP MAY use a different reply code when needed, as long as the general rules of Section 4.2 are followed. When the implementor has a choice between a 4xx and 5xx reply code, a Server-FTP SHOULD send a 4xx (temporary failure) code when there is any reasonable possibility that a failed FTP will succeed a few hours later.

A User-FTP SHOULD generally use only the highest-order digit of a 3-digit reply code for making a procedural decision, to prevent difficulties when a Server-FTP uses non-standard reply codes.

A User-FTP MUST be able to handle multi-line replies. If the implementation imposes a limit on the number of lines and if this limit is exceeded, the User-FTP MUST recover, e.g., by ignoring the excess lines until the end of the multi-line reply is reached.

A User-FTP SHOULD NOT interpret a 421 reply code ("Service not available, closing control connection") specially, but SHOULD detect closing of the control connection by the server.

DISCUSSION:

Server implementations that fail to strictly follow the reply rules often cause FTP user programs to hang. Note that RFC-959 resolved ambiguities in the reply rules found in earlier FTP specifications and must be followed.

It is important to choose FTP reply codes that properly distinguish between temporary and permanent failures, to allow the successful use of file transfer client daemons. These programs depend on the reply codes to decide whether or not to retry a failed transfer; using a permanent failure code (5xx) for a temporary error will cause these programs to give up unnecessarily.

When the meaning of a reply matches exactly the text shown in RFC-959, uniformity will be enhanced by using the RFC-959 text verbatim. However, a Server-FTP implementor is encouraged to choose reply text that conveys specific system-dependent information, when appropriate.

4.1.2.12 Connections: RFC-959 Section 5.2

The words "and the port used" in the second paragraph of this section of RFC-959 are erroneous (historical), and they should be ignored.

On a multihomed server host, the default data transfer port (L-1) MUST be associated with the same local IP address as the corresponding control connection to port L.

A user-FTP MUST NOT send any Telnet controls other than SYNCH and IP on an FTP control connection. In particular, it MUST NOT attempt to negotiate Telnet options on the control connection. However, a server-FTP MUST be capable of accepting and refusing Telnet negotiations (i.e., sending DONT/WONT).

DISCUSSION:
Although the RFC says: "Server- and User- processes should follow the conventions for the Telnet protocol...[on the control connection]", it is not the intent that Telnet option negotiation is to be employed.

4.1.2.13 Minimum Implementation; RFC-959 Section 5.1

The following commands and options MUST be supported by every server-FTP and user-FTP, except in cases where the underlying file system or operating system does not allow or support a particular command.

Type: ASCII Non-print, IMAGE, LOCAL 8
Mode: Stream
Structure: File, Record*
Commands:
USER, PASS, ACCT,
PORT, PASV,
TYPE, MODE, STRU,
RETR, STOR, APPE,
RNFR, RNTO, DELE,
CWD, CDUP, RMD, MKD, PWD,

LIST, NLST,
SYST, STAT,
HELP, NOOP, QUIT.

*Record structure is REQUIRED only for hosts whose file systems support record structure.

DISCUSSION:
Vendors are encouraged to implement a larger subset of the protocol. For example, there are important robustness features in the protocol (e.g., Restart, ABOR, block mode) that would be an aid to some Internet users but are not widely implemented.

A host that does not have record structures in its file system may still accept files with STRU R, recording the byte stream literally.

4.1.3 SPECIFIC ISSUES

4.1.3.1 Non-standard Command Verbs

FTP allows "experimental" commands, whose names begin with "X". If these commands are subsequently adopted as standards, there may still be existing implementations using the "X" form. At present, this is true for the directory commands:

RFC-959 "Experimental"

MKD	XMKD
RMD	XRMD
PWD	XPWD
CDUP	XCUP
CWD	XCWD

All FTP implementations SHOULD recognize both forms of these commands, by simply equating them with extra entries in the command lookup table.

IMPLEMENTATION:
A User-FTP can access a server that supports only the "X" forms by implementing a mode switch, or automatically using the following procedure: if the RFC-959 form of one of the above commands is rejected with a 500 or 502 response code, then try the experimental form; any other response would be passed to the user.

4.1.3.2 Idle Timeout

A Server-FTP process SHOULD have an idle timeout, which will terminate the process and close the control connection if the server is inactive (i.e., no command or data transfer in progress) for a long period of time. The idle timeout time SHOULD be configurable, and the default should be at least 5 minutes.

A client FTP process ("User-PI" in RFC-959) will need timeouts on responses only if it is invoked from a program.

DISCUSSION:
Without a timeout, a Server-FTP process may be left pending indefinitely if the corresponding client crashes without closing the control connection.

4.1.3.3 Concurrency of Data and Control

DISCUSSION:
The intent of the designers of FTP was that a user should be able to send a STAT command at any time while data transfer was in progress and that the server-FTP would reply immediately with status -- e.g., the number of bytes transferred so far. Similarly, an ABOR command should be possible at any time during a data transfer.

Unfortunately, some small-machine operating systems make such concurrent programming difficult, and some other implementers seek minimal solutions, so some FTP implementations do not allow concurrent use of the data and control connections. Even such a minimal server must be prepared to accept and defer a STAT or ABOR command that arrives during data transfer.

4.1.3.4 FTP Restart Mechanism

The description of the 110 reply on pp. 40-41 of RFC-959 is incorrect; the correct description is as follows. A restart reply message, sent over the control connection from the receiving FTP to the User-FTP, has the format:

110 MARK ssss = rrrr

Here:

* ssss is a text string that appeared in a Restart Marker

in the data stream and encodes a position in the sender's file system;

* rrrr encodes the corresponding position in the receiver's file system.

The encoding, which is specific to a particular file system and network implementation, is always generated and interpreted by the same system, either sender or receiver.

When an FTP that implements restart receives a Restart Marker in the data stream, it SHOULD force the data to that point to be written to stable storage before encoding the corresponding position rrrr. An FTP sending Restart Markers MUST NOT assume that 110 replies will be returned synchronously with the data, i.e., it must not await a 110 reply before sending more data.

Two new reply codes are hereby defined for errors encountered in restarting a transfer:

554 Requested action not taken: invalid REST parameter.

A 554 reply may result from a FTP service command that follows a REST command. The reply indicates that the existing file at the Server-FTP cannot be repositioned as specified in the REST.

555 Requested action not taken: type or stru mismatch.

A 555 reply may result from an APPE command or from any FTP service command following a REST command. The reply indicates that there is some mismatch between the current transfer parameters (type and stru) and the attributes of the existing file.

DISCUSSION:

Note that the FTP Restart mechanism requires that Block or Compressed mode be used for data transfer, to allow the Restart Markers to be included within the data stream. The frequency of Restart Markers can be low.

Restart Markers mark a place in the data stream, but the receiver may be performing some transformation on the data as it is stored into stable storage. In general, the receiver's encoding must include any state information necessary to restart this transformation at any point of the FTP data stream. For example, in TYPE

A transfers, some receiver hosts transform CR LF sequences into a single LF character on disk. If a Restart Marker happens to fall between CR and LF, the receiver must encode in rrrr that the transfer must be restarted in a "CR has been seen and discarded" state.

Note that the Restart Marker is required to be encoded as a string of printable ASCII characters, regardless of the type of the data.

RFC-959 says that restart information is to be returned "to the user". This should not be taken literally. In general, the User-FTP should save the restart information (ssss,rrrr) in stable storage, e.g., append it to a restart control file. An empty restart control file should be created when the transfer first starts and deleted automatically when the transfer completes successfully. It is suggested that this file have a name derived in an easily-identifiable manner from the name of the file being transferred and the remote host name; this is analogous to the means used by many text editors for naming "backup" files.

There are three cases for FTP restart.

(1) User-to-Server Transfer

The User-FTP puts Restart Markers <ssss> at convenient places in the data stream. When the Server-FTP receives a Marker, it writes all prior data to disk, encodes its file system position and transformation state as rrrr, and returns a "110 MARK ssss = rrrr" reply over the control connection. The User-FTP appends the pair (ssss,rrrr) to its restart control file.

To restart the transfer, the User-FTP fetches the last (ssss,rrrr) pair from the restart control file, repositions its local file system and transformation state using ssss, and sends the command "REST rrrr" to the Server-FTP.

(2) Server-to-User Transfer

The Server-FTP puts Restart Markers <ssss> at convenient places in the data stream. When the User-FTP receives a Marker, it writes all prior data to disk, encodes its file system position and

transformation state as rrrr, and appends the pair (rrrr,ssss) to its restart control file.

To restart the transfer, the User-FTP fetches the last (rrrr,ssss) pair from the restart control file, repositions its local file system and transformation state using rrrr, and sends the command "REST ssss" to the Server-FTP.

(3) Server-to-Server ("Third-Party") Transfer

The sending Server-FTP puts Restart Markers <ssss> at convenient places in the data stream. When it receives a Marker, the receiving Server-FTP writes all prior data to disk, encodes its file system position and transformation state as rrrr, and sends a "110 MARK ssss = rrrr" reply over the control connection to the User. The User-FTP appends the pair (ssss,rrrr) to its restart control file.

To restart the transfer, the User-FTP fetches the last (ssss,rrrr) pair from the restart control file, sends "REST ssss" to the sending Server-FTP, and sends "REST rrrr" to the receiving Server-FTP.

4.1.4 FTP/USER INTERFACE

This section discusses the user interface for a User-FTP program.

4.1.4.1 Pathname Specification

Since FTP is intended for use in a heterogeneous environment, User-FTP implementations MUST support remote pathnames as arbitrary character strings, so that their form and content are not limited by the conventions of the local operating system.

DISCUSSION:
In particular, remote pathnames can be of arbitrary length, and all the printing ASCII characters as well as space (0x20) must be allowed. RFC-959 allows a pathname to contain any 7-bit ASCII character except CR or LF.

4.1.4.2 "QUOTE" Command

A User-FTP program MUST implement a "QUOTE" command that will pass an arbitrary character string to the server and display all resulting response messages to the user.

To make the "QUOTE" command useful, a User-FTP SHOULD send transfer control commands to the server as the user enters them, rather than saving all the commands and sending them to the server only when a data transfer is started.

DISCUSSION:
The "QUOTE" command is essential to allow the user to access servers that require system-specific commands (e.g., SITE or ALLO), or to invoke new or optional features that are not implemented by the User-FTP. For example, "QUOTE" may be used to specify "TYPE A T" to send a print file to hosts that require the distinction, even if the User-FTP does not recognize that TYPE.

4.1.4.3 Displaying Replies to User

A User-FTP SHOULD display to the user the full text of all error reply messages it receives. It SHOULD have a "verbose" mode in which all commands it sends and the full text and reply codes it receives are displayed, for diagnosis of problems.

4.1.4.4 Maintaining Synchronization

The state machine in a User-FTP SHOULD be forgiving of missing and unexpected reply messages, in order to maintain command synchronization with the server.

4.1.5 FTP REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Implement TYPE T if same as TYPE N	4.1.2.2		x				
File/Record transform invertible if poss.	4.1.2.4		x				
User-FTP send PORT cmd for stream mode	4.1.2.5		x				
Server-FTP implement PASV	4.1.2.6	x					
PASV is per-transfer	4.1.2.6	x					
NLST reply usable in RETR cmds	4.1.2.7	x					
Implied type for LIST and NLST	4.1.2.7		x				
SITE cmd for non-standard features	4.1.2.8		x				
STOU cmd return pathname as specified	4.1.2.9	x					
Use TCP READ boundaries on control conn.	4.1.2.10					x	
Server-FTP send only correct reply format	4.1.2.11	x					
Server-FTP use defined reply code if poss.	4.1.2.11		x				
New reply code following Section 4.2	4.1.2.11			x			
User-FTP use only high digit of reply	4.1.2.11		x				
User-FTP handle multi-line reply lines	4.1.2.11	x					
User-FTP handle 421 reply specially	4.1.2.11				x		
Default data port same IP addr as ctl conn	4.1.2.12	x					
User-FTP send Telnet cmds exc. SYNCH, IP	4.1.2.12					x	
User-FTP negotiate Telnet options	4.1.2.12					x	
Server-FTP handle Telnet options	4.1.2.12	x					
Handle "Experimental" directory cmds	4.1.3.1		x				
Idle timeout in server-FTP	4.1.3.2		x				
Configurable idle timeout	4.1.3.2		x				
Receiver checkpoint data at Restart Marker	4.1.3.4		x				
Sender assume 110 replies are synchronous	4.1.3.4					x	
Support TYPE:							
ASCII - Non-Print (AN)	4.1.2.13	x					
ASCII - Telnet (AT) -- if same as AN	4.1.2.2		x				
ASCII - Carriage Control (AC)	959 3.1.1.5.2			x			
EBCDIC - (any form)	959 3.1.1.2			x			
IMAGE	4.1.2.1	x					
LOCAL 8	4.1.2.1	x					

LOCAL m	4.1.2.1		x			2
Support MODE:						
Stream	4.1.2.13	x				
Block	959 3.4.2		x			
Support STRUCTURE:						
File	4.1.2.13	x				
Record	4.1.2.13	x				3
Page	4.1.2.3			x		
Support commands:						
USER	4.1.2.13	x				
PASS	4.1.2.13	x				
ACCT	4.1.2.13	x				
CWD	4.1.2.13	x				
CDUP	4.1.2.13	x				
SMNT	959 5.3.1		x			
REIN	959 5.3.1		x			
QUIT	4.1.2.13	x				
PORT	4.1.2.13	x				
PASV	4.1.2.6	x				
TYPE	4.1.2.13	x				1
STRU	4.1.2.13	x				1
MODE	4.1.2.13	x				1
RETR	4.1.2.13	x				
STOR	4.1.2.13	x				
STOU	959 5.3.1		x			
APPE	4.1.2.13	x				
ALLO	959 5.3.1		x			
REST	959 5.3.1		x			
RNFR	4.1.2.13	x				
RNTO	4.1.2.13	x				
ABOR	959 5.3.1		x			
DELE	4.1.2.13	x				
RMD	4.1.2.13	x				
MKD	4.1.2.13	x				
PWD	4.1.2.13	x				
LIST	4.1.2.13	x				
NLST	4.1.2.13	x				
SITE	4.1.2.8		x			
STAT	4.1.2.13	x				
SYST	4.1.2.13	x				
HELP	4.1.2.13	x				
NOOP	4.1.2.13	x				

User Interface:						
Arbitrary pathnames	4.1.4.1	x				
Implement "QUOTE" command	4.1.4.2	x				
Transfer control commands immediately	4.1.4.2		x			
Display error messages to user	4.1.4.3		x			
Verbose mode	4.1.4.3		x			
Maintain synchronization with server	4.1.4.4		x			

Footnotes:

(1) For the values shown earlier.

(2) Here m is number of bits in a memory word.

(3) Required for host with record-structured file system, optional otherwise.

4.2 TRIVIAL FILE TRANSFER PROTOCOL -- TFTP

4.2.1 INTRODUCTION

The Trivial File Transfer Protocol TFTP is defined in RFC-783 [TFTP:1].

TFTP provides its own reliable delivery with UDP as its transport protocol, using a simple stop-and-wait acknowledgment system. Since TFTP has an effective window of only one 512 octet segment, it can provide good performance only over paths that have a small delay*bandwidth product. The TFTP file interface is very simple, providing no access control or security.

TFTP's most important application is bootstrapping a host over a local network, since it is simple and small enough to be easily implemented in EPROM [BOOT:1, BOOT:2]. Vendors are urged to support TFTP for booting.

4.2.2 PROTOCOL WALK-THROUGH

The TFTP specification [TFTP:1] is written in an open style, and does not fully specify many parts of the protocol.

4.2.2.1 Transfer Modes: RFC-783, Page 3

The transfer mode "mail" SHOULD NOT be supported.

4.2.2.2 UDP Header: RFC-783, Page 17

The Length field of a UDP header is incorrectly defined; it includes the UDP header length (8).

4.2.3 SPECIFIC ISSUES

4.2.3.1 Sorcerer's Apprentice Syndrome

There is a serious bug, known as the "Sorcerer's Apprentice Syndrome," in the protocol specification. While it does not cause incorrect operation of the transfer (the file will always be transferred correctly if the transfer completes), this bug may cause excessive retransmission, which may cause the transfer to time out.

Implementations MUST contain the fix for this problem: the sender (i.e., the side originating the DATA packets) must never resend the current DATA packet on receipt of a

duplicate ACK.

DISCUSSION:

The bug is caused by the protocol rule that either side, on receiving an old duplicate datagram, may resend the current datagram. If a packet is delayed in the network but later successfully delivered after either side has timed out and retransmitted a packet, a duplicate copy of the response may be generated. If the other side responds to this duplicate with a duplicate of its own, then every datagram will be sent in duplicate for the remainder of the transfer (unless a datagram is lost, breaking the repetition). Worse yet, since the delay is often caused by congestion, this duplicate transmission will usually causes more congestion, leading to more delayed packets, etc.

The following example may help to clarify this problem.

```
    TFTP A                    TFTP B

(1)  Receive ACK X-1
     Send DATA X
(2)                           Receive DATA X
                              Send ACK X
       (ACK X is delayed in network,
        and  A times out):
(3)  Retransmit DATA X

(4)                           Receive DATA X again
                              Send ACK X again
(5)  Receive (delayed) ACK X
     Send DATA X+1
(6)                           Receive DATA X+1
                              Send ACK X+1
(7)  Receive ACK X again
     Send DATA X+1 again
(8)                           Receive DATA X+1 again
                              Send ACK X+1 again
(9)  Receive ACK X+1
     Send DATA X+2
(10)                          Receive DATA X+2
                              Send ACK X+3
(11) Receive ACK X+1 again
     Send DATA X+2 again
(12)                          Receive DATA X+2 again
                              Send ACK X+3 again
```

Notice that once the delayed ACK arrives, the protocol settles down to duplicate all further packets (sequences 5-8 and 9-12). The problem is caused not by either side timing out, but by both sides retransmitting the current packet when they receive a duplicate.

The fix is to break the retransmission loop, as indicated above. This is analogous to the behavior of TCP. It is then possible to remove the retransmission timer on the receiver, since the resent ACK will never cause any action; this is a useful simplification where TFTP is used in a bootstrap program. It is OK to allow the timer to remain, and it may be helpful if the retransmitted ACK replaces one that was genuinely lost in the network. The sender still requires a retransmit timer, of course.

4.2.3.2 Timeout Algorithms

A TFTP implementation MUST use an adaptive timeout.

IMPLEMENTATION:
TCP retransmission algorithms provide a useful base to work from. At least an exponential backoff of retransmission timeout is necessary.

4.2.3.3 Extensions

A variety of non-standard extensions have been made to TFTP, including additional transfer modes and a secure operation mode (with passwords). None of these have been standardized.

4.2.3.4 Access Control

A server TFTP implementation SHOULD include some configurable access control over what pathnames are allowed in TFTP operations.

4.2.3.5 Broadcast Request

A TFTP request directed to a broadcast address SHOULD be silently ignored.

DISCUSSION:
Due to the weak access control capability of TFTP, directed broadcasts of TFTP requests to random networks

could create a significant security hole.

4.2.4 TFTP REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Fix Sorcerer's Apprentice Syndrome	4.2.3.1	x					
Transfer modes:							
netascii	RFC-783	x					
octet	RFC-783	x					
mail	4.2.2.1				x		
extensions	4.2.3.3			x			
Use adaptive timeout	4.2.3.2	x					
Configurable access control	4.2.3.4		x				
Silently ignore broadcast request	4.2.3.5		x				

5. ELECTRONIC MAIL -- SMTP and RFC-822

5.1 INTRODUCTION

In the TCP/IP protocol suite, electronic mail in a format specified in RFC-822 [SMTP:2] is transmitted using the Simple Mail Transfer Protocol (SMTP) defined in RFC-821 [SMTP:1].

While SMTP has remained unchanged over the years, the Internet community has made several changes in the way SMTP is used. In particular, the conversion to the Domain Name System (DNS) has caused changes in address formats and in mail routing. In this section, we assume familiarity with the concepts and terminology of the DNS, whose requirements are given in Section 6.1.

RFC-822 specifies the Internet standard format for electronic mail messages. RFC-822 supercedes an older standard, RFC-733, that may still be in use in a few places, although it is obsolete. The two formats are sometimes referred to simply by number ("822" and "733").

RFC-822 is used in some non-Internet mail environments with different mail transfer protocols than SMTP, and SMTP has also been adapted for use in some non-Internet environments. Note that this document presents the rules for the use of SMTP and RFC-822 for the Internet environment only; other mail environments that use these protocols may be expected to have their own rules.

5.2 PROTOCOL WALK-THROUGH

This section covers both RFC-821 and RFC-822.

The SMTP specification in RFC-821 is clear and contains numerous examples, so implementors should not find it difficult to understand. This section simply updates or annotates portions of RFC-821 to conform with current usage.

RFC-822 is a long and dense document, defining a rich syntax. Unfortunately, incomplete or defective implementations of RFC-822 are common. In fact, nearly all of the many formats of RFC-822 are actually used, so an implementation generally needs to recognize and correctly interpret all of the RFC-822 syntax.

5.2.1 The SMTP Model: RFC-821 Section 2

DISCUSSION:
Mail is sent by a series of request/response transactions between a client, the "sender-SMTP," and a server, the

"receiver-SMTP". These transactions pass (1) the message proper, which is composed of header and body, and (2) SMTP source and destination addresses, referred to as the "envelope".

The SMTP programs are analogous to Message Transfer Agents (MTAs) of X.400. There will be another level of protocol software, closer to the end user, that is responsible for composing and analyzing RFC-822 message headers; this component is known as the "User Agent" in X.400, and we use that term in this document. There is a clear logical distinction between the User Agent and the SMTP implementation, since they operate on different levels of protocol. Note, however, that this distinction is may not be exactly reflected the structure of typical implementations of Internet mail. Often there is a program known as the "mailer" that implements SMTP and also some of the User Agent functions; the rest of the User Agent functions are included in a user interface used for entering and reading mail.

The SMTP envelope is constructed at the originating site, typically by the User Agent when the message is first queued for the Sender-SMTP program. The envelope addresses may be derived from information in the message header, supplied by the user interface (e.g., to implement a bcc: request), or derived from local configuration information (e.g., expansion of a mailing list). The SMTP envelope cannot in general be re-derived from the header at a later stage in message delivery, so the envelope is transmitted separately from the message itself using the MAIL and RCPT commands of SMTP.

The text of RFC-821 suggests that mail is to be delivered to an individual user at a host. With the advent of the domain system and of mail routing using mail-exchange (MX) resource records, implementors should now think of delivering mail to a user at a domain, which may or may not be a particular host. This DOES NOT change the fact that SMTP is a host-to-host mail exchange protocol.

5.2.2 Canonicalization: RFC-821 Section 3.1

The domain names that a Sender-SMTP sends in MAIL and RCPT commands MUST have been "canonicalized," i.e., they must be fully-qualified principal names or domain literals, not nicknames or domain abbreviations. A canonicalized name either identifies a host directly or is an MX name; it cannot be a

CNAME.

5.2.3 VRFY and EXPN Commands: RFC-821 Section 3.3

A receiver-SMTP MUST implement VRFY and SHOULD implement EXPN (this requirement overrides RFC-821). However, there MAY be configuration information to disable VRFY and EXPN in a particular installation; this might even allow EXPN to be disabled for selected lists.

A new reply code is defined for the VRFY command:

252 Cannot VRFY user (e.g., info is not local), but will take message for this user and attempt delivery.

DISCUSSION:
SMTP users and administrators make regular use of these commands for diagnosing mail delivery problems. With the increasing use of multi-level mailing list expansion (sometimes more than two levels), EXPN has been increasingly important for diagnosing inadvertent mail loops. On the other hand, some feel that EXPN represents a significant privacy, and perhaps even a security, exposure.

5.2.4 SEND, SOML, and SAML Commands: RFC-821 Section 3.4

An SMTP MAY implement the commands to send a message to a user's terminal: SEND, SOML, and SAML.

DISCUSSION:
It has been suggested that the use of mail relaying through an MX record is inconsistent with the intent of SEND to deliver a message immediately and directly to a user's terminal. However, an SMTP receiver that is unable to write directly to the user terminal can return a "251 User Not Local" reply to the RCPT following a SEND, to inform the originator of possibly deferred delivery.

5.2.5 HELO Command: RFC-821 Section 3.5

The sender-SMTP MUST ensure that the <domain> parameter in a HELO command is a valid principal host domain name for the client host. As a result, the receiver-SMTP will not have to perform MX resolution on this name in order to validate the HELO parameter.

The HELO receiver MAY verify that the HELO parameter really

corresponds to the IP address of the sender. However, the receiver MUST NOT refuse to accept a message, even if the sender's HELO command fails verification.

DISCUSSION:
Verifying the HELO parameter requires a domain name lookup and may therefore take considerable time. An alternative tool for tracking bogus mail sources is suggested below (see "DATA Command").

Note also that the HELO argument is still required to have valid <domain> syntax, since it will appear in a Received: line; otherwise, a 501 error is to be sent.

IMPLEMENTATION:
When HELO parameter validation fails, a suggested procedure is to insert a note about the unknown authenticity of the sender into the message header (e.g., in the "Received:" line).

5.2.6 Mail Relay: RFC-821 Section 3.6

We distinguish three types of mail (store-and-) forwarding:

(1) A simple forwarder or "mail exchanger" forwards a message using private knowledge about the recipient; see section 3.2 of RFC-821.

(2) An SMTP mail "relay" forwards a message within an SMTP mail environment as the result of an explicit source route (as defined in section 3.6 of RFC-821). The SMTP relay function uses the "@...:" form of source route from RFC-822 (see Section 5.2.19 below).

(3) A mail "gateway" passes a message between different environments. The rules for mail gateways are discussed below in Section 5.3.7.

An Internet host that is forwarding a message but is not a gateway to a different mail environment (i.e., it falls under (1) or (2)) SHOULD NOT alter any existing header fields, although the host will add an appropriate Received: line as required in Section 5.2.8.

A Sender-SMTP SHOULD NOT send a RCPT TO: command containing an explicit source route using the "@...:" address form. Thus, the relay function defined in section 3.6 of RFC-821 should not be used.

DISCUSSION:
The intent is to discourage all source routing and to abolish explicit source routing for mail delivery within the Internet environment. Source-routing is unnecessary; the simple target address "user@domain" should always suffice. This is the result of an explicit architectural decision to use universal naming rather than source routing for mail. Thus, SMTP provides end-to-end connectivity, and the DNS provides globally-unique, location-independent names. MX records handle the major case where source routing might otherwise be needed.

A receiver-SMTP MUST accept the explicit source route syntax in the envelope, but it MAY implement the relay function as defined in section 3.6 of RFC-821. If it does not implement the relay function, it SHOULD attempt to deliver the message directly to the host to the right of the right-most "@" sign.

DISCUSSION:
For example, suppose a host that does not implement the relay function receives a message with the SMTP command: "RCPT TO:<@ALPHA,@BETA:joe@GAMMA>", where ALPHA, BETA, and GAMMA represent domain names. Rather than immediately refusing the message with a 550 error reply as suggested on page 20 of RFC-821, the host should try to forward the message to GAMMA directly, using: "RCPT TO:<joe@GAMMA>". Since this host does not support relaying, it is not required to update the reverse path.

Some have suggested that source routing may be needed occasionally for manually routing mail around failures; however, the reality and importance of this need is controversial. The use of explicit SMTP mail relaying for this purpose is discouraged, and in fact it may not be successful, as many host systems do not support it. Some have used the "%-hack" (see Section 5.2.16) for this purpose.

5.2.7 RCPT Command: RFC-821 Section 4.1.1

A host that supports a receiver-SMTP MUST support the reserved mailbox "Postmaster".

The receiver-SMTP MAY verify RCPT parameters as they arrive; however, RCPT responses MUST NOT be delayed beyond a reasonable time (see Section 5.3.2).

Therefore, a "250 OK" response to a RCPT does not necessarily

imply that the delivery address(es) are valid. Errors found after message acceptance will be reported by mailing a notification message to an appropriate address (see Section 5.3.3).

DISCUSSION:
The set of conditions under which a RCPT parameter can be validated immediately is an engineering design choice. Reporting destination mailbox errors to the Sender-SMTP before mail is transferred is generally desirable to save time and network bandwidth, but this advantage is lost if RCPT verification is lengthy.

For example, the receiver can verify immediately any simple local reference, such as a single locally-registered mailbox. On the other hand, the "reasonable time" limitation generally implies deferring verification of a mailing list until after the message has been transferred and accepted, since verifying a large mailing list can take a very long time. An implementation might or might not choose to defer validation of addresses that are non-local and therefore require a DNS lookup. If a DNS lookup is performed but a soft domain system error (e.g., timeout) occurs, validity must be assumed.

5.2.8 DATA Command: RFC-821 Section 4.1.1

Every receiver-SMTP (not just one that "accepts a message for relaying or for final delivery" [SMTP:1]) MUST insert a "Received:" line at the beginning of a message. In this line, called a "time stamp line" in RFC-821:

* The FROM field SHOULD contain both (1) the name of the source host as presented in the HELO command and (2) a domain literal containing the IP address of the source, determined from the TCP connection.

* The ID field MAY contain an "@" as suggested in RFC-822, but this is not required.

* The FOR field MAY contain a list of <path> entries when multiple RCPT commands have been given.

An Internet mail program MUST NOT change a Received: line that was previously added to the message header.

DISCUSSION:
Including both the source host and the IP source address in the Received: line may provide enough information for tracking illicit mail sources and eliminate a need to explicitly verify the HELO parameter.

Received: lines are primarily intended for humans tracing mail routes, primarily of diagnosis of faults. See also the discussion under 5.3.7.

When the receiver-SMTP makes "final delivery" of a message, then it MUST pass the MAIL FROM: address from the SMTP envelope with the message, for use if an error notification message must be sent later (see Section 5.3.3). There is an analogous requirement when gatewaying from the Internet into a different mail environment; see Section 5.3.7.

DISCUSSION:
Note that the final reply to the DATA command depends only upon the successful transfer and storage of the message. Any problem with the destination address(es) must either (1) have been reported in an SMTP error reply to the RCPT command(s), or (2) be reported in a later error message mailed to the originator.

IMPLEMENTATION:
The MAIL FROM: information may be passed as a parameter or in a Return-Path: line inserted at the beginning of the message.

5.2.9 Command Syntax: RFC-821 Section 4.1.2

The syntax shown in RFC-821 for the MAIL FROM: command omits the case of an empty path: "MAIL FROM: <>" (see RFC-821 Page 15). An empty reverse path MUST be supported.

5.2.10 SMTP Replies: RFC-821 Section 4.2

A receiver-SMTP SHOULD send only the reply codes listed in section 4.2.2 of RFC-821 or in this document. A receiver-SMTP SHOULD use the text shown in examples in RFC-821 whenever appropriate.

A sender-SMTP MUST determine its actions only by the reply code, not by the text (except for 251 and 551 replies); any text, including no text at all, must be acceptable. The space (blank) following the reply code is considered part of the text. Whenever possible, a sender-SMTP SHOULD test only the

first digit of the reply code, as specified in Appendix E of RFC-821.

DISCUSSION:
Interoperability problems have arisen with SMTP systems using reply codes that are not listed explicitly in RFC-821 Section 4.3 but are legal according to the theory of reply codes explained in Appendix E.

5.2.11 Transparency: RFC-821 Section 4.5.2

Implementors MUST be sure that their mail systems always add and delete periods to ensure message transparency.

5.2.12 WKS Use in MX Processing: RFC-974, p. 5

RFC-974 [SMTP:3] recommended that the domain system be queried for WKS ("Well-Known Service") records, to verify that each proposed mail target does support SMTP. Later experience has shown that WKS is not widely supported, so the WKS step in MX processing SHOULD NOT be used.

The following are notes on RFC-822, organized by section of that document.

5.2.13 RFC-822 Message Specification: RFC-822 Section 4

The syntax shown for the Return-path line omits the possibility of a null return path, which is used to prevent looping of error notifications (see Section 5.3.3). The complete syntax is:

```
return = "Return-path"  ":" route-addr
       / "Return-path"  ":" "<" ">"
```

The set of optional header fields is hereby expanded to include the Content-Type field defined in RFC-1049 [SMTP:7]. This field "allows mail reading systems to automatically identify the type of a structured message body and to process it for display accordingly". [SMTP:7] A User Agent MAY support this field.

5.2.14 RFC-822 Date and Time Specification: RFC-822 Section 5

The syntax for the date is hereby changed to:

```
date = 1*2DIGIT month 2*4DIGIT
```

All mail software SHOULD use 4-digit years in dates, to ease the transition to the next century.

There is a strong trend towards the use of numeric timezone indicators, and implementations SHOULD use numeric timezones instead of timezone names. However, all implementations MUST accept either notation. If timezone names are used, they MUST be exactly as defined in RFC-822.

The military time zones are specified incorrectly in RFC-822: they count the wrong way from UT (the signs are reversed). As a result, military time zones in RFC-822 headers carry no information.

Finally, note that there is a typo in the definition of "zone" in the syntax summary of appendix D; the correct definition occurs in Section 3 of RFC-822.

5.2.15 RFC-822 Syntax Change: RFC-822 Section 6.1

The syntactic definition of "mailbox" in RFC-822 is hereby changed to:

```
   mailbox =  addr-spec             ; simple address
           / [phrase] route-addr    ; name & addr-spec
```

That is, the phrase preceding a route address is now OPTIONAL. This change makes the following header field legal, for example:

```
    From: <craig@nnsc.nsf.net>
```

5.2.16 RFC-822 Local-part: RFC-822 Section 6.2

The basic mailbox address specification has the form: "local-part@domain". Here "local-part", sometimes called the "left-hand side" of the address, is domain-dependent.

A host that is forwarding the message but is not the destination host implied by the right-hand side "domain" MUST NOT interpret or modify the "local-part" of the address.

When mail is to be gatewayed from the Internet mail environment into a foreign mail environment (see Section 5.3.7), routing information for that foreign environment MAY be embedded within the "local-part" of the address. The gateway will then interpret this local part appropriately for the foreign mail environment.

DISCUSSION:

Although source routes are discouraged within the Internet (see Section 5.2.6), there are non-Internet mail environments whose delivery mechanisms do depend upon source routes. Source routes for extra-Internet environments can generally be buried in the "local-part" of the address (see Section 5.2.16) while mail traverses the Internet. When the mail reaches the appropriate Internet mail gateway, the gateway will interpret the local-part and build the necessary address or route for the target mail environment.

For example, an Internet host might send mail to: "a!b!c!user@gateway-domain". The complex local part "a!b!c!user" would be uninterpreted within the Internet domain, but could be parsed and understood by the specified mail gateway.

An embedded source route is sometimes encoded in the "local-part" using "%" as a right-binding routing operator. For example, in:

user%domain%relay3%relay2@relay1

the "%" convention implies that the mail is to be routed from "relay1" through "relay2", "relay3", and finally to "user" at "domain". This is commonly known as the "%-hack". It is suggested that "%" have lower precedence than any other routing operator (e.g., "!") hidden in the local-part; for example, "a!b%c" would be interpreted as "(a!b)%c".

Only the target host (in this case, "relay1") is permitted to analyze the local-part "user%domain%relay3%relay2".

5.2.17 Domain Literals: RFC-822 Section 6.2.3

A mailer MUST be able to accept and parse an Internet domain literal whose content ("dtext"; see RFC-822) is a dotted-decimal host address. This satisfies the requirement of Section 2.1 for the case of mail.

An SMTP MUST accept and recognize a domain literal for any of its own IP addresses.

5.2.18 Common Address Formatting Errors: RFC-822 Section 6.1

Errors in formatting or parsing 822 addresses are unfortunately common. This section mentions only the most common errors. A User Agent MUST accept all valid RFC-822 address formats, and MUST NOT generate illegal address syntax.

- A common error is to leave out the semicolon after a group identifier.

- Some systems fail to fully-qualify domain names in messages they generate. The right-hand side of an "@" sign in a header address field MUST be a fully-qualified domain name.

  For example, some systems fail to fully-qualify the From: address; this prevents a "reply" command in the user interface from automatically constructing a return address.

  DISCUSSION:
  Although RFC-822 allows the local use of abbreviated domain names within a domain, the application of RFC-822 in Internet mail does not allow this. The intent is that an Internet host must not send an SMTP message header containing an abbreviated domain name in an address field. This allows the address fields of the header to be passed without alteration across the Internet, as required in Section 5.2.6.

- Some systems mis-parse multiple-hop explicit source routes such as:

  @relay1,@relay2,@relay3:user@domain.

- Some systems over-qualify domain names by adding a trailing dot to some or all domain names in addresses or message-ids. This violates RFC-822 syntax.

5.2.19 Explicit Source Routes: RFC-822 Section 6.2.7

Internet host software SHOULD NOT create an RFC-822 header containing an address with an explicit source route, but MUST accept such headers for compatibility with earlier systems.

DISCUSSION:

In an understatement, RFC-822 says "The use of explicit source routing is discouraged". Many hosts implemented RFC-822 source routes incorrectly, so the syntax cannot be used unambiguously in practice. Many users feel the syntax is ugly. Explicit source routes are not needed in the mail envelope for delivery; see Section 5.2.6. For all these reasons, explicit source routes using the RFC-822 notations are not to be used in Internet mail headers.

As stated in Section 5.2.16, it is necessary to allow an explicit source route to be buried in the local-part of an address, e.g., using the "%-hack", in order to allow mail to be gatewayed into another environment in which explicit source routing is necessary. The vigilant will observe that there is no way for a User Agent to detect and prevent the use of such implicit source routing when the destination is within the Internet. We can only discourage source routing of any kind within the Internet, as unnecessary and undesirable.

5.3 SPECIFIC ISSUES

5.3.1 SMTP Queueing Strategies

The common structure of a host SMTP implementation includes user mailboxes, one or more areas for queueing messages in transit, and one or more daemon processes for sending and receiving mail. The exact structure will vary depending on the needs of the users on the host and the number and size of mailing lists supported by the host. We describe several optimizations that have proved helpful, particularly for mailers supporting high traffic levels.

Any queueing strategy MUST include:

- Timeouts on all activities. See Section 5.3.2.
- Never sending error messages in response to error messages.

5.3.1.1 Sending Strategy

The general model of a sender-SMTP is one or more processes that periodically attempt to transmit outgoing mail. In a typical system, the program that composes a message has some method for requesting immediate attention for a new piece of outgoing mail, while mail that cannot be transmitted

immediately MUST be queued and periodically retried by the sender. A mail queue entry will include not only the message itself but also the envelope information.

The sender MUST delay retrying a particular destination after one attempt has failed. In general, the retry interval SHOULD be at least 30 minutes; however, more sophisticated and variable strategies will be beneficial when the sender-SMTP can determine the reason for non-delivery.

Retries continue until the message is transmitted or the sender gives up; the give-up time generally needs to be at least 4-5 days. The parameters to the retry algorithm MUST be configurable.

A sender SHOULD keep a list of hosts it cannot reach and corresponding timeouts, rather than just retrying queued mail items.

DISCUSSION:

Experience suggests that failures are typically transient (the target system has crashed), favoring a policy of two connection attempts in the first hour the message is in the queue, and then backing off to once every two or three hours.

The sender-SMTP can shorten the queueing delay by cooperation with the receiver-SMTP. In particular, if mail is received from a particular address, it is good evidence that any mail queued for that host can now be sent.

The strategy may be further modified as a result of multiple addresses per host (see Section 5.3.4), to optimize delivery time vs. resource usage.

A sender-SMTP may have a large queue of messages for each unavailable destination host, and if it retried all these messages in every retry cycle, there would be excessive Internet overhead and the daemon would be blocked for a long period. Note that an SMTP can generally determine that a delivery attempt has failed only after a timeout of a minute or more; a one minute timeout per connection will result in a very large delay if it is repeated for dozens or even hundreds of queued messages.

When the same message is to be delivered to several users on the same host, only one copy of the message SHOULD be transmitted. That is, the sender-SMTP should use the command sequence: RCPT, RCPT,... RCPT, DATA instead of the sequence: RCPT, DATA, RCPT, DATA,... RCPT, DATA. Implementation of this efficiency feature is strongly urged.

Similarly, the sender-SMTP MAY support multiple concurrent outgoing mail transactions to achieve timely delivery. However, some limit SHOULD be imposed to protect the host from devoting all its resources to mail.

The use of the different addresses of a multihomed host is discussed below.

5.3.1.2 Receiving strategy

The receiver-SMTP SHOULD attempt to keep a pending listen on the SMTP port at all times. This will require the support of multiple incoming TCP connections for SMTP. Some limit MAY be imposed.

IMPLEMENTATION:
When the receiver-SMTP receives mail from a particular host address, it could notify the sender-SMTP to retry any mail pending for that host address.

5.3.2 Timeouts in SMTP

There are two approaches to timeouts in the sender-SMTP: (a) limit the time for each SMTP command separately, or (b) limit the time for the entire SMTP dialogue for a single mail message. A sender-SMTP SHOULD use option (a), per-command timeouts. Timeouts SHOULD be easily reconfigurable, preferably without recompiling the SMTP code.

DISCUSSION:
Timeouts are an essential feature of an SMTP implementation. If the timeouts are too long (or worse, there are no timeouts), Internet communication failures or software bugs in receiver-SMTP programs can tie up SMTP processes indefinitely. If the timeouts are too short, resources will be wasted with attempts that time out part way through message delivery.

If option (b) is used, the timeout has to be very large, e.g., an hour, to allow time to expand very large mailing lists. The timeout may also need to increase linearly

with the size of the message, to account for the time to transmit a very large message. A large fixed timeout leads to two problems: a failure can still tie up the sender for a very long time, and very large messages may still spuriously time out (which is a wasteful failure!).

Using the recommended option (a), a timer is set for each SMTP command and for each buffer of the data transfer. The latter means that the overall timeout is inherently proportional to the size of the message.

Based on extensive experience with busy mail-relay hosts, the minimum per-command timeout values SHOULD be as follows:

- Initial 220 Message: 5 minutes

  A Sender-SMTP process needs to distinguish between a failed TCP connection and a delay in receiving the initial 220 greeting message. Many receiver-SMTPs will accept a TCP connection but delay delivery of the 220 message until their system load will permit more mail to be processed.

- MAIL Command: 5 minutes

- RCPT Command: 5 minutes

  A longer timeout would be required if processing of mailing lists and aliases were not deferred until after the message was accepted.

- DATA Initiation: 2 minutes

  This is while awaiting the "354 Start Input" reply to a DATA command.

- Data Block: 3 minutes

  This is while awaiting the completion of each TCP SEND call transmitting a chunk of data.

- DATA Termination: 10 minutes.

  This is while awaiting the "250 OK" reply. When the receiver gets the final period terminating the message data, it typically performs processing to deliver the message to a user mailbox. A spurious timeout at this point would be very wasteful, since the message has been

successfully sent.

A receiver-SMTP SHOULD have a timeout of at least 5 minutes while it is awaiting the next command from the sender.

5.3.3 Reliable Mail Receipt

When the receiver-SMTP accepts a piece of mail (by sending a "250 OK" message in response to DATA), it is accepting responsibility for delivering or relaying the message. It must take this responsibility seriously, i.e., it MUST NOT lose the message for frivolous reasons, e.g., because the host later crashes or because of a predictable resource shortage.

If there is a delivery failure after acceptance of a message, the receiver-SMTP MUST formulate and mail a notification message. This notification MUST be sent using a null ("<>") reverse path in the envelope; see Section 3.6 of RFC-821. The recipient of this notification SHOULD be the address from the envelope return path (or the Return-Path: line). However, if this address is null ("<>"), the receiver-SMTP MUST NOT send a notification. If the address is an explicit source route, it SHOULD be stripped down to its final hop.

DISCUSSION:

For example, suppose that an error notification must be sent for a message that arrived with: "MAIL FROM:<@a,@b:user@d>". The notification message should be sent to: "RCPT TO:<user@d>".

Some delivery failures after the message is accepted by SMTP will be unavoidable. For example, it may be impossible for the receiver-SMTP to validate all the delivery addresses in RCPT command(s) due to a "soft" domain system error or because the target is a mailing list (see earlier discussion of RCPT).

To avoid receiving duplicate messages as the result of timeouts, a receiver-SMTP MUST seek to minimize the time required to respond to the final "." that ends a message transfer. See RFC-1047 [SMTP:4] for a discussion of this problem.

5.3.4 Reliable Mail Transmission

To transmit a message, a sender-SMTP determines the IP address of the target host from the destination address in the envelope. Specifically, it maps the string to the right of the

"@" sign into an IP address. This mapping or the transfer itself may fail with a soft error, in which case the sender-SMTP will requeue the outgoing mail for a later retry, as required in Section 5.3.1.1.

When it succeeds, the mapping can result in a list of alternative delivery addresses rather than a single address, because of (a) multiple MX records, (b) multihoming, or both. To provide reliable mail transmission, the sender-SMTP MUST be able to try (and retry) each of the addresses in this list in order, until a delivery attempt succeeds. However, there MAY also be a configurable limit on the number of alternate addresses that can be tried. In any case, a host SHOULD try at least two addresses.

The following information is to be used to rank the host addresses:

(1) Multiple MX Records -- these contain a preference indication that should be used in sorting. If there are multiple destinations with the same preference and there is no clear reason to favor one (e.g., by address preference), then the sender-SMTP SHOULD pick one at random to spread the load across multiple mail exchanges for a specific organization; note that this is a refinement of the procedure in [DNS:3].

(2) Multihomed host -- The destination host (perhaps taken from the preferred MX record) may be multihomed, in which case the domain name resolver will return a list of alternative IP addresses. It is the responsibility of the domain name resolver interface (see Section 6.1.3.4 below) to have ordered this list by decreasing preference, and SMTP MUST try them in the order presented.

DISCUSSION:
Although the capability to try multiple alternative addresses is required, there may be circumstances where specific installations want to limit or disable the use of alternative addresses. The question of whether a sender should attempt retries using the different addresses of a multihomed host has been controversial. The main argument for using the multiple addresses is that it maximizes the probability of timely delivery, and indeed sometimes the probability of any delivery; the counter argument is that it may result in unnecessary resource use.

Note that resource use is also strongly determined by the

sending strategy discussed in Section 5.3.1.

5.3.5 Domain Name Support

SMTP implementations MUST use the mechanism defined in Section 6.1 for mapping between domain names and IP addresses. This means that every Internet SMTP MUST include support for the Internet DNS.

In particular, a sender-SMTP MUST support the MX record scheme [SMTP:3]. See also Section 7.4 of [DNS:2] for information on domain name support for SMTP.

5.3.6 Mailing Lists and Aliases

An SMTP-capable host SHOULD support both the alias and the list form of address expansion for multiple delivery. When a message is delivered or forwarded to each address of an expanded list form, the return address in the envelope ("MAIL FROM:") MUST be changed to be the address of a person who administers the list, but the message header MUST be left unchanged; in particular, the "From" field of the message is unaffected.

DISCUSSION:

An important mail facility is a mechanism for multi-destination delivery of a single message, by transforming or "expanding" a pseudo-mailbox address into a list of destination mailbox addresses. When a message is sent to such a pseudo-mailbox (sometimes called an "exploder"), copies are forwarded or redistributed to each mailbox in the expanded list. We classify such a pseudo-mailbox as an "alias" or a "list", depending upon the expansion rules:

(a) Alias

To expand an alias, the recipient mailer simply replaces the pseudo-mailbox address in the envelope with each of the expanded addresses in turn; the rest of the envelope and the message body are left unchanged. The message is then delivered or forwarded to each expanded address.

(b) List

A mailing list may be said to operate by "redistribution" rather than by "forwarding". To

expand a list, the recipient mailer replaces the pseudo-mailbox address in the envelope with each of the expanded addresses in turn. The return address in the envelope is changed so that all error messages generated by the final deliveries will be returned to a list administrator, not to the message originator, who generally has no control over the contents of the list and will typically find error messages annoying.

5.3.7 Mail Gatewaying

Gatewaying mail between different mail environments, i.e., different mail formats and protocols, is complex and does not easily yield to standardization. See for example [SMTP:5a], [SMTP:5b]. However, some general requirements may be given for a gateway between the Internet and another mail environment.

(A) Header fields MAY be rewritten when necessary as messages are gatewayed across mail environment boundaries.

DISCUSSION:
This may involve interpreting the local-part of the destination address, as suggested in Section 5.2.16.

The other mail systems gatewayed to the Internet generally use a subset of RFC-822 headers, but some of them do not have an equivalent to the SMTP envelope. Therefore, when a message leaves the Internet environment, it may be necessary to fold the SMTP envelope information into the message header. A possible solution would be to create new header fields to carry the envelope information (e.g., "X-SMTP-MAIL:" and "X-SMTP-RCPT:"); however, this would require changes in mail programs in the foreign environment.

(B) When forwarding a message into or out of the Internet environment, a gateway MUST prepend a Received: line, but it MUST NOT alter in any way a Received: line that is already in the header.

DISCUSSION:
This requirement is a subset of the general "Received:" line requirement of Section 5.2.8; it is restated here for emphasis.

Received: fields of messages originating from other

environments may not conform exactly to RFC822. However, the most important use of Received: lines is for debugging mail faults, and this debugging can be severely hampered by well-meaning gateways that try to "fix" a Received: line.

The gateway is strongly encouraged to indicate the environment and protocol in the "via" clauses of Received field(s) that it supplies.

(C) From the Internet side, the gateway SHOULD accept all valid address formats in SMTP commands and in RFC-822 headers, and all valid RFC-822 messages. Although a gateway must accept an RFC-822 explicit source route ("@...:" format) in either the RFC-822 header or in the envelope, it MAY or may not act on the source route; see Sections 5.2.6 and 5.2.19.

DISCUSSION:
It is often tempting to restrict the range of addresses accepted at the mail gateway to simplify the translation into addresses for the remote environment. This practice is based on the assumption that mail users have control over the addresses their mailers send to the mail gateway. In practice, however, users have little control over the addresses that are finally sent; their mailers are free to change addresses into any legal RFC-822 format.

(D) The gateway MUST ensure that all header fields of a message that it forwards into the Internet meet the requirements for Internet mail. In particular, all addresses in "From:", "To:", "Cc:", etc., fields must be transformed (if necessary) to satisfy RFC-822 syntax, and they must be effective and useful for sending replies.

(E) The translation algorithm used to convert mail from the Internet protocols to another environment's protocol SHOULD try to ensure that error messages from the foreign mail environment are delivered to the return path from the SMTP envelope, not to the sender listed in the "From:" field of the RFC-822 message.

DISCUSSION:
Internet mail lists usually place the address of the mail list maintainer in the envelope but leave the

original message header intact (with the "From:" field containing the original sender). This yields the behavior the average recipient expects: a reply to the header gets sent to the original sender, not to a mail list maintainer; however, errors get sent to the maintainer (who can fix the problem) and not the sender (who probably cannot).

(F) Similarly, when forwarding a message from another environment into the Internet, the gateway SHOULD set the envelope return path in accordance with an error message return address, if any, supplied by the foreign environment.

5.3.8 Maximum Message Size

Mailer software MUST be able to send and receive messages of at least 64K bytes in length (including header), and a much larger maximum size is highly desirable.

DISCUSSION:

Although SMTP does not define the maximum size of a message, many systems impose implementation limits.

The current de facto minimum limit in the Internet is 64K bytes. However, electronic mail is used for a variety of purposes that create much larger messages. For example, mail is often used instead of FTP for transmitting ASCII files, and in particular to transmit entire documents. As a result, messages can be 1 megabyte or even larger. We note that the present document together with its lower-layer companion contains 0.5 megabytes.

5.4 SMTP REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
RECEIVER-SMTP:							
Implement VRFY	5.2.3	x					
Implement EXPN	5.2.3		x				
EXPN, VRFY configurable	5.2.3			x			
Implement SEND, SOML, SAML	5.2.4			x			
Verify HELO parameter	5.2.5			x			
Refuse message with bad HELO	5.2.5					x	
Accept explicit src-route syntax in env.	5.2.6	x					
Support "postmaster"	5.2.7	x					
Process RCPT when received (except lists)	5.2.7			x			
Long delay of RCPT responses	5.2.7					x	
Add Received: line	5.2.8	x					
Received: line include domain literal	5.2.8		x				
Change previous Received: line	5.2.8					x	
Pass Return-Path info (final deliv/gwy)	5.2.8	x					
Support empty reverse path	5.2.9	x					
Send only official reply codes	5.2.10		x				
Send text from RFC-821 when appropriate	5.2.10		x				
Delete "." for transparency	5.2.11	x					
Accept and recognize self domain literal(s)	5.2.17	x					
Error message about error message	5.3.1					x	
Keep pending listen on SMTP port	5.3.1.2		x				
Provide limit on recv concurrency	5.3.1.2			x			
Wait at least 5 mins for next sender cmd	5.3.2		x				
Avoidable delivery failure after "250 OK"	5.3.3					x	
Send error notification msg after accept	5.3.3	x					
Send using null return path	5.3.3	x					
Send to envelope return path	5.3.3		x				
Send to null address	5.3.3					x	
Strip off explicit src route	5.3.3		x				
Minimize acceptance delay (RFC-1047)	5.3.3	x					

SENDER-SMTP:							
Canonicalized domain names in MAIL, RCPT	5.2.2	x					
Implement SEND, SOML, SAML	5.2.4			x			
Send valid principal host name in HELO	5.2.5	x					
Send explicit source route in RCPT TO:	5.2.6				x		
Use only reply code to determine action	5.2.10	x					
Use only high digit of reply code when poss.	5.2.10		x				
Add "." for transparency	5.2.11	x					
Retry messages after soft failure	5.3.1.1	x					
Delay before retry	5.3.1.1	x					
Configurable retry parameters	5.3.1.1	x					
Retry once per each queued dest host	5.3.1.1		x				
Multiple RCPT's for same DATA	5.3.1.1		x				
Support multiple concurrent transactions	5.3.1.1			x			
Provide limit on concurrency	5.3.1.1		x				
Timeouts on all activities	5.3.1	x					
Per-command timeouts	5.3.2		x				
Timeouts easily reconfigurable	5.3.2		x				
Recommended times	5.3.2		x				
Try alternate addr's in order	5.3.4	x					
Configurable limit on alternate tries	5.3.4			x			
Try at least two alternates	5.3.4		x				
Load-split across equal MX alternates	5.3.4		x				
Use the Domain Name System	5.3.5	x					
Support MX records	5.3.5	x					
Use WKS records in MX processing	5.2.12				x		
MAIL FORWARDING:							
Alter existing header field(s)	5.2.6				x		
Implement relay function: 821/section 3.6	5.2.6			x			
If not, deliver to RHS domain	5.2.6		x				
Interpret 'local-part' of addr	5.2.16					x	
MAILING LISTS AND ALIASES							
Support both	5.3.6		x				
Report mail list error to local admin.	5.3.6	x					
MAIL GATEWAYS:							
Embed foreign mail route in local-part	5.2.16			x			
Rewrite header fields when necessary	5.3.7			x			
Prepend Received: line	5.3.7	x					
Change existing Received: line	5.3.7					x	
Accept full RFC-822 on Internet side	5.3.7		x				
Act on RFC-822 explicit source route	5.3.7			x			

Send only valid RFC-822 on Internet side	5.3.7	x				
Deliver error msgs to envelope addr	5.3.7		x			
Set env return path from err return addr	5.3.7		x			
USER AGENT -- RFC-822						
Allow user to enter <route> address	5.2.6				x	
Support RFC-1049 Content Type field	5.2.13			x		
Use 4-digit years	5.2.14		x			
Generate numeric timezones	5.2.14		x			
Accept all timezones	5.2.14	x				
Use non-num timezones from RFC-822	5.2.14	x				
Omit phrase before route-addr	5.2.15			x		
Accept and parse dot.dec. domain literals	5.2.17	x				
Accept all RFC-822 address formats	5.2.18	x				
Generate invalid RFC-822 address format	5.2.18					x
Fully-qualified domain names in header	5.2.18	x				
Create explicit src route in header	5.2.19				x	
Accept explicit src route in header	5.2.19	x				
Send/recv at least 64KB messages	5.3.8	x				

6. SUPPORT SERVICES

6.1 DOMAIN NAME TRANSLATION

6.1.1 INTRODUCTION

Every host MUST implement a resolver for the Domain Name System (DNS), and it MUST implement a mechanism using this DNS resolver to convert host names to IP addresses and vice-versa [DNS:1, DNS:2].

In addition to the DNS, a host MAY also implement a host name translation mechanism that searches a local Internet host table. See Section 6.1.3.8 for more information on this option.

DISCUSSION:

Internet host name translation was originally performed by searching local copies of a table of all hosts. This table became too large to update and distribute in a timely manner and too large to fit into many hosts, so the DNS was invented.

The DNS creates a distributed database used primarily for the translation between host names and host addresses. Implementation of DNS software is required. The DNS consists of two logically distinct parts: name servers and resolvers (although implementations often combine these two logical parts in the interest of efficiency) [DNS:2].

Domain name servers store authoritative data about certain sections of the database and answer queries about the data. Domain resolvers query domain name servers for data on behalf of user processes. Every host therefore needs a DNS resolver; some host machines will also need to run domain name servers. Since no name server has complete information, in general it is necessary to obtain information from more than one name server to resolve a query.

6.1.2 PROTOCOL WALK-THROUGH

An implementor must study references [DNS:1] and [DNS:2] carefully. They provide a thorough description of the theory, protocol, and implementation of the domain name system, and reflect several years of experience.

6.1.2.1 Resource Records with Zero TTL: RFC-1035 Section 3.2.1

All DNS name servers and resolvers MUST properly handle RRs with a zero TTL: return the RR to the client but do not cache it.

DISCUSSION:
Zero TTL values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached; they are useful for extremely volatile data.

6.1.2.2 QCLASS Values: RFC-1035 Section 3.2.5

A query with "QCLASS=*" SHOULD NOT be used unless the requestor is seeking data from more than one class. In particular, if the requestor is only interested in Internet data types, QCLASS=IN MUST be used.

6.1.2.3 Unused Fields: RFC-1035 Section 4.1.1

Unused fields in a query or response message MUST be zero.

6.1.2.4 Compression: RFC-1035 Section 4.1.4

Name servers MUST use compression in responses.

DISCUSSION:
Compression is essential to avoid overflowing UDP datagrams; see Section 6.1.3.2.

6.1.2.5 Misusing Configuration Info: RFC-1035 Section 6.1.2

Recursive name servers and full-service resolvers generally have some configuration information containing hints about the location of root or local name servers. An implementation MUST NOT include any of these hints in a response.

DISCUSSION:
Many implementors have found it convenient to store these hints as if they were cached data, but some neglected to ensure that this "cached data" was not included in responses. This has caused serious problems in the Internet when the hints were obsolete or incorrect.

6.1.3 SPECIFIC ISSUES

6.1.3.1 Resolver Implementation

A name resolver SHOULD be able to multiplex concurrent requests if the host supports concurrent processes.

In implementing a DNS resolver, one of two different models MAY optionally be chosen: a full-service resolver, or a stub resolver.

(A) Full-Service Resolver

A full-service resolver is a complete implementation of the resolver service, and is capable of dealing with communication failures, failure of individual name servers, location of the proper name server for a given name, etc. It must satisfy the following requirements:

- The resolver MUST implement a local caching function to avoid repeated remote access for identical requests, and MUST time out information in the cache.

- The resolver SHOULD be configurable with start-up information pointing to multiple root name servers and multiple name servers for the local domain. This insures that the resolver will be able to access the whole name space in normal cases, and will be able to access local domain information should the local network become disconnected from the rest of the Internet.

(B) Stub Resolver

A "stub resolver" relies on the services of a recursive name server on the connected network or a "nearby" network. This scheme allows the host to pass on the burden of the resolver function to a name server on another host. This model is often essential for less capable hosts, such as PCs, and is also recommended when the host is one of several workstations on a local network, because it allows all of the workstations to share the cache of the recursive name server and hence reduce the number of domain requests exported by the local network.

At a minimum, the stub resolver MUST be capable of directing its requests to redundant recursive name servers. Note that recursive name servers are allowed to restrict the sources of requests that they will honor, so the host administrator must verify that the service will be provided. Stub resolvers MAY implement caching if they choose, but if so, MUST timeout cached information.

6.1.3.2 Transport Protocols

DNS resolvers and recursive servers MUST support UDP, and SHOULD support TCP, for sending (non-zone-transfer) queries. Specifically, a DNS resolver or server that is sending a non-zone-transfer query MUST send a UDP query first. If the Answer section of the response is truncated and if the requester supports TCP, it SHOULD try the query again using TCP.

DNS servers MUST be able to service UDP queries and SHOULD be able to service TCP queries. A name server MAY limit the resources it devotes to TCP queries, but it SHOULD NOT refuse to service a TCP query just because it would have succeeded with UDP.

Truncated responses MUST NOT be saved (cached) and later used in such a way that the fact that they are truncated is lost.

DISCUSSION:

UDP is preferred over TCP for queries because UDP queries have much lower overhead, both in packet count and in connection state. The use of UDP is essential for heavily-loaded servers, especially the root servers. UDP also offers additional robustness, since a resolver can attempt several UDP queries to different servers for the cost of a single TCP query.

It is possible for a DNS response to be truncated, although this is a very rare occurrence in the present Internet DNS. Practically speaking, truncation cannot be predicted, since it is data-dependent. The dependencies include the number of RRs in the answer, the size of each RR, and the savings in space realized by the name compression algorithm. As a rule of thumb, truncation in NS and MX lists should not occur for answers containing 15 or fewer RRs.

Whether it is possible to use a truncated answer depends on the application. A mailer must not use a truncated MX response, since this could lead to mail loops.

Responsible practices can make UDP suffice in the vast majority of cases. Name servers must use compression in responses. Resolvers must differentiate truncation of the Additional section of a response (which only loses extra information) from truncation of the Answer section (which for MX records renders the response unusable by mailers). Database administrators should list only a reasonable number of primary names in lists of name servers, MX alternatives, etc.

However, it is also clear that some new DNS record types defined in the future will contain information exceeding the 512 byte limit that applies to UDP, and hence will require TCP. Thus, resolvers and name servers should implement TCP services as a backup to UDP today, with the knowledge that they will require the TCP service in the future.

By private agreement, name servers and resolvers MAY arrange to use TCP for all traffic between themselves. TCP MUST be used for zone transfers.

A DNS server MUST have sufficient internal concurrency that it can continue to process UDP queries while awaiting a response or performing a zone transfer on an open TCP connection [DNS:2].

A server MAY support a UDP query that is delivered using an IP broadcast or multicast address. However, the Recursion Desired bit MUST NOT be set in a query that is multicast, and MUST be ignored by name servers receiving queries via a broadcast or multicast address. A host that sends broadcast or multicast DNS queries SHOULD send them only as occasional probes, caching the IP address(es) it obtains from the response(s) so it can normally send unicast queries.

DISCUSSION:
Broadcast or (especially) IP multicast can provide a way to locate nearby name servers without knowing their IP addresses in advance. However, general broadcasting of recursive queries can result in excessive and unnecessary load on both network and servers.

6.1.3.3 Efficient Resource Usage

The following requirements on servers and resolvers are very important to the health of the Internet as a whole, particularly when DNS services are invoked repeatedly by higher level automatic servers, such as mailers.

(1) The resolver MUST implement retransmission controls to insure that it does not waste communication bandwidth, and MUST impose finite bounds on the resources consumed to respond to a single request. See [DNS:2] pages 43-44 for specific recommendations.

(2) After a query has been retransmitted several times without a response, an implementation MUST give up and return a soft error to the application.

(3) All DNS name servers and resolvers SHOULD cache temporary failures, with a timeout period of the order of minutes.

DISCUSSION:
This will prevent applications that immediately retry soft failures (in violation of Section 2.2 of this document) from generating excessive DNS traffic.

(4) All DNS name servers and resolvers SHOULD cache negative responses that indicate the specified name, or data of the specified type, does not exist, as described in [DNS:2].

(5) When a DNS server or resolver retries a UDP query, the retry interval SHOULD be constrained by an exponential backoff algorithm, and SHOULD also have upper and lower bounds.

IMPLEMENTATION:
A measured RTT and variance (if available) should be used to calculate an initial retransmission interval. If this information is not available, a default of no less than 5 seconds should be used. Implementations may limit the retransmission interval, but this limit must exceed twice the Internet maximum segment lifetime plus service delay at the name server.

(6) When a resolver or server receives a Source Quench for

a query it has issued, it SHOULD take steps to reduce the rate of querying that server in the near future. A server MAY ignore a Source Quench that it receives as the result of sending a response datagram.

IMPLEMENTATION:
One recommended action to reduce the rate is to send the next query attempt to an alternate server, if there is one available. Another is to backoff the retry interval for the same server.

6.1.3.4 Multihomed Hosts

When the host name-to-address function encounters a host with multiple addresses, it SHOULD rank or sort the addresses using knowledge of the immediately connected network number(s) and any other applicable performance or history information.

DISCUSSION:
The different addresses of a multihomed host generally imply different Internet paths, and some paths may be preferable to others in performance, reliability, or administrative restrictions. There is no general way for the domain system to determine the best path. A recommended approach is to base this decision on local configuration information set by the system administrator.

IMPLEMENTATION:
The following scheme has been used successfully:

(a) Incorporate into the host configuration data a Network-Preference List, that is simply a list of networks in preferred order. This list may be empty if there is no preference.

(b) When a host name is mapped into a list of IP addresses, these addresses should be sorted by network number, into the same order as the corresponding networks in the Network-Preference List. IP addresses whose networks do not appear in the Network-Preference List should be placed at the end of the list.

6.1.3.5 Extensibility

DNS software MUST support all well-known, class-independent formats [DNS:2], and SHOULD be written to minimize the trauma associated with the introduction of new well-known types and local experimentation with non-standard types.

DISCUSSION:

The data types and classes used by the DNS are extensible, and thus new types will be added and old types deleted or redefined. Introduction of new data types ought to be dependent only upon the rules for compression of domain names inside DNS messages, and the translation between printable (i.e., master file) and internal formats for Resource Records (RRs).

Compression relies on knowledge of the format of data inside a particular RR. Hence compression must only be used for the contents of well-known, class-independent RRs, and must never be used for class-specific RRs or RR types that are not well-known. The owner name of an RR is always eligible for compression.

A name server may acquire, via zone transfer, RRs that the server doesn't know how to convert to printable format. A resolver can receive similar information as the result of queries. For proper operation, this data must be preserved, and hence the implication is that DNS software cannot use textual formats for internal storage.

The DNS defines domain name syntax very generally -- a string of labels each containing up to 63 8-bit octets, separated by dots, and with a maximum total of 255 octets. Particular applications of the DNS are permitted to further constrain the syntax of the domain names they use, although the DNS deployment has led to some applications allowing more general names. In particular, Section 2.1 of this document liberalizes slightly the syntax of a legal Internet host name that was defined in RFC-952 [DNS:4].

6.1.3.6 Status of RR Types

Name servers MUST be able to load all RR types except MD and MF from configuration files. The MD and MF types are obsolete and MUST NOT be implemented; in particular, name servers MUST NOT load these types from configuration files.

DISCUSSION:
The RR types MB, MG, MR, NULL, MINFO and RP are considered experimental, and applications that use the DNS cannot expect these RR types to be supported by most domains. Furthermore these types are subject to redefinition.

The TXT and WKS RR types have not been widely used by Internet sites; as a result, an application cannot rely on the the existence of a TXT or WKS RR in most domains.

6.1.3.7 Robustness

DNS software may need to operate in environments where the root servers or other servers are unavailable due to network connectivity or other problems. In this situation, DNS name servers and resolvers MUST continue to provide service for the reachable part of the name space, while giving temporary failures for the rest.

DISCUSSION:
Although the DNS is meant to be used primarily in the connected Internet, it should be possible to use the system in networks which are unconnected to the Internet. Hence implementations must not depend on access to root servers before providing service for local names.

6.1.3.8 Local Host Table

DISCUSSION:
A host may use a local host table as a backup or supplement to the DNS. This raises the question of which takes precedence, the DNS or the host table; the most flexible approach would make this a configuration option.

Typically, the contents of such a supplementary host table will be determined locally by the site. However, a publically-available table of Internet hosts is maintained by the DDN Network Information Center (DDN NIC), with a format documented in [DNS:4]. This table can be retrieved from the DDN NIC using a protocol described in [DNS:5]. It must be noted that this table contains only a small fraction of all Internet hosts. Hosts using this protocol to retrieve the DDN NIC host table should use the VERSION command to check if the

table has changed before requesting the entire table with the ALL command. The VERSION identifier should be treated as an arbitrary string and tested only for equality; no numerical sequence may be assumed.

The DDN NIC host table includes administrative information that is not needed for host operation and is therefore not currently included in the DNS database; examples include network and gateway entries. However, much of this additional information will be added to the DNS in the future. Conversely, the DNS provides essential services (in particular, MX records) that are not available from the DDN NIC host table.

6.1.4 DNS USER INTERFACE

6.1.4.1 DNS Administration

This document is concerned with design and implementation issues in host software, not with administrative or operational issues. However, administrative issues are of particular importance in the DNS, since errors in particular segments of this large distributed database can cause poor or erroneous performance for many sites. These issues are discussed in [DNS:6] and [DNS:7].

6.1.4.2 DNS User Interface

Hosts MUST provide an interface to the DNS for all application programs running on the host. This interface will typically direct requests to a system process to perform the resolver function [DNS:1, 6.1:2].

At a minimum, the basic interface MUST support a request for all information of a specific type and class associated with a specific name, and it MUST return either all of the requested information, a hard error code, or a soft error indication. When there is no error, the basic interface returns the complete response information without modification, deletion, or ordering, so that the basic interface will not need to be changed to accommodate new data types.

DISCUSSION:
The soft error indication is an essential part of the interface, since it may not always be possible to access particular information from the DNS; see Section 6.1.3.3.

A host MAY provide other DNS interfaces tailored to particular functions, transforming the raw domain data into formats more suited to these functions. In particular, a host MUST provide a DNS interface to facilitate translation between host addresses and host names.

6.1.4.3 Interface Abbreviation Facilities

User interfaces MAY provide a method for users to enter abbreviations for commonly-used names. Although the definition of such methods is outside of the scope of the DNS specification, certain rules are necessary to insure that these methods allow access to the entire DNS name space and to prevent excessive use of Internet resources.

If an abbreviation method is provided, then:

(a) There MUST be some convention for denoting that a name is already complete, so that the abbreviation method(s) are suppressed. A trailing dot is the usual method.

(b) Abbreviation expansion MUST be done exactly once, and MUST be done in the context in which the name was entered.

DISCUSSION:
For example, if an abbreviation is used in a mail program for a destination, the abbreviation should be expanded into a full domain name and stored in the queued message with an indication that it is already complete. Otherwise, the abbreviation might be expanded with a mail system search list, not the user's, or a name could grow due to repeated canonicalizations attempts interacting with wildcards.

The two most common abbreviation methods are:

(1) Interface-level aliases

Interface-level aliases are conceptually implemented as a list of alias/domain name pairs. The list can be per-user or per-host, and separate lists can be associated with different functions, e.g. one list for host name-to-address translation, and a different list for mail domains. When the user enters a name, the interface attempts to match the name to the alias component of a list entry, and if a matching entry can

be found, the name is replaced by the domain name found in the pair.

Note that interface-level aliases and CNAMEs are completely separate mechanisms; interface-level aliases are a local matter while CNAMEs are an Internet-wide aliasing mechanism which is a required part of any DNS implementation.

(2) Search Lists

A search list is conceptually implemented as an ordered list of domain names. When the user enters a name, the domain names in the search list are used as suffixes to the user-supplied name, one by one, until a domain name with the desired associated data is found, or the search list is exhausted. Search lists often contain the name of the local host's parent domain or other ancestor domains. Search lists are often per-user or per-process.

It SHOULD be possible for an administrator to disable a DNS search-list facility. Administrative denial may be warranted in some cases, to prevent abuse of the DNS.

There is danger that a search-list mechanism will generate excessive queries to the root servers while testing whether user input is a complete domain name, lacking a final period to mark it as complete. A search-list mechanism MUST have one of, and SHOULD have both of, the following two provisions to prevent this:

(a) The local resolver/name server can implement caching of negative responses (see Section 6.1.3.3).

(b) The search list expander can require two or more interior dots in a generated domain name before it tries using the name in a query to non-local domain servers, such as the root.

DISCUSSION:
The intent of this requirement is to avoid excessive delay for the user as the search list is tested, and more importantly to prevent excessive traffic to the root and other high-level servers. For example, if the user supplied a name "X" and the search list contained the root as a component,

a query would have to consult a root server before the next search list alternative could be tried. The resulting load seen by the root servers and gateways near the root would be multiplied by the number of hosts in the Internet.

The negative caching alternative limits the effect to the first time a name is used. The interior dot rule is simpler to implement but can prevent easy use of some top-level names.

6.1.5 DOMAIN NAME SYSTEM REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
GENERAL ISSUES							
Implement DNS name-to-address conversion	6.1.1	x					
Implement DNS address-to-name conversion	6.1.1	x					
Support conversions using host table	6.1.1			x			
Properly handle RR with zero TTL	6.1.2.1	x					
Use QCLASS=* unnecessarily	6.1.2.2		x				
Use QCLASS=IN for Internet class	6.1.2.2	x					
Unused fields zero	6.1.2.3	x					
Use compression in responses	6.1.2.4	x					
Include config info in responses	6.1.2.5					x	
Support all well-known, class-indep. types	6.1.3.5	x					
Easily expand type list	6.1.3.5		x				
Load all RR types (except MD and MF)	6.1.3.6	x					
Load MD or MF type	6.1.3.6					x	
Operate when root servers, etc. unavailable	6.1.3.7	x					
RESOLVER ISSUES:							
Resolver support multiple concurrent requests	6.1.3.1		x				
Full-service resolver:	6.1.3.1			x			
Local caching	6.1.3.1	x					

Information in local cache times out	6.1.3.1	x					
Configurable with starting info	6.1.3.1		x				
Stub resolver:	6.1.3.1			x			
Use redundant recursive name servers	6.1.3.1	x					
Local caching	6.1.3.1			x			
Information in local cache times out	6.1.3.1	x					
Support for remote multi-homed hosts:							
Sort multiple addresses by preference list	6.1.3.4		x				
TRANSPORT PROTOCOLS:							
Support UDP queries	6.1.3.2	x					
Support TCP queries	6.1.3.2		x				
Send query using UDP first	6.1.3.2	x					1
Try TCP if UDP answers are truncated	6.1.3.2		x				
Name server limit TCP query resources	6.1.3.2			x			
Punish unnecessary TCP query	6.1.3.2				x		
Use truncated data as if it were not	6.1.3.2					x	
Private agreement to use only TCP	6.1.3.2			x			
Use TCP for zone transfers	6.1.3.2	x					
TCP usage not block UDP queries	6.1.3.2	x					
Support broadcast or multicast queries	6.1.3.2			x			
RD bit set in query	6.1.3.2					x	
RD bit ignored by server is b'cast/m'cast	6.1.3.2	x					
Send only as occasional probe for addr's	6.1.3.2		x				
RESOURCE USAGE:							
Transmission controls, per [DNS:2]	6.1.3.3	x					
Finite bounds per request	6.1.3.3	x					
Failure after retries => soft error	6.1.3.3	x					
Cache temporary failures	6.1.3.3		x				
Cache negative responses	6.1.3.3		x				
Retries use exponential backoff	6.1.3.3		x				
Upper, lower bounds	6.1.3.3		x				
Client handle Source Quench	6.1.3.3		x				
Server ignore Source Quench	6.1.3.3			x			
USER INTERFACE:							
All programs have access to DNS interface	6.1.4.2	x					
Able to request all info for given name	6.1.4.2	x					
Returns complete info or error	6.1.4.2	x					
Special interfaces	6.1.4.2			x			
Name<->Address translation	6.1.4.2	x					
Abbreviation Facilities:	6.1.4.3			x			

Convention for complete names	6.1.4.3	x					
Conversion exactly once	6.1.4.3	x					
Conversion in proper context	6.1.4.3	x					
Search list:	6.1.4.3			x			
Administrator can disable	6.1.4.3		x				
Prevention of excessive root queries	6.1.4.3	x					
Both methods	6.1.4.3		x				
		-	-	-	-	-	--
		-	-	-	-	-	--

1. Unless there is private agreement between particular resolver and particular server.

6.2 HOST INITIALIZATION

6.2.1 INTRODUCTION

This section discusses the initialization of host software across a connected network, or more generally across an Internet path. This is necessary for a diskless host, and may optionally be used for a host with disk drives. For a diskless host, the initialization process is called "network booting" and is controlled by a bootstrap program located in a boot ROM.

To initialize a diskless host across the network, there are two distinct phases:

(1) Configure the IP layer.

Diskless machines often have no permanent storage in which to store network configuration information, so that sufficient configuration information must be obtained dynamically to support the loading phase that follows. This information must include at least the IP addresses of the host and of the boot server. To support booting across a gateway, the address mask and a list of default gateways are also required.

(2) Load the host system code.

During the loading phase, an appropriate file transfer protocol is used to copy the system code across the network from the boot server.

A host with a disk may perform the first step, dynamic configuration. This is important for microcomputers, whose floppy disks allow network configuration information to be mistakenly duplicated on more than one host. Also, installation of new hosts is much simpler if they automatically obtain their configuration information from a central server, saving administrator time and decreasing the probability of mistakes.

6.2.2 REQUIREMENTS

6.2.2.1 Dynamic Configuration

A number of protocol provisions have been made for dynamic configuration.

- ICMP Information Request/Reply messages

This obsolete message pair was designed to allow a host to find the number of the network it is on. Unfortunately, it was useful only if the host already knew the host number part of its IP address, information that hosts requiring dynamic configuration seldom had.

- Reverse Address Resolution Protocol (RARP) [BOOT:4]

  RARP is a link-layer protocol for a broadcast medium that allows a host to find its IP address given its link layer address. Unfortunately, RARP does not work across IP gateways and therefore requires a RARP server on every network. In addition, RARP does not provide any other configuration information.

- ICMP Address Mask Request/Reply messages

  These ICMP messages allow a host to learn the address mask for a particular network interface.

- BOOTP Protocol [BOOT:2]

  This protocol allows a host to determine the IP addresses of the local host and the boot server, the name of an appropriate boot file, and optionally the address mask and list of default gateways. To locate a BOOTP server, the host broadcasts a BOOTP request using UDP. Ad hoc gateway extensions have been used to transmit the BOOTP broadcast through gateways, and in the future the IP Multicasting facility will provide a standard mechanism for this purpose.

The suggested approach to dynamic configuration is to use the BOOTP protocol with the extensions defined in "BOOTP Vendor Information Extensions" RFC-1084 [BOOT:3]. RFC-1084 defines some important general (not vendor-specific) extensions. In particular, these extensions allow the address mask to be supplied in BOOTP; we RECOMMEND that the address mask be supplied in this manner.

DISCUSSION:
Historically, subnetting was defined long after IP, and so a separate mechanism (ICMP Address Mask messages) was designed to supply the address mask to a host. However, the IP address mask and the corresponding IP address conceptually form a pair, and for operational

simplicity they ought to be defined at the same time and by the same mechanism, whether a configuration file or a dynamic mechanism like BOOTP.

Note that BOOTP is not sufficiently general to specify the configurations of all interfaces of a multihomed host. A multihomed host must either use BOOTP separately for each interface, or configure one interface using BOOTP to perform the loading, and perform the complete initialization from a file later.

Application layer configuration information is expected to be obtained from files after loading of the system code.

6.2.2.2 Loading Phase

A suggested approach for the loading phase is to use TFTP [BOOT:1] between the IP addresses established by BOOTP.

TFTP to a broadcast address SHOULD NOT be used, for reasons explained in Section 4.2.3.4.

6.3 REMOTE MANAGEMENT

6.3.1 INTRODUCTION

The Internet community has recently put considerable effort into the development of network management protocols. The result has been a two-pronged approach [MGT:1, MGT:6]: the Simple Network Management Protocol (SNMP) [MGT:4] and the Common Management Information Protocol over TCP (CMOT) [MGT:5].

In order to be managed using SNMP or CMOT, a host will need to implement an appropriate management agent. An Internet host SHOULD include an agent for either SNMP or CMOT.

Both SNMP and CMOT operate on a Management Information Base (MIB) that defines a collection of management values. By reading and setting these values, a remote application may query and change the state of the managed system.

A standard MIB [MGT:3] has been defined for use by both management protocols, using data types defined by the Structure of Management Information (SMI) defined in [MGT:2]. Additional MIB variables can be introduced under the "enterprises" and "experimental" subtrees of the MIB naming space [MGT:2].

Every protocol module in the host SHOULD implement the relevant MIB variables. A host SHOULD implement the MIB variables as defined in the most recent standard MIB, and MAY implement other MIB variables when appropriate and useful.

6.3.2 PROTOCOL WALK-THROUGH

The MIB is intended to cover both hosts and gateways, although there may be detailed differences in MIB application to the two cases. This section contains the appropriate interpretation of the MIB for hosts. It is likely that later versions of the MIB will include more entries for host management.

A managed host must implement the following groups of MIB object definitions: System, Interfaces, Address Translation, IP, ICMP, TCP, and UDP.

The following specific interpretations apply to hosts:

o ipInHdrErrors

  Note that the error "time-to-live exceeded" can occur in a host only when it is forwarding a source-routed datagram.

- ipOutNoRoutes

  This object counts datagrams discarded because no route can be found. This may happen in a host if all the default gateways in the host's configuration are down.

- ipFragOKs, ipFragFails, ipFragCreates

  A host that does not implement intentional fragmentation (see "Fragmentation" section of [INTRO:1]) MUST return the value zero for these three objects.

- icmpOutRedirects

  For a host, this object MUST always be zero, since hosts do not send Redirects.

- icmpOutAddrMaskReps

  For a host, this object MUST always be zero, unless the host is an authoritative source of address mask information.

- ipAddrTable

  For a host, the "IP Address Table" object is effectively a table of logical interfaces.

- ipRoutingTable

  For a host, the "IP Routing Table" object is effectively a combination of the host's Routing Cache and the static route table described in "Routing Outbound Datagrams" section of [INTRO:1].

  Within each ipRouteEntry, ipRouteMetric1...4 normally will have no meaning for a host and SHOULD always be -1, while ipRouteType will normally have the value "remote".

  If destinations on the connected network do not appear in the Route Cache (see "Routing Outbound Datagrams section of [INTRO:1]), there will be no entries with ipRouteType of "direct".

DISCUSSION:
  The current MIB does not include Type-of-Service in an ipRouteEntry, but a future revision is expected to make

this addition.

We also expect the MIB to be expanded to allow the remote management of applications (e.g., the ability to partially reconfigure mail systems). Network service applications such as mail systems should therefore be written with the "hooks" for remote management.

6.3.3 MANAGEMENT REQUIREMENTS SUMMARY

FEATURE	SECTION	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Support SNMP or CMOT agent	6.3.1		x				
Implement specified objects in standard MIB	6.3.1		x				

7. REFERENCES

This section lists the primary references with which every implementer must be thoroughly familiar. It also lists some secondary references that are suggested additional reading.

INTRODUCTORY REFERENCES:

[INTRO:1] "Requirements for Internet Hosts -- Communication Layers," IETF Host Requirements Working Group, R. Braden, Ed., RFC-1122, October 1989.

[INTRO:2] "DDN Protocol Handbook," NIC-50004, NIC-50005, NIC-50006, (three volumes), SRI International, December 1985.

[INTRO:3] "Official Internet Protocols," J. Reynolds and J. Postel, RFC-1011, May 1987.

This document is republished periodically with new RFC numbers; the latest version must be used.

[INTRO:4] "Protocol Document Order Information," O. Jacobsen and J. Postel, RFC-980, March 1986.

[INTRO:5] "Assigned Numbers," J. Reynolds and J. Postel, RFC-1010, May 1987.

This document is republished periodically with new RFC numbers; the latest version must be used.

TELNET REFERENCES:

[TELNET:1] "Telnet Protocol Specification," J. Postel and J. Reynolds, RFC-854, May 1983.

[TELNET:2] "Telnet Option Specification," J. Postel and J. Reynolds, RFC-855, May 1983.

[TELNET:3] "Telnet Binary Transmission," J. Postel and J. Reynolds, RFC-856, May 1983.

[TELNET:4] "Telnet Echo Option," J. Postel and J. Reynolds, RFC-857, May 1983.

[TELNET:5] "Telnet Suppress Go Ahead Option," J. Postel and J.

Reynolds, RFC-858, May 1983.

[TELNET:6] "Telnet Status Option," J. Postel and J. Reynolds, RFC-859, May 1983.

[TELNET:7] "Telnet Timing Mark Option," J. Postel and J. Reynolds, RFC-860, May 1983.

[TELNET:8] "Telnet Extended Options List," J. Postel and J. Reynolds, RFC-861, May 1983.

[TELNET:9] "Telnet End-Of-Record Option," J. Postel, RFC-855, December 1983.

[TELNET:10] "Telnet Terminal-Type Option," J. VanBokkelen, RFC-1091, February 1989.

This document supercedes RFC-930.

[TELNET:11] "Telnet Window Size Option," D. Waitzman, RFC-1073, October 1988.

[TELNET:12] "Telnet Linemode Option," D. Borman, RFC-1116, August 1989.

[TELNET:13] "Telnet Terminal Speed Option," C. Hedrick, RFC-1079, December 1988.

[TELNET:14] "Telnet Remote Flow Control Option," C. Hedrick, RFC-1080, November 1988.

SECONDARY TELNET REFERENCES:

[TELNET:15] "Telnet Protocol," MIL-STD-1782, U.S. Department of Defense, May 1984.

This document is intended to describe the same protocol as RFC-854. In case of conflict, RFC-854 takes precedence, and the present document takes precedence over both.

[TELNET:16] "SUPDUP Protocol," M. Crispin, RFC-734, October 1977.

[TELNET:17] "Telnet SUPDUP Option," M. Crispin, RFC-736, October 1977.

[TELNET:18] "Data Entry Terminal Option," J. Day, RFC-732, June 1977.

[TELNET:19] "TELNET Data Entry Terminal option -- DODIIS Implementation," A. Yasuda and T. Thompson, RFC-1043, February 1988.

FTP REFERENCES:

[FTP:1] "File Transfer Protocol," J. Postel and J. Reynolds, RFC-959, October 1985.

[FTP:2] "Document File Format Standards," J. Postel, RFC-678, December 1974.

[FTP:3] "File Transfer Protocol," MIL-STD-1780, U.S. Department of Defense, May 1984.

This document is based on an earlier version of the FTP specification (RFC-765) and is obsolete.

TFTP REFERENCES:

[TFTP:1] "The TFTP Protocol Revision 2," K. Sollins, RFC-783, June 1981.

MAIL REFERENCES:

[SMTP:1] "Simple Mail Transfer Protocol," J. Postel, RFC-821, August 1982.

[SMTP:2] "Standard For The Format of ARPA Internet Text Messages," D. Crocker, RFC-822, August 1982.

This document obsoleted an earlier specification, RFC-733.

[SMTP:3] "Mail Routing and the Domain System," C. Partridge, RFC-974, January 1986.

This RFC describes the use of MX records, a mandatory extension to the mail delivery process.

[SMTP:4] "Duplicate Messages and SMTP," C. Partridge, RFC-1047, February 1988.

[SMTP:5a] "Mapping between X.400 and RFC 822," S. Kille, RFC-987, June 1986.

[SMTP:5b] "Addendum to RFC-987," S. Kille, RFC-???, September 1987.

The two preceding RFC's define a proposed standard for gatewaying mail between the Internet and the X.400 environments.

[SMTP:6] "Simple Mail Transfer Protocol," MIL-STD-1781, U.S. Department of Defense, May 1984.

This specification is intended to describe the same protocol as does RFC-821. However, MIL-STD-1781 is incomplete; in particular, it does not include MX records [SMTP:3].

[SMTP:7] "A Content-Type Field for Internet Messages," M. Sirbu, RFC-1049, March 1988.

DOMAIN NAME SYSTEM REFERENCES:

[DNS:1] "Domain Names - Concepts and Facilities," P. Mockapetris, RFC-1034, November 1987.

This document and the following one obsolete RFC-882, RFC-883, and RFC-973.

[DNS:2] "Domain Names - Implementation and Specification," RFC-1035, P. Mockapetris, November 1987.

[DNS:3] "Mail Routing and the Domain System," C. Partridge, RFC-974, January 1986.

[DNS:4] "DoD Internet Host Table Specification," K. Harrenstein, RFC-952, M. Stahl, E. Feinler, October 1985.

SECONDARY DNS REFERENCES:

[DNS:5] "Hostname Server," K. Harrenstein, M. Stahl, E. Feinler, RFC-953, October 1985.

[DNS:6] "Domain Administrators Guide," M. Stahl, RFC-1032, November 1987.

[DNS:7] "Domain Administrators Operations Guide," M. Lottor, RFC-1033, November 1987.

[DNS:8] "The Domain Name System Handbook," Vol. 4 of Internet Protocol Handbook, NIC 50007, SRI Network Information Center, August 1989.

SYSTEM INITIALIZATION REFERENCES:

[BOOT:1] "Bootstrap Loading Using TFTP," R. Finlayson, RFC-906, June 1984.

[BOOT:2] "Bootstrap Protocol (BOOTP)," W. Croft and J. Gilmore, RFC-951, September 1985.

[BOOT:3] "BOOTP Vendor Information Extensions," J. Reynolds, RFC-1084, December 1988.

Note: this RFC revised and obsoleted RFC-1048.

[BOOT:4] "A Reverse Address Resolution Protocol," R. Finlayson, T. Mann, J. Mogul, and M. Theimer, RFC-903, June 1984.

MANAGEMENT REFERENCES:

[MGT:1] "IAB Recommendations for the Development of Internet Network Management Standards," V. Cerf, RFC-1052, April 1988.

[MGT:2] "Structure and Identification of Management Information for TCP/IP-based internets," M. Rose and K. McCloghrie, RFC-1065, August 1988.

[MGT:3] "Management Information Base for Network Management of TCP/IP-based internets," M. Rose and K. McCloghrie, RFC-1066, August 1988.

[MGT:4] "A Simple Network Management Protocol," J. Case, M. Fedor, M. Schoffstall, and C. Davin, RFC-1098, April 1989.

[MGT:5] "The Common Management Information Services and Protocol over TCP/IP," U. Warrier and L. Besaw, RFC-1095, April 1989.

[MGT:6] "Report of the Second Ad Hoc Network Management Review Group," V. Cerf, RFC-1109, August 1989.

Security Considerations

There are many security issues in the application and support programs of host software, but a full discussion is beyond the scope of this RFC. Security-related issues are mentioned in sections concerning TFTP (Sections 4.2.1, 4.2.3.4, 4.2.3.5), the SMTP VRFY and EXPN commands (Section 5.2.3), the SMTP HELO command (5.2.5), and the SMTP DATA command (Section 5.2.8).

Author's Address

Robert Braden
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

Phone: (213) 822 1511

EMail: Braden@ISI.EDU