Network Working Group
Request for Comments: 1312
Obsoletes: RFC 1159

R. Nelson
Crynwr Software
G. Arnold
Sun Microsystems, Inc.

April 1992

Message Send Protocol 2

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. Discussion and suggestions for improvement are requested. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Discussion

The Message Send Protocol is used to send a short message to a given user on a given terminal on a given host. Unix's write command offers a limited form of this service through its host-local write command. This service is also known on some hosts as "SEND".

As the Internet grows, more and more people are using hosts that do not run Internet protocols at all times. These hosts may be able to use a simple protocol that can be implemented using UDP and IP. The Message Send Protocol is one such protocol.

Note that a message sending protocol is already defined using TCP. The SMTP protocol includes a "SEND" command that will direct mail to a user's terminal. SMTP's SEND is not useful in this instance because SMTP's SEND is not implemented by the majority of vendors at this time, and is difficult to use by unskilled users. For the purposes of standardization, we will include a TCP based Message Send Service.

Message Syntax

The message consists of several parts, all of which must be present The first part is a single octet indicating the protocol revision, currently decimal 66, 'B'. The remaining parts are null-terminated sequences of eight-bit characters in the ISO 8859/1 alphabet. Some parts may be empty. All comparisons of parts (e.g., recipient,

cookie, etc.) are case-insensitive. The parts are as follows:

RECIPIENT The name of the user that the message is directed to. If this part is empty, the message may be delivered to any user of the destination system.

RECIP-TERM The name of the terminal to which the message is to be delivered. The syntax and semantics of terminal names are outside the scope of this specification. If this part is empty, the "right" terminal is chosen. This is a system-dependent function. If this part consists of the string "*", all terminals on the destination system are implied. If the RECIPIENT part is empty but the RECIP-TERM is not, the message is written on the specified terminal. If both the RECIPIENT and RECIP-TERM parts are empty, the message should be written on the "console", which is defined as some place where the message is most likely to be seen by a human operator or administrator.

MESSAGE The actual message. The server need not preserve the formatting and white-space content of the message if this is necessary to display it. New lines should be represented using the usual Netascii CR + LF. (Following the Internet tradition, a server should probably be prepared to accept a message in which some other end-of-line convention is followed, but a conforming client must use CR + LF.)

The message text may only contain printable characters from the ISO 8859/1 set, which is upward compatible from USASCII, plus CR, LF and TAB. No other control codes or escape sequences may be included: the client should strip them from the message before it is transmitted, and the server must check each incoming message for illegal codes. (A server may choose to display the message after stripping out such codes, or may reject the entire message.) If the MESSAGE part is empty, the message may be discarded by the server.

SENDER The username of the sender. (This and subsequent parts were not present in version 1 of the Message Send Protocol.) This part should not be empty. A server may choose to accept, reject or ignore messages in which the SENDER part is empty.

SENDER-TERM The name of the sending user’s terminal. This part may be empty. The intention is that a recipient may reply

to a message by sending the reply to the user SENDER at terminal SENDER-TERM on the originating system. (The sender's hostname should be retrieved from the transport software.)

COOKIE — A magic cookie. This part must be present in all messages, but is only of significance for the UDP service. The combination of the sender's UDP port number and this cookie should be unique. A client may elect to transmit a particular message several times to increase the chances of its reception; a server may use the cookie and port to identify duplicate messages and discard them. A reasonable cookie is the time of day represented in a readable format. The maximum length of a cookie is 32 octets, excluding the terminating null.

SIGNATURE — A token which, if present, may be used by the server to verify the identity of the sender. The use of the SIGNATURE part is discussed further in the section on Security, below.

The total length of the message shall be less than 512 octets. This includes all eight parts, and any terminating nulls. UDP packets are limited to 512 octets.

If this protocol is changed, the revision number will be changed.

TCP Based Message Send Service

One Message Send Service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 18. Once a connection is established a message is sent by the client over the connection.

The server replies with a single character indicating positive ("+") or negative ("-") acknowledgment, immediately followed by an optional message of explanation, terminated with a null. The positive acknowledgement means that the message was successfully delivered to some user/terminal, and that the negative acknowledgement means that the message was NOT delivered to any terminal.

The positive acknowledgement message can contain information about what user and terminal the message was delivered to in the case of incomplete user/terminal fields in the message. The negative acknowledgement can contain information about WHY the message was not delivered (no such user/terminal, system failure, user doesn't accept

messages, etc).

Multiple messages can be sent over the same channel. The client should close first (the server may/should not close directly after the acknowledgement is sent) and the server may close after some timeout on the order of minutes. If the sever is unable to decode a message, or no message is received within a suitable timeout, it may close the channel (on the assumption that the sender may have formatted the data incorrectly).

UDP Based Message Send Service

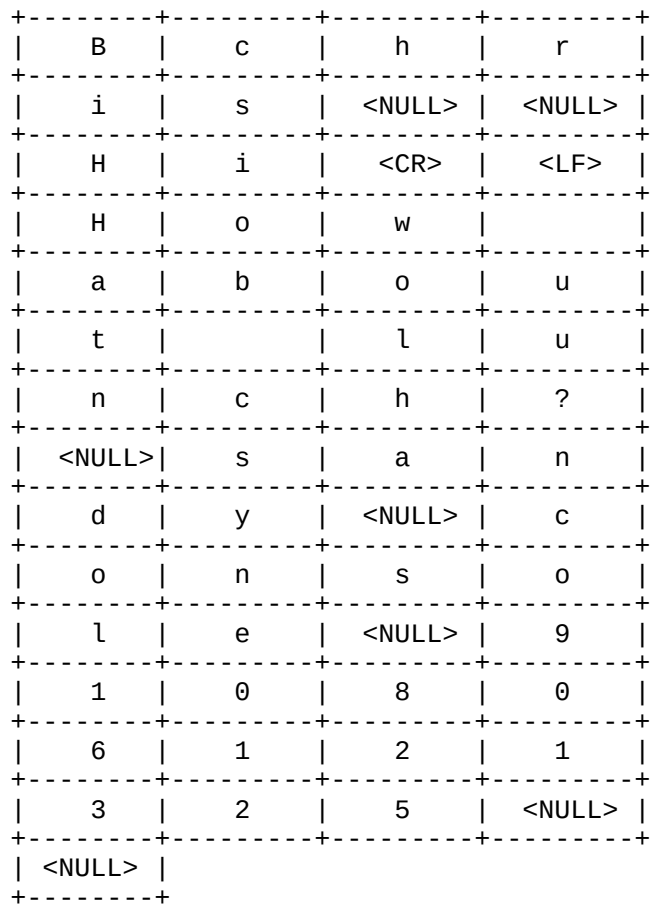
Another Message Send Service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 18. When a datagram is received by the server, an answering datagram may be sent back to the client. If the message was addressed to a particular user (i.e., the RECIPIENT part was non-empty) and was successfully delivered to that user, a positive acknowledgement should be sent (as described above). If the message was directed at any user (i.e., the RECIPIENT part is empty), or if the message could not be delivered for some reason, no reply is sent.

The reason for this policy is that the UDP service may be used to broadcast messages addressed to a particular user on an unknown system or all users on all systems. In either case, it is inappropriate for all servers to send replies. An alternative approach might have been to require that a server only send a reply if a message was addressed explicitly to that system and was not broadcast. Unfortunately, the most popular network programming API does not provide an easy way for an application to determine this; furthermore such a policy would provide no feedback to the sender of a broadcast message to a particular recipient. The approach adopted here provides a reasonable compromise.

Example of Message Encoding

Consider a situation in which the user "sandy" is logged into the console of system "alpha", and wishes to send a message to the user "chris". "chris" is known to be logged in on the system "beta" but the exact terminal is unknown. The message consists of two lines of text, "Hi" followed by "How about lunch?".

The message would be encoded as follows:



Note that the RECIP-TERM and SIGNATURE parts are empty. The COOKIE is the string "910806121325", which in this implementation indicates that the message was sent at 12:13:25 on the 6th of August, 1991. The identity if the sending and receiving systems is not included in the message; the server must obtain this information from the transport service.

Advisories

Client and server implementations must follow the character set restrictions noted in the MESSAGE part description. Failure to do so may have undesirable effects on the operation of the receiver's terminal; more seriously, it may open up a significant security

"hole". The checks must be made on any part of the message which may be displayed, including the sender's name and terminal. This is one case where the admonition to "be liberal in what you accept" is not applicable. A server may chose to apply additional checks to an incoming message, and to reject any message which may pose a security risk. For example, a system using a PostScript-based display may reject a message which might be interpreted as an executable PostScript program.

The underlying transport, whether TCP or UDP, is expected to provide checksums for the message and any response.

The semantics of the various RECIPIENT and RECIP-TERM combinations may be confusing. The introduction of the "*" wildcard designation in the RECIP-TERM part makes it possible to send a message to all terminals on the designated system (if RECIPIENT is empty), or to all terminals at which a particular recipient has logged in.

A positive acknowledgement may indicate only that the Message Send server was able to successfully invoke a local message delivery service. It may not be possible for true end-to-end semantics to be inferred.

For example, a Message Send server may employ a local delivery mechanism which calls upon the services of a window system to display the message in a pop-up window. This process may take some significant time to complete, and it is unclear whether it is useful for the server to wait for an indeterminate period before returning an acknowledgement. Therefore, this specification does not prescribe whether the acknowledgement is associated with delivery of the message to the local service, the display of the message, or confirmation by the user that the message has been read by, e.g., dismissing the pop-up window.

Security Considerations

Those who plan to implement this service must ensure that the following issues are reflected in the documentation of their products, and that their implementations include sufficient configuration controls to allow systems and network administrators to achieve the appropriate levels of usability and security.

First, this service may allow someone to write on a user's terminal without the user giving his or her permission. Where possible, users should be provided with a mechanism for disabling this.

Second, it is extremely important for implementors to observe the rules for filtering message text as discussed under Message Syntax

above. Failure to do this may introduce major security holes.

The third issue concerns the verification of the sender's identity. If the recipient is fooled into believing that a message is from a particular user, various security issues may arise. For example, the recipient may send a reply containing confidential material.

This service is primarily intended for "open" environments: controlled local area networks used by reasonably trusted participants, in which security considerations may be relaxed in the interests of ease of use and administration. In such an environment it is appropriate to trust the user name and source IP address as identifying the actual sender of the message.

Within more security-conscious environments, this assumption is probably unacceptable. As has been widely noted, there is no way within the current Internet architecture to ensure that the source address of an IP datagram is correct. Hence it is entirely possible for someone to spoof the IP address.

The obvious, and simplest, answer is to disallow the use of this protocol in such situations. However a more constructive approach is to incorporate within the protocol some mechanism by which a server can reliably identify the sender.

In this version of the protocol specification, we define a SIGNATURE part within a message. If this part is empty, the identity of the sender cannot be verified, and the server implementation may elect to reject all such requests. If the part is not empty, it is treated as a case-insensitive text encoding of some security token. This RFC does not define the encoding or interpretation of this token. We expect that such matters will form part of future RFCs on security and privacy issues; at an appropriate time, this RFC will be re-issued to include references to these RFCs.

Acknowledgements

PostScript is a trademark of Adobe Systems, Inc.

Authors' Addresses

Russell Nelson
Crynwr Software
11 Grant St.
Potsdam, NY 13676

Phone: (315) 268-1925
EMail: nelson@crynwr.com

Geoff Arnold
Sun Microsystems, Inc.
2 Federal Street
Billerica, MA 01821

Phone: (508) 671-0317
EMail: geoff@east.sun.com