4.2 Mail Server Protocols

The mail server responds with PUP type "iAmEcho" (type 2) to PUP's of type "echoMe" (type 1) and length no more than 128 bytes sent to it on socket 54B.

Any Grapevine mail server will accept messages submitted to it by the following protocol, and will deliver them to the specified recipients, according to the delivery semantics defined in section three. This *mail submission protocol* allows a sequence of *commands* on a Byte Stream Protocol connection. The mail server is usually willing to establish such a connection in response to an RFC packet sent to the mail server on socket 56B. The commands each consist of the client sending a word representing an operation, probably followed by some *arguments*, then the server generally sends some *results*. Not all commands are allowed at any time. The restrictions on the order of the commands are described in terms of the *state* of the stream; violation of these restrictions is a protocol violation. The state may be *idle*, *started*, *noItem*, or *inItem*. The state is initially "idle".

StartSend: operation = 20, arguments = [sender name, password, return-to name, boolean]

> This command is allowed only if the state is "idle". The password should be that of the sender. At present, the server ignores the password, but checks that the sender name would be valid as a recipient name; we reserve the right to start checking the sender password some time in the future, without notice. The server checks that the return-to name would be valid as a recipient. The boolean should be TRUE iff the client wishes the server to validate the recipient names during the connection. The result is a byte, with the following values. 0 means everything is ok. 1 means the sender password is incorrect (not checked at present). 2 means the sender name is invalid. 3 means the return-to name is invalid. 4 means that the server could not validate some name because of communication problems. If this byte is 0, then the state of the connection becomes "started".

AddRecipient: operation = 21, arguments = [name]

> This command is allowed only if the state is "started". The name is added to the list of recipients for this message; there is no result.

CheckValidity: operation = 22, no arguments.

> This command is allowed only if the state is "started". If the boolean argument of the StartSend command was TRUE, then for each recipient which appears to be invalid the server sends a number specifying which recipient it was (counting the calls of AddRecipient, from 1) followed by the name of the invalid recipient, and these recipients are removed from the recipient list of the message. Then (regardless of the value of the boolean) the server sends the number 0 followed by a number indicating how many names remain in the message's recipient list. The state of the stream becomes "noItem".

StartItem: operation = 23, arguments = [word]

> If the state of the stream is "inItem", then the current message body item is terminated and its length calculated and the state of the stream becomes "noItem". The state of the stream must now be "noItem". The word specifies the type of a message body item, and the state of the stream becomes "inItem". The acceptable types of message body item have been described in section three, in connection with the message delivery semantics. There is no result.

AddToItem: operation = 24, arguments = [number, sequence of bytes]

> This command is allowed only if the state is "inItem". The number specifies how many bytes are in the sequence. The bytes are appended to the current message body item. There is no result.

Send: operation = 26, no arguments.

> This command is allowed only if the state is "inItem". The current message body item is terminated (as in StartItem), and all the data associated with the message is recorded in stable storage. The server commits to deliver the message. The result is an ack. The state of the stream becomes "idle".

Expand: operation = 27, argument = [name]

> This command is allowed at any time and does not affect the state of the stream. If the name would be treated by the mail servers as a distribution list during the delivery algorithm (including a foreign distribution list) then for each name in that list, the server sends a boolean TRUE followed by the name. Then a boolean FALSE is sent. Then a byte is sent. The value of this byte is: 0 if the name was treated as a distribution list, 1 if the name would be an invalid recipient, 2 if the name would be an individual recipient (including foreign mail system recipients), 3 if the mail server could not decide because of communication failures. The intent of this command is to allow a client to show a user the potential contents of a distribution list.

The following protocols allow a client to inspect and modify the state of an in-box on a mail server. Mail arrives in an in-box as the major effect of the mail server delivery algorithm. Note that an individual may have several in-boxes, defined by the individual's mailbox-list in the naming database, and any mail retrieval package should arrange to inspect all of the individual's in-boxes. The in-box mechanism is complicated by facilities (*TOC entries* and *deleted messages*) which are designed to assist a dumb terminal system to provide a user with temporary access to mail in the user's in-boxes. The intent is that a user should retrieve all messages from the in-box and flush the in-box when software with secondary storage is available to the user. Keeping substantial amounts of mail in an in-box for substantial periods of time will degrade the performance of the mail servers. The TOC mechanism allows a client to associate a *TOC entry* (represented by a remark, i.e. a string of up to 64 characters) with each message in an in-box. Provision is also made for a client to *delete* individual messages from an in-box; a deleted message still occupies a place in the in-box until the in-box is *flushed*, but deleting the message frees the resources used by the message body and property list. No TOC entry may be associated with a deleted message. If there is no in-box on this server for an individual, then the protocol provides the illusion that there is an in-box containing no messages.

The mail server accepts PUP's of type "mailCheckLaurel" (type 214B) and length no more than 128 bytes on socket 54B. If the PUP contains the characters of a name, and this mail server has a non-empty in-box containing messages for a recipient of that name (even if they are all deleted messages, see below), then the mail server replies with a PUP of type "mailIsNew" (type 211B); otherwise it replies with a PUP of type "mailNotNew" (type 212B). Note that the server does not respond to PUP's of type "mailCheck" (type 210B). Note that the server does not check whether the naming database presently indicates that this server is in the user's mailbox-list. This polling protocol is intended to allow a client to inspect the state of an individual's in-box without going to

the expense of establishing a Byte Stream Protocol connection.

An in-box may be accessed after establishing a Byte Stream Protocol connection in response to an RFC packet sent to a mail server on socket 57B. The state of this stream is described as one of *idle*, *open*, or *inMessage*, and there are restrictions on which commands may be used depending on the state. Violation of these restrictions is a violation of the protocol. The state is initially "idle". Each command consists of the client sending a word representing an operation, possibly followed by some arguments, then the server sending some results. The commands allow the client to open a mailbox, then sequentially inspect or modify the messages in it; the order of the messages is approximately that in which they were sent, and the order does not change between sessions; messages are not added to an in-box while a client has it open; only one client may have a particular in-box open at one time. A client, having opened an in-box, may flush it: this causes the in-box to be empty. If a client wishes to close an in-box without flushing it, the client must terminate the connection.

A mail server may occasionally use a remote file server for storing contents of in-boxes; if this has been done and the file server is unavailable, the mail server will close the client's mail retrieval connection arbitrarily (sorry!). The location of such a remote file server is of no concern to a client of the mail retrieval protocol, but has been described in section three.

OpenInBox: operation = 0, arguments = [name, password]

> The state must be "idle". The result is a byte with the following values: 1 if the name is for a group, 2 if the name is for an individual and the password is correct, 3 if the name is not registered in the naming database, 4 if the name could not be checked because of communication failures, 5 if the password is incorrect. If this byte has the value 2, then the state becomes "open". The byte is followed by a word, which should be ignored.

NextMessage: operation = 1, no arguments.

> If the state is "inMessage", it becomes "open". The state must now be "open". The result is a sequence of three booleans. The first is TRUE iff there is another message in the in-box, and this becomes the current message; if this boolean is FALSE the others are undefined. The second is TRUE iff the message is *archived*; this is a hint to the client that access to the message may involve access to a remote file server; we recommend that you indicate this fact to your user. The third is TRUE iff the message is deleted. If the message is not deleted, then the state becomes "inMessage".

ReadTOC: operation = 2, no arguments.

> The state must be "inMessage". If there is a TOC entry associated with this message, the result is a remark. Otherwise the result is an empty string.

ReadMessage: operation = 3, no arguments.

> The state must be "inMessage". The result is a sequence of message body items, representing the message's property list followed by the message body. Each item has the following format: a number, which is the item's type, followed by a long number, which is the item's length, followed by that number of bytes, followed by an extra byte if the length was odd (thus each item occupies an even number of bytes). The items sent are: an item of type "postmark" (whose bytes are the timestamp of the message's property list), an item of type "sender" (whose bytes are the sender name of the message's property list), an item of type "return-to" (whose bytes are the return-to name of the message's property list), an

item of type "recipients" (whose bytes are the recipient names of the message's property list), the items that constitute the message body in the order provided by the submitting client, and an item of type 177777B. The item of type 177777B is of length 0. The items are followed by a Byte Stream Protocol *mark* byte. The value and meaning of the property list item types is given earlier in the description of the message delivery semantics.

WriteTOC: operation = 4, argument = [remark]

The state must be "inMessage". The remark is associated with the message as a TOC entry, (replacing any earlier TOC entry for this message); if the remark is the empty string, no TOC entry is associated. The result is an ack.

DeleteMessage: operation = 5, no arguments.

The state must be "inMessage". The current message is permanently deleted from this in-box (as is any associated TOC entry). The result is an ack. The state becomes "open".

Flush: operation = 6, no arguments.

The state must not be "idle". The in-box is emptied. The result is an ack. The state is now "idle".

5. A Mesa Grapevine client interface

This section describes informally the interface provided to clients programming in Mesa by the Cedar *GrapevineUser* package. The purpose of this section is not primarily to document that package for its clients, but to indicate how the Grapevine naming database may be used to provide an interface that makes transparent such network effects as the replication and distribution of services, and the failure of particular instances of services. If the reader is intending only to be a client of the Cedar package, then this section should be read briefly; precise specification of the package is in the public definitions files for the package. If the reader intends to implement an equivalent package (in Mesa or another language), then I strongly recommend perusal of the source files of both the definitions and implementations of this package. All files concerned with this package are available on [Indigo]<Cedar>Grapevine>*.mesa and *.bcd.

GrapevineUser provides several public interfaces. These are named *GVNames*, *GVSend*, and *GVRetrieve*. The GVNames interface provides access to most of the naming database enquiry and update operations; GVSend provides for submission of messages; GVRetrieve allows a client to read messages from in-boxes. The *GVBasics* interface defines some data types common to the other interfaces. Each of these interfaces uses the naming database to provide an interface that is independent of particular computers or network addresses. A client of GrapevineUser never is concerned with such things as host names.

In the following descriptions, the precise declarations found in the definitions files are not repeated: the reader should look in the sources of those definitions files.

The GVNames interface provides for database enquiries and updates. The results of these operations are a return code, represented by some subset of the enumerated type Outcome, and in some cases a list of names. Each operation may give a return code "allDown", to indicate that because of communication failures the requested operation could not be performed; this indicates that after its best efforts to contact any of the multiple suitable registration servers, the package had failed. None of the operation raises a SIGNAL. The operations each have an obvious mapping into the commands described in the registration server protocol.

The transparency provided by the GVNames interface is achieved as follows. Generally, the GVNames implementation caches a stream to some registration server. When it is asked to perform an enquiry about some name, the implementation uses the cached stream to attempt the enquiry. If this stream fails (because that registration server is now unavailable), or if this enquiry gives a "WrongServer" return-code, then the implementation attempts to locate a suitable registration server. It does this by using a resource location interface (described below) to find a network address of some server which knows about the name in question. That is, if the name in question is of the form *x.reg*, the implementation tries to locate a server in the group named *reg.gv*. If this succeeds, such a server should be able to answer the enquiry; if this fails because *reg.gv*" is not a group, then the name is invalid; if this fails because none of those servers is available then the return code given is "allDown".

The GVSend interface implementation proceeds in a similar manner. Each of its procedures has a straightforward mapping into a command specified in the mail submission protocol. The GVSend implementation attempts to cache the network address of a suitable mail server, but if this fails (or initially) the implementation uses the resource location interface to locate a server in the group "MailDrop.ms". Only if this resource location fails is mail submission not possible.

The GVRetrieve interface is more complicated, because it provides a client with access to all of the client's in-boxes. The implementation determines the sites of a client's in-boxes by using the Expand database enquiry. The interface also uses the single PUP polling protocol to obtain hints about the state of the in-boxes. When a client wishes to inspect mail, the implementation establishes connections only to those in-boxes which have indicated they are non-empty during the polling protocol; this is important to minimize the connection load on the mail servers, since a client's secondary in-boxes will almost always be empty. The GVRetrieve interface will also provide clients with access to their mailbox on a foreign mail server (using the MTP protocol), and will function satisfactorily in an environment where there are no Grapevine servers, only IFS mail servers. To determine whether it is in a non-Grapevine environment, the GVRetrieve implementation considers whether the client's registry is registered in the PUP Name Lookup Server database; if the registry is registered there and maps to precisely *one* network address, then the registry is assumed to be implemented on an IFS mail server (at that network address) with no aid from Grapevine; otherwise the Grapevine servers are used.

The GrapevineUser package uses the following algorithm to perform resource location. The resource location interface is provided with the name of a group in the naming database. It reads the members of this group, giving it a list of names of potential servers. For each of these, it reads their connect-site from the database. The implementation then sends a PUP of type "echoMe" to each of these servers. For each server which replies with an "iAmEcho" PUP, the implementation calls back to its caller, offering the network address of that server. In this manner, the caller of this interface is provided with network addresses of potential instances of the service named by the given group, and the caller can attempt to establish a connection with these, in turn, until a satisfactory one is found. The "echoMe" mechanism is intended for efficiency: the servers are tried in order of their responsiveness, and each one tried is responding to PUP's, so an attempt to connect will be resolved rapidly.

There are several optimizations used within the GrapevineUser implementation, and the potential implementor is strongly recommended to inspect the GrapevineUser implementations.

6. Administrative Facilities

The major administrative facility in Grapevine is the naming database. Most system administration can be done by performing updates to that database, and an interactive program, called *Maintain*, is available to perform these updates. Maintain is documented (partially, at the present time) elsewhere.

In addition, there are some facilities provided by Grapevine to monitor and affect the state of the computers which constitute Grapevine. These facilities change from release to release of the software, so this section is likely to be slightly incorrect at any time. These facilities are defined precisely only by their implementation. The facilities in question are: a local terminal interface, an FTP server, a Telnet server, and a log file.

The local display shows various statistics about the state of the servers. These statistics are the same as those available through the Telnet server "Display Statistics" command. There is also a short typescript, which is of use only to wizards. When the local keyboard and mouse have not been touched for a minute, the display reverts to plain white, with a cursor slowly traversing near the top of the screen; moving a key or the mouse causes the statistics to reappear.

The local keyboard is used only when a server is initialized or restarted. When a server is initialized for the first time, it will determine its own name, ask you to type its password, and arrange to fetch copies of the appropriate database entries. When a server is restarted, it will verify its password (asking for it to be re-typed only if this verification fails), and if necessary will alter its connect-site in the naming database. Note that registration servers and mail servers cohabiting a single machine must have distinct names (typically, the same simple-name, with registries "gv" and "ms"). The case of initializing the first server in the world is necessarily special - consult an expert for details.

Each Grapevine server provides an FTP server which allows access to files on the Grapevine server's local disk filing system. Clients of this FTP server must provide credentials corresponding to a valid name and password of an individual in the Grapevine database. The files which support the naming database and message buffering are not accessible. To be permitted to read the file "GV.log", the client must be a member of the group "LogReaders^.ms". To be permitted to access other files, the client must be a member of the group "Transport^.ms". The FTP server supports enumeration of files, with the string "*" meaning all files.

The log file contains single-line entries recording various events as they occur in the server. The file is named "GV.log" and is 120 Alto pages long. The file is used as a circular buffer. Each cycle round this buffer may also be written to backing files on an IFS file server. For a server whose mail server has a name of the form *x.ms*, the individual *Log-x.ms* should have a connect-site whose value is a string of the form *[host]<path>*. This will cause the log files to be written cyclically to 40 files with names of the form *<path>x-00!1* through *<path>x-39!1* on the file server *host* using the FTP protocol. For example, if "Cabernet.ms" is a mail server and the individual "Log-Cabernet.ms" exists and has connect-site "[Ivy]<DMS>Log>", then files such as "<DMS>Log>Cabernet-09!1" will be stored on the file server "Ivy". If the required *Log-x.ms* individual does not exist, no attempt will be made to write the log files to a file server; the individual is examined only when the server is restarting, and is not revisited during normal running.

The Telnet server (known as the "Viticulturists' Entrance") provides various facilites to a remote terminal user. Some of these facilities are privileged: they are allowed only to an individual who has logged in, and who is a member of the group "Transport.ms" or whose name is "Wizard.gv" and they are only available after using the "Enable" command. Most commands can be stopped by

typing DEL. The facilities change frequently, but at present are as follows.

Add-Registry:

Interacts with the user to perform the steps needed to add a registry to a running registration server.

Display Histograms:

Types histograms of various events. At present only mail retrieval delays.

Display Inboxes:

Types a summary of the state of clients' in-boxes on this server.

Display Other-Servers:

Types this server's view of the availability of other servers that it knows about.

Display Policy-Controls:

Types information about the various internal server controls on operations. This includes whether the operation is presently "allowed", how many instances of the operation are allowed simultaneously, the highest number that have occurred simultaneously, and the total number that have occurred. See also the "Set Policy-Controls" command.

Display Queues:

Types a summary of the queues of not-yet-delivered messages maintained by the mail server.

Display Statistics:

Types various statistics kept by the server, including server version and uptime.

Display Time:

Types the local time of day.

Enable:

Allows use of the privileged commands if the logged in user is "Wizard.gv" or is a member of the group "Transport.ms".

Force Archive:

Forces transfer of the contents of an in-box to a backing file server. The choice of backing file server is described below.

Force Background-Process:

Forces immediate activation of one of the periodic processes in the server. The "Archiver" process scans inboxes looking for ones which should be written out to a remote file server (this normally hapens about 11 p.m. each evening); the "ReadPending" process considers those messages which are on the pending queue because there was nowhere to deliver them (this normally happens every 15 minutes); the "RegPurger" process scans the naming

database looking for information representing dead entries or removed members of lists which is more than 14 days old and can be removed from the disk (this normally hapens about 11 p.m. each the evening).

Force MSMail-Login:

Causes the process that reads mail server internal mail to login to GrapevineUser afresh, causing it to reconsider the location of its inboxes.

Force RSMail-Login:

Causes the process that reads registration server internal mail to login to GrapevineUser afresh, causing it to reconsider the location of its inboxes.

Force Purge:

Asks for an R-Name. If this R-Name corresponds to a dead entry in the naming database, removes that entry immediately (without waiting for the normal 14 day timeout). This allows immediate re-use of that name, but is incorrect unless all copies of the appropriate registry throughout Grapevine know that the entry was dead.

Login:

Asks for name and password, and attempts to authenticate the name by use of the naming database.

Maintain:

Enters the Maintain program, which allows maintenance of the naming database.

Quit:

Terminates this connection.

Restart:

Asks for an Alto executive command line to place in the local file "Rem.cm", asks for a comment to write in the log, asks for more confirmation, then stops the server (which returns to the Alto executive, which will execute the command line from "Rem.cm").

Set Archive-Days:

Alters the timeout used by the in-box archiver process. This alteration applies only to the next run of the archiver, then the timeout reverts to its default 7 days.

Set Policy-Control:

Alters the state of various internal server controls from "allowed" to "not allowed", or *vice versa*. Setting the control to be "not allowed" prevents the operation in question from being started subsequently; it does not affect operations currently in progress. The controls form a hierarchy; all appropriate levels in the hierarchy must be "allowed" for an operation to be permitted. The controls in question are:

Work: must be allowed for any operation to be allowed

Connection: controls incoming connections other than the Viticulturists' Entrance.

ClientInput: controls mail submission connections

ServerInput: controls mail forwarding connections from other Grapevine servers
ReadMail: controls mail retrieval connections.
RegExpand: controls registration server enquiry connections.
Lily: controls Telnet server connections to any local Lily server.
MTP: controls MTP server connections from clients.
FTP: controls FTP server connections from clients.
Telnet: controls Viticulturists' Entrance connections.
MainLine: controls the following four operations:
ReadInput: controls processing messages queued for delivery.
ReadPending: controls processing messages from the "pending" queue.
ReadForward: controls forwarding message to other servers.
Remailing: controls remailing of messages from inboxes.
Background: controls the following operations:
RSReadMail: controls reading registration server database update messages
MSReadMail: controls reading mail server in-box re-mail requests.
Archiver: controls the transfer of in-boxes to remote file servers.
RegPurger: controls the nightly clean-up of the naming database.

Wait-until-idle:

Waits until the only activity in the server is this Telnet connection. This wait may also be terminated by typing DEL.