The Construction of a TCP/IP to Apple Event Gateway for use in Distributed Computing Experimentation

Christopher Haupt, Rochester Institute of Technology

Abstract

Apple event aware applications open the possibility of sharing unique, application specific resources with other Apple event speaking programs. Because Apple events are a System 7 specific feature, it is not possible to use these resources from non-Macintosh platforms without major alterations of the Apple event aware applications. This paper provides a summary of a project to allow non-Mac platforms access to Apple event applications via a protocol that facilitates the transfer of pseudo-Apple events over TCP/IP. The TCP/IP to Apple event protocol is discussed, as is an application that implements gateway functionality on the Macintosh platform. To facilitate the use of the gateway, a client API is introduced. This paper documents the first implementation of the protocol and proposes some future enhancements. Two current experiments making use of the gateway are described illustrating possible uses of the protocol.

1. Introduction

Why a TCP/IP to Apple Event Gateway?

The group of high-level events collectively known as Apple events define a standard body of event signals that can be used in a wide variety of inter– and intra–application communication situations (IAC). Defined within the *Apple Event Registry*, hundreds of Apple events are described in this lengthy developer tome. Activities such as controlling the Finder, requesting graphics operations, or accessing tables of data are but a small sampling of the breadth of event types.

A limiting factor to wide spread use of Apple events is that they are an exclusive feature of the System 7 Macintosh operating system. As such, they can not be directly used to communicate with Macs running older OS software, nor can they generally be used with other vendors' operating systems. Additionally, if two System 7 Macs are not connected via an intervening AppleTalk network–or bridged in some manner–it is not possible to communicate via Apple events.

It is with these issues in mind that the TCP/IP to Apple Event Gateway project was conceived. The TCP/IP to Apple Event (TCPAE) Gateway package defines a simple protocol by which machines running the TCP/IP network protocol stack may issue pseudo-Apple event signals to gateway machines.

The TCPAE package includes a client application programming interface (API) for TCP machines and a gateway server[1] application for Macintosh host machines. It is the gateway application's responsibility to convert pseudo-Apple events into the real thing and dispatch them to their appropriate destinations.

Structure of this Paper

This paper provides an overview of the TCPAE protocol project. This paper is not a tutorial on the use of Apple events or TCP/IP. For that, I would refer you to the bibliography as a good starting place, especially [2][4][5][8][9][14][17]. I wouldn't presume to be prepared to give a tutorial as this work is of an experimental nature at the time of this writing.

Part 2 describes the details of the TCPAE protocol. A short discussion regarding the significance of Apple events is followed with some details centering around the choice of TCP/IP as the primary networking protocol for the gateway. Lastly, part two describes the actual TCPAE protocol, including the layout of the data packets necessary to communicate with the gateway.

Part 3 provides an overview of the construction of the gateway. A brief functional specification is provided and the implementation decisions that were required to bring the specification to reality are mentioned. The section closes with a description of second version enhancements that are planned for the gateway project.

Part 4 defines the client API package. The version one API is implemented as an object code library enabling quick linkage with custom programs without having to worry about writing special networking code. The current API functions are documented in this section and some of the proposed enhancements are also described.

Part 5 provides a glimpse at two of the current experiments to exercise the gateway which are in progress at RIT.

The paper provides some concluding remarks in part 6.

2. The TCPAE Protocol

Why use Apple Events?

With the creation of System 7, Apple Computer has defined an accessible high level event protocol with a stable[2] dictionary of common, sharable events [2]. Dubbed the *Apple Event Interprocess Messaging Protocol* or *Apple events* for short, the protocol defines the mechanism by which applications can share data and trigger events within themselves, other programs on the same machine, or on Macs

[1]A note on the words *gateway* and *server*. In this paper, the two terms are used interchangeably to mean the TCPAE gateway application that resides on the Macintosh. If they are used otherwise and the meaning isn't clear by context, I will attempt to make it clear explicitly.

[2]By stable, I mean mostly stable. Apple event suites have been known to change slightly, some (a la Finder) will probably change in the future, and others are not in official existence yet.

across an AppleTalk connected network.

While System 7 and Apple events portended to a world of software that easily takes advantage of services provided by others, until recently this dream hasn't been the case. Only within the last half year or so have developers begun to release truly Apple event savvy programs.

With these releases, the power of interapplication communication has begun to become apparent. Apple events enable software features which allow for the easy use of the specialities of various packages from outside of their native environments. Scripting from such applications as Hypercard and Frontier is opening the door in the Macintosh world to simple, automated, and remote task management in a generalized way.

Hindering some of these possibilities are two major barriers: 1) Apple events are a Macintosh System 7 specific feature, and as such are confined to that environment; 2) Apple events take advantage of AppleTalk protocols as their underlying transport. Without special software or hardware connections for AppleTalk networks, it isn't possible for Macs to send events across other wide area networks. Worse, it is difficult–or near impossible–to effectively communicate and/or send Apple events between applications residing on hardware and operating system platforms other than Macintosh.

Why TCP?

The *TCP/IP Internet Protocol Suite* (TCP/IP) provides a nearly universal set of network service protocols. TCP/IP has been implemented on practically all major computing platforms. As such, it provides a rich set of options for writing client applications that can communicate with Macintosh Apple event-aware resources.

The Transmission Control Protocol (TCP) provides guaranteed, ordered peer to peer delivery of data. As such, it is extremely useful for setting up virtual circuits between two applications when reliability is the most desirable attribute of the network connection.

The TCPAE protocol

The TCPAE protocol defines the preferred message format used to communicate between the TCPAE gateway application and its clients. The current version of the protocol is heavily influenced by TcpPlay [16] as are the TCP portions of the client and gateway code.

This section describes the general format of TCPAE packets and illustrates common message formats for the primary gateway functions implemented in version one of the protocol. A client API which hides implementation detail is provided to communicate with the gateway. However, there may be situations where a programmer wishes to implement his or her own TCP client[3] code. See the later sections of this paper for a discussion detailing the general implementation.

TCPAE packets are constructed in the same way regardless of their source and destination. Each packet is preceded with an identifying header and is followed by message specific data if applicable.

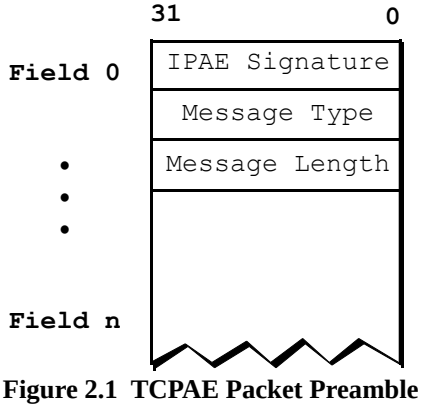

Figure 2.1 TCPAE Packet Preamble

Figure 2.1 is a diagram of the preamble layout. The packet header begins with a longword signature of the protocol. Four bytes encode `IPAE` into the Mac OSType format. This signature has two purposes. First, it acts as a simple identification tag to help the gateway filter

3

out extraneous packets that may be sent to its connection socket from other applications. Second, it serves as a synchronization mechanism to mark the beginning of each transaction between client and server.

Immediately following the signature is a longword value which encodes the packet content type. The packet type is primarily used to instruct the gateway or client on how to decode the attached data stream. The type can also be thought of as the command that is being sent to the remote host for interpretation.

The last field of the header is a longword value that supplies the length in bytes of the following data stream. This value should be zero if the command type does not require supplemental data[4]. The TCPAE protocol does not impose any special structure upon attached data and does not require any special terminating sequence at the end of the data buffer being sent. It only requires an accurate buffer count.

Version one of the protocol implements three packet command types: *status*, *reply*, and *Apple event.*

• The *status* command, denoted by the constant `kStatusCmd` in the API, can be sent from client to server or server to client. Its purpose is to request common information and statistics from the query target. Currently, the standard reply to the status command includes the implemented version of the TCPAE protocol, the security options in effect, and the number of transactions handled since image activation. The Figure 2.2 illustrates a status packet.

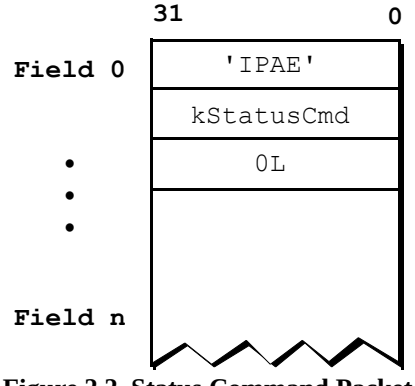

Figure 2.2 Status Command Packet

• The *reply* type is denoted with the symbol `kReplyCmd`. This packet is used to communicate the results of a previous command back to the originator of that command. Figure 2.3 shows the packet format.

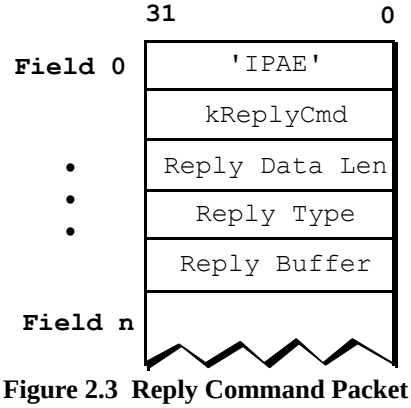

Figure 2.3 Reply Command Packet

The data buffer attached to the packet header can have a variable length dependent upon the type of reply. The first long word after the header fields is the reply type. Usually this field will contain the command type that this packet is responding to. Following the reply type is a variable number of bytes that contain the actual reply. The length of this buffer is obtained by taking the header message length field and subtracting four (for the longword sub-type field). An example of a reply packet answering back from a status command is shown in figure 2.4.

[4]For the curious, if you specify a number other than zero and send data to the gateway when not necessary, the gateway will pull the data off the socket but then discard it.

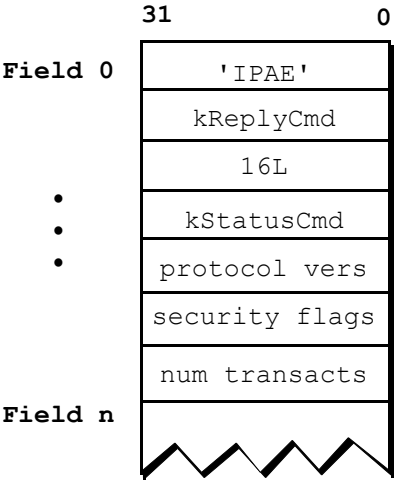

Figure 2.4 A Reply to a Status Command

• The final packet type is used for *Apple event* command packets. Denoted symbolically by `kAECmd`, this is the packet that is at the heart of the gateway system. It represents the Apple event that is to be dispatched on the gateway Macintosh. The version implementing protocol one can have a variable data buffer size, but the buffer is always constructed with the same first four initial fields after the standard header. First is a longword holding an AEKeyword for the class of the Apple event. Second is also a longword cast AEKeyword holding the Apple event identifier. Third is a field containing a longword with the method by which the target for the Apple event is specified. Fourth is the address of the Apple event target and is of variable length. Additional fields are possible and are dependent upon the Apple event class and event id. See figure 2.5 for an illustration of this packet type.

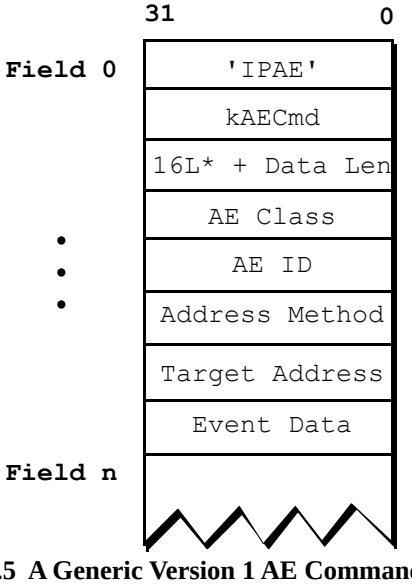

Figure 2.5 A Generic Version 1 AE Command Packet

Version one of TCPAE is limited to sending Apple events to programs by their application signature. This method is symbolically denoted with `kByAppSig` within field three of the packet buffer and with an OSType encoded signature (a longword) in field four. Other addressing techniques are planned for future protocol releases. Figure 2.5 places an asterisk next to the 16L constant in the attached message length field to reinforce that in this version of the protocol the buffer will be minimally 16 bytes in length. (Four each for the event class, identifier, addressing type constant, and the OSType application signature–more if event data follows.)

Currently, TCPAE optimizes the use of three Apple events: Open Application[5], Quit Application, and Do Script. While the TCPAE protocol is sufficient to describe any event, only these three events can be automatically dispatched by the gateway. This limitation will be lifted with the addition of the `kAEBuildCmd` planned for the next gateway release (see below).

[5]The protocol fakes an OAPP by sending to the Finder an Open Selection event with the target application as the selected item. After launching the app, the Finder sends an OAPP to the target.

3. The TCPAE Gateway

The Version 1.0 Gateway

The TCPAE gateway application constructed to implement version one of the TCPAE protocol has been created to meet a certain minimal functionality level. This section describes the specification for the gateway and describes how its various functions are generally implemented.

The TCPAE gateway is built to act as a conduit between network clients running the TCP/IP protocol stack and a Macintosh running System 7 with MacTCP. Clients can send requests via the TCP protocol which are subsequently converted into corresponding Apple events by the gateway. Version one of the gateway application specifically interprets Open Application, Quit Application, and Do Script events.

Version one is a single threaded server thereby effectively limiting the number of simultaneous connections to one at a time. Most transactions between client and server are not time consuming, so throughput concerns have been ignored for this version.

The gateway application is small and requires little resources (usually less than 100K of RAM). The application is a suitable candidate for launching from within the Startup Items folder. It has a simple interface which is useful for monitoring current transactions and for creating, modifying, and deleting security parameters (see figure 3.1 below).

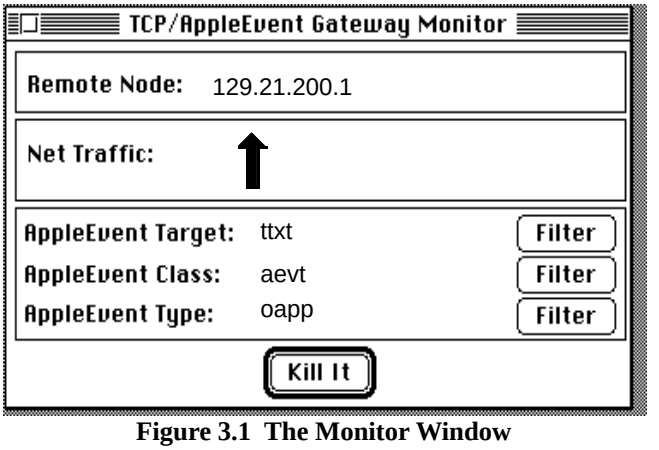

Figure 3.1 The Monitor Window

General Gateway Structure

The gateway can be represented by a simple finite state machine. Figure 3.2 illustrates the network activity loop of the application. When not initializing or shutting down, the TCPAE gateway is generally in one of four main states. First, it opens a passive listen TCP socket and awaits a client connection. Second, upon receiving a connection request, it verifies that the sender is indeed an authorized client and then buffers the client's command request. Third, once a command is completely received, it is sent to an interpreter for execution. It is within the interpretation step that the command is parsed and an Apple event is built and posted (assuming an Apple event command). The last general state is a reply state in which any command results are transmitted back to the client.

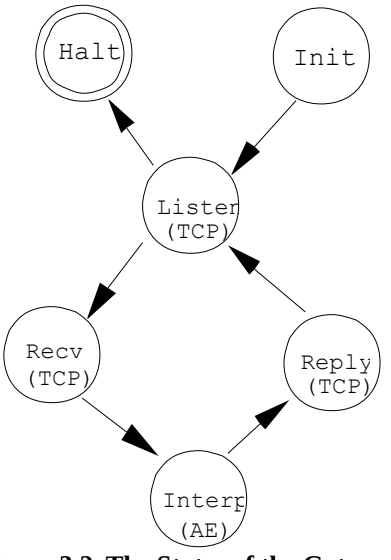

Figure 3.2 The States of the Gateway

The TCP state machine as outlined above is implemented using the MacTCP driver library of routines [5]. The diagram indicates in which of the four states TCP driver and Apple Event Manager calls are used. After TCP initialization, a `TCPPassiveOpen` call is made by the gateway on a stream previously setup by a `TCPCreate` call in the initialization step. The passive open listens on the TCPAE port (port number 1200 is temporarily being used as the listening socket). An asynchronous notification routine (ASR) registered with the TCP driver is called when activity is detected on the socket. An active open call from a client, errors, timeouts, or TCP urgent requests may trigger the ASR. The ASR implemented in this project maintains a set of counters and global flags to help signal state changes during normal Macintosh event loop processing. When a connection is detected, the gateway will transfer from a listening state to a receiving state.[6]

The receiving state has several steps. First, the TCPAE packet preamble will be read in and examined to ascertain that this is indeed: 1) a valid packet, 2) a valid command type, and 3) in sync. The third step is necessary in the cases where a garbled transmission may occur (which TCP tends to prevent) or more likely, to guard against problems of Endian mechanics.[7] It is the client's responsibility to transmit its data in the proper ordering for the Macintosh to understand. Packets that are out of order will often have their headers jumbled. The gateway simply ignores these packets. As TCP/IP defines a network standard byte order using the Big Endian style [8], a developer must primarily be concerned with his or her own data as encapsulated by TCP and not the network level packetization problem.

After the receiving state accepts a packet header, it finishes reading in the attached data buffer if one exists and is still outstanding. The gateway next makes an initial security check to verify that the client host is allowed to make connections. If this check fails, the connection is severed.

On successful completion of reading the entire packet, the command and its associated data are passed on to the interpretation state. The interpreter disassembles the command and its data.

• A *status* command triggers the immediate construction of a reply buffer with a dump of current global values storing the protocol version, security flags, and activity counter. The data is passed to the reply TCP code and the state is changed, strangely enough, to the reply state.

• An incoming *reply* command type is currently not used by the gateway application and is quietly discarded. The state immediately changes back to the passive listen state.

• The *Apple event* command directs the interpreter to assess the class and event id of the request. The interpreter first determines if this is an event for which it has a constructor, and if so, it examines the active security to see if this is an event it is allowed to send–checking event class, id, and target address.

[6]Throughout this discussion, it is implicitly understood that errors and TCP urgent data requests are special events and they will put the state machine in an exception mode.

[7]Sounds impressive, but is a hardware problem. Some architectures read their most significant bytes from low to high memory addresses–so called Big Endian–and others from high to low–the Little Endians. Macs using the Motorola 68K family of processors are Big Endians.

In this version of the gateway, the Open Application, Quit Application, and Do Script events have built in *constructors*. Constructors are functions which are optimized to build `AppleEvent` data structures from supplied client data.

Listing 3.1 below provides a sample constructor framework for the Do Script event. In this simple example, the Do Script constructor sets the `kAENoReply` flag on the outgoing Apple event. This flag tells the target application to not bother returning a Reply Apple event. Reply Apple events are used by an event receiver to return minimal result codes and messages from the Apple event handling mechanism. The constructor in listing 3.1 will only return the numeric status codes for the `AESend` Apple Event Manager call within the gate and not the results of the target application–this is an example of a "send and forget" constructor; a method in which the success of the event's execution is not tracked.

The Do Script constructor built in to the TCPAE gateway is slightly more complex than the example detailed in the listing. In the current version of the gateway, Reply Apple event messages are requested and their contents, if any, are returned to the client. The current implementation does not examine additional, supplementary Apple events that the target application may want to return to a sender. See the gateway futures discussion below for more on that matter.

The constructor for Do Script and Quit Application are nearly identical. The constructor for Open Application has to be created differently as this event is normally sent by the Finder when it launches an application. The recommended mechanism for a program to launch another application under System 7 is to send the Finder an Open Selection event with the application to be launched as the selected object [2][4]. Version one of the gateway does indeed use this method–the address resolution code maps the application signature to the proper Alias and File System Specification records through judicious use of `PBCatSearch` and other Desktop Database tricks. However, some concern has been raised for the case where Finder is not running[8]. This has yet to be investigated.

Constructors include the mechanism by which the Apple event target address is resolved. Currently, addressing is limited to the use of application signatures. An unfortunate side-effect of this design choice is that Apple events can not be posted to applications across an AppleTalk network–the target must be resident on the same machine as the gateway.

If the target is found and legal, and all necessary parameters for the desired event are at hand, the Apple event is created and dispatched.

Reply Apple events or status codes returned by `AESend` (or errors returned by other Apple Event Manager routines) are encapsulated in a reply packet and are passed on to the client via the gateway reply state code.

The reply state transmits the reply packet that may have accumulated data during previous gateway states. It is possible for the reply packet to be empty, in which case is simply acts as an acknowledgement marker for the client.

[8]Like how does the Finder launch the app when the Finder isn't there to receive the Open Selection to begin with. The temporary answer being implemented in an interim gateway is to use the Process Manager and `LaunchApplication` during that special case.

```
OSErr SendDOSC(OSType signature, long length, char * aScript)
{
OSErr             sendStat;
AEAddressDesc     theAETarget;
AppleEvent        theAE, replyAE;
long              theIntlLen;
Ptr               theIntlBuf;

// note that TCPAEStatus is a simple error handler that has been used here to
// hide the clutter of the typical checks composed of lots of if-thens,switches,
// etc...
//
// address the ae
TCPAEStatus(AECreateDesc(typeApplSignature, (Ptr) &signature,
                         sizeof(OSType), &theAETarget));
// create the ae
TCPAEStatus(AECreateAppleEvent(kAEMiscStandards, kAEDoScript,
                  &theAETarget, kAutoGenerateReturnID,
                  kAnyTransactionID, &theAE));
//embed the data
// PARAM 1
// build international string
TCPAEStatus(BuildIntlBuffer(length, aScript, &theIntlLen, theIntlBuf));

// the direct param is a buffer of type IntlText
TCPAEStatus(AEPutParamPtr(&theAE, keyDirectObject, typeChar,
                                    theIntlBuf, theIntlLen));

// send the ae
sendStat = AESend(&theAE, &replyAE, kAENoReply + kAENeverInteract,
                         kAENormalPriority, kAEDefaultTimeout,nil,nil);

// clean up the descriptor chain we created
TCPAEStatus(AEDisposeDesc(&theAETarget));
TCPAEStatus(AEDisposeDesc(&theAE));

// allow a connectionInvalid during phase 1 of the app
return((sendStat == connectionInvalid) ? noErr : sendStat);
} /*SendDOSC*/
```

Listing 3.1 Simple Do Script Constructor and Dispatcher

Security

The issue of server machine security has been addressed in a simple and straightforward manner for the first version of the gateway.

Remote host access control and filter sets provide two passive mechanisms to limit the source of incoming client connects and the kind and target of remote commands.

When access control is enabled, all incoming network requests are put into a pause state until explicitly permitted to continue by a user located at the gateway machine. This version of access control is not selective, see gateway futures below regarding that feature.

Filters and filter sets provide a higher granularity of control on the kinds of activity permitted on the gateway machine. Filters are specific instances of an entity which are not allowed to be used when creating an Apple event[9]. Filters can be specified for certain Apple event classes/suites, for event ids, or for event target applications (specific app signatures, in this version).

Filtered items can be manually entered or chosen from a predefined list. Figure 3.3 is a diagram of a filter choice dialog for Apple event suites/classes. It shows two Suites which will be not accepted by the gateway: the standard Miscellaneous Suite and a custom suite with the type `GAME`. The custom suite `RNDR` is in the process of being added.

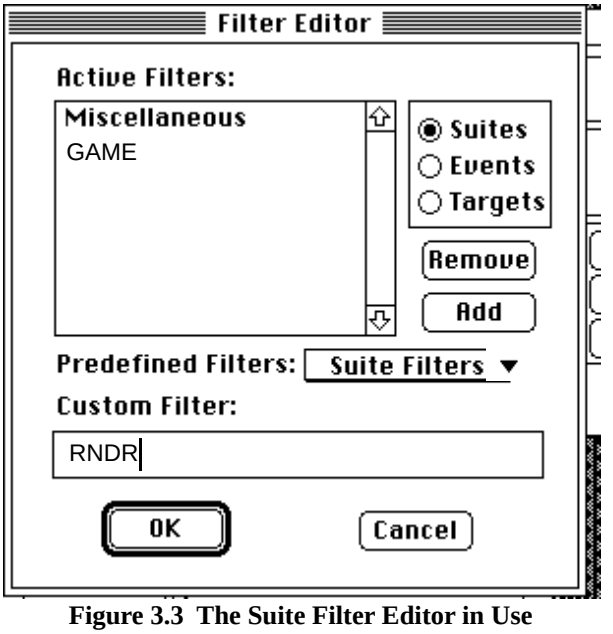

Figure 3.3 The Suite Filter Editor in Use

Filter choices can be saved in filter sets. A filter set is simply a predefined set of filters. Filter set documents can be placed in the System 7 Startup Items folder to enable the initial execution of the gateway with a set of filters immediately in place.

Gateway Futures

TCPAE has been designed to allow simple expansion without significant reworking of the protocol. The gateway application documented in this paper is the result of a first stab to implement the early communication protocol. It is a proof of concept if you will.

Several enhancements have been proposed and work to implement them will continue to occur in the foreseeable future. This sub-section outlines some of the proposals in no particular order of priority.

Support for Enhanced API

One of the first improvements will be to support the proposed enhancements for the client API. See section 4 below for a description of

[9]In the first version of the gateway it has been desirable to keep security rather simple. In the case of filters, the choice to filter out events (as opposed to specifying what *is* allowed) was made primarily because there are few events that we want to normally prevent. Future versions of the gateway may have the option of working with filters in either mode.

these changes.

Jens Peter Alfke's AEBuild

The current implementation is obviously limited by the small number of built-in event constructors. The original plan for a next version was to include additional built-ins, possibly implementing the rest of the Required and Core suites. Further work after that would go the route of drop in code resources so developers could then implement their own suites as desired.

In the progress of researching Apple events, the utility function AEBuild [1] was discovered. This compact routine takes as its parameters a textual representation of an Apple event and parses them into the descriptor records necessary for that particular event. The end result is an `AppleEvent` record structure ready for posting.

By including this function in the gateway, a considerable amount of work will not be reinvented. By adding a `kAEBuildCmd` to the TCPAE protocol, the client can then send the textual representation of any event and it will be constructed on the fly at the gate.

Paranoia Security Mode

The current access control scheme has been intended to serve during testing and within a small network environment. In an environment such as RIT, where you have the challenge of both an inquisitive student population plus live links to the Internet, it is likely that gateways left unattended would soon become subjects of some level of unauthorized access.

It is desirable to leave the gateways up and running on unattended Macs (see section 5 below).

The first improvement to access control will be the ability to specify *trusted nodes*–computers from which unlimited access is allowed. A follow on feature would be the ability to specify actual username and password combinations, possibly tying in the Users and Groups function of System 7 or by using the upcoming features of OCE.

Filters could be tied to the above scheme in a way such that specific remote nodes or users would be subject to certain filter sets.

Scripting Wires

As Apple event aware programs continue to emerge, scripting environments will become more and more useful for controlling day to day activity.

Hypercard, Frontier, application specific scripting languages, and the future AppleScript environment will certainly be among the regular collection of many users' tools.

To this end, the gateway itself needs to be enhanced such that its own functions are available to scripting languages. The first experiment into this realm will probably entail the addition of Frontier "wires" to the gateway.

Active Process Logging
The gateway monitor only displays the current transaction. It would be advantageous to have the ability to log all activity that passes through the server. Such data could include messaging statistics, client addresses, types of events and their targets, etc.

Multi-Threaded Server

As mentioned in the first part of this section, the gateway is single threaded. As MacTCP can support multiple concurrent channels between the Mac and different clients, it would be useful to be able to support many incoming and outgoing event request simultaneously. This would be especially important on those machines that maintain some popular service or data source.

Better Reply Messaging

The current gateway only returns rudimentary acknowledgements and Reply Apple event messages. Perhaps in conjunction with the ability of working in the reverse direction outlined below, a more sophisticated return messaging protocol needs to be put into place.

The Other Direction

Version one of the gateway is really a monodirectional gateway. The packets that get sent back to clients are limited to status messages,

acknowledgements, and some data.

The first enhancement will be to allow for unlimited transference of data back to the client. This is obviously necessary to support events whose purpose is to provide the client with results of a nontrivial nature.

The second natural enhancement would be to allow gateways to communicate with each other across a TCP link. By adding some kind of remote communication Apple event to the Mac's vocabulary, it would be possible for applications to transparently communicate with each other across TCP/IP networks.

A further out possibility would be to provide a gateway like application on non-Macintosh platforms to which Mac gateways can send events. The remote gateway would construct platform specific commands and then execute them. In this way, the user gets a kind of remote shell capability on the other machine.

4. TCPAE Gateway Client API

The Version 1.0 API

In this section, the client API that implements version one of the TCPAE protocol will be described. The API is designed to shield the object library user from the details of the protocol and any communication issues between the client hardware and the gateway Macintosh.

The API is currently implemented for the DEC Ultrix and Sun OS operating systems.

All of the functions below return an integer value as a function result. An error result represents the values returned by the Apple Event Manager, PPC Toolbox, and other Mac OS managers. A `noErr` is returned on success.

```
TCPAEOpenGate(int *context,
            char *gateAddr)
```

`TCPAEOpenGate` causes the API to establish a link between the client application and a gateway described by the TCP/IP address supplied in the `gateAddr` variable. The routine returns an unique id for the new session in the `context` variable. The context id is used in subsequent calls to specify a particular gateway. It is conceivable for a client to have multiple live links open concurrently.

```
TCPAECloseGate(int context)
```

`TCPAECloseGate` performs a clean shutdown of the link specified by the `context` variable.

```
TCPAEGateStatus(int context,
            StatusRec *buf)
```

`TCPAEGateStatus` queries the gateway specified by `context` for statistics and various state information. Version one returns a `StatusRec` buffer shown below.

```
typedef struct _StatusRec {
short protoVers; // vers of protocol
short filterFlags;// event filtering
                  //  active? which?
long transCnt;    // number of packets
                  // served via gateway
} StatusRec;
```

```
TCPAESendOAPP(int cntx,
            char *target)
```

`TCPAESendOAPP` requests the gateway specified by `cntx` to trigger the Finder into sending an Open Application event–in other words, it will cause the launching of the application specified by `target`. In this version, `target` is the application's signature, specified by a four character sequence of alphanumerics. The API handles the proper translation of this string format to the longword

utilized by the Mac.

As described in the previous section, this call implements the event by a sending the Finder an Open Selection event with the selected object being the target, if found. If more than one application with the same signature resides on the gateway system, the application found first in the desktop database of the target Mac will be selected.

```
TCPAESendQUIT(int cntx,
            char *target)
```

`TCPAESendQUIT` is the converse of the `TCPAESendOAPP`; it sends a Quit Application Apple event to an application currently running on the gateway system with the signature designated by `target`[10].

```
TCPAESendDOSC(int cntx,
            char *target,
            char *script)
```

`TCPAESendDOSC` constructs a simple version of the Do Script Miscellaneous Suite event. The event is posted to the application running with the signature specified by `target` on the link given by `cntx`. The `script` variable is a null terminated string containing a sequence of commands (a script) native to the target application. This script is stuffed into an International Text buffer when it arrives at the gateway as outlined for Do Script [2]. Results of the script execution, if any, are buffered by the gateway for transmission back to the client only if they come back via the default Reply Apple event. This shortcoming is in the process of being "repaired."

API Futures

As with the gateway application, the client API has a number of near term enhancements and additions planned. These are described in no particular order. All of these improvements require a corresponding change or addition within the gateway application.

```
TCPAESendDOSC(int cntx,
            char *target,
            int scriptType,
            char *scriptBuf)
```

The enhanced Do Script call would add the ability to specify the kind of Do Script event to be built, choosing either one that embeds a script within an International Text buffer or one that specifies a script file local to the gateway machine by the use of an Alias record. In the case of an Alias record, the `scriptBuf` variable would contain the path and filename of the desired script.

```
TCPAESendBuild(int cntx,
            char *target,
            char *buf)
```

`TCPAESendBuild` would use Alfke's AEBuild [1] library routine on the description supplied in `buf` to create an on-the-fly event to be addressed by application signature (or whatever addressing method is supported).

```
TCPAEGetTargets(int cntx,
            int targCount,
            BrowseRec*targBuf)
```

`TCPAEGetTargets` would implement a simple, non-interactive process browser. It would return `targCount` number of records defined by `BrowseRec` within `targBuf`.

```
typedef struct _BrowseRec {
short nameLen;   // len of progName
char  *progName;// program name
long  signature;// program sig
```

[10]If more than one application is running with the same signature, then the gateway behaves in a nondeterministic manner, depending instead on the behavior of the Apple Event Manager, AppleTalk, and the Process Manager for selecting a target. This problem is true for all events handled with the current addressing scheme.

```
long  type;     // prog type
} BrowseRec;
```

These triplets of program information (minimally triplets, they could be larger order tuples) would contain program name, signature, and type for the active processes running on the gateway machine. Future information could possibly include process serial numbers, network information, etc. In general, `TCPAEGetTargets` could be used as a kind of PPC Browser for remote clients searching for Apple event targets.

5. TCPAE Gateway in Use

Working with Remote Macs

The TCPAE Gateway opens the possibility of remotely controlling Macintosh computers across very wide area networks. With an ever growing number of applications that understand and make good use of Apple events, it is possible to take advantage of these programs in a faceless[11] manner.

As a hypothetical example, suppose that a word processing program that has decent spelling and grammar checking capabilities could be remotely instructed to start up, read in some text, spell check the document, and produce an annotated version of all errors discovered. These results could then be transmitted back to the client for further use, perhaps even integrating the change listing in with another, different word processor.

Such an example supposes an application that is Apple event aware to a high degree. We don't have to wait for applications to be released with this level of sophistication, however. By using current scripting and macro language utilities–such as Frontier, Hypercard, or even QuickKeys–it is possible to write specialized linking scripts that the TCPAE Gateway can call upon to facilitate communication.

For instance, at RIT we have regular disk-server backup jobs that run on a nightly basis. These jobs do not back up local Macintosh hard disks, instead relying on the user to copy or store important files to the server via a network copy operation (via FTP, DECNet copy, or other mechanisms). By creating a Macintosh script in Frontier which automates the copying process, it is possible to trigger that script remotely by the nightly backup job through the gateway. Because temporary storage space on the disk-server platform fluctuates, timed jobs within Frontier may fail due to insufficient disk space. Instead of having many Macs constantly attempting and then failing copy operatings over and over again, the server's backup job can request Macs to transmit data when it has an appropriate amount of storage free.

Although the above problem can be solved in a number of different ways, it illustrates just one of the interactive/non-interactive tasks that you can accomplish using the gateway and other off-the-shelf applications. The Open Application, Quit Application, and Do Script events offer a sufficient amount of flexibility for initial experimentation.

Distributed Computing Experimentation

Of even more interest is using the TCPAE Gateway to facilitate the use of custom task modules. A current research project is under way at RIT to build an inexpensive, distributed rendering and animation system.

As is often the case in large computing environments, many workstation and personal computer class machines are networked together and can communicate through an universal protocol. Commonly these farms of workstations often see little activity during certain portions of the day. Like many other academic environments, RIT's network is unified by TCP/IP and is relatively quiet in the early morning hours.

The focus of this particular research project is to use idle workstations together to create a large, unified graphics environment, possibly modeled on a Linda tuple-space-like paradigm [7].

Initially, each machine contains small, faceless applications that are very specific to single tasks. Some may be optimized for graphical transformations while others for different rendering operations such as ray tracing. A master application on a single workstation creates discrete chunks of work of a particular task type and then releases them into the environment. After the computation engines finish with their tasks, they send the results back to the master system and wait for more work. The master system assembles the results into images

[11]By faceless, I mean that you don't have to be in front of the primary console of the machine, but rather you can use the major functionality of the program without having to use a graphical user interface.

or parts of an animation. The goal of this work is to build a system in which a simple, yet realistically rendered animation can be constructed in short order and displayed in near real-time with new frames that are supplied constantly by the distributed environment.

With the large interest in and quantity of Macintoshes at RIT, it is a logical choice as one of the platforms for the graphics environment–especially with the large number of machines that are idle in public computer labs at night. Our current efforts include building some of these rendering engines such that they are Apple event aware. In this way, we can send work requests to the engines via the TCPAE gateway while using high-end graphics workstations for the master display units. By using Apple events, we can also make use of the graphics capabilities of the renderers from within Macintosh specific applications. The beauty of using this method is that the API hides all of the complexity of writing TCP code at both the Mac and client ends.

Two near term goals are to complete some non-trivial renderings of biological objects partially modeled with L-systems [15] and to examine ways of implementing transformation specific engines using some of the matrix concepts proposed in [10][12] and [13]. These projects may very well be the subject of yet another paper.

6. Conclusions

The TCPAE gateway can provide a simple, yet powerful mechanism by which computers running the TCP/IP protocols can make use of Apple event aware applications on Macs that are connected to a common network. This scheme is seen to be especially powerful when non-Macintoshes can take advantage of Macs that provide specific services via Apple events that are normally only available to other Macs.

This paper has provided an overview of the TCPAE Gateway project that I have been working on at RIT. It has described the basic elements of the TCPAE protocol. It has shown how protocol packets are constructed and what they are used for. The paper next examined the gateway application implementation–describing its current functionality and some of the proposed enhancements under review. The client API was documented next. This API provides a set of communication tools which effectively hide the mechanics of TCP connections. Lastly, a couple of the current experiments that are using the TCPAE gateway at RIT were discussed.

I feel that the TCPAE gateway is an enabling technology. It provides a conduit through which two other applications communicate where that would not normally be possible without specific code changes. I hope that others may find uses for the gateway and client software and look forward to hearing about these activities.

Bibliography

[1] Jens Peter Alfke. *Apple Events: The AE Builder/Printer*. Apple Computer, Inc., Cupertino, California. 1991.

[2] Apple Computer, Inc.. *Apple Event Registry*. Apple Computer, Inc., Cupertino, California. 1992.

[3] Apple Computer, Inc.. *Macintosh Apple Event Object Support Library*. Apple Computer, Inc., Cupertino California. 1991. (Developer Note–DTS)

[4] Apple Computer, Inc. *Inside Macintosh, Volumes I - VI*. Addison-Wesley Publishing Company, Inc., Reading Massachusetts.1985-1991.

[5] Apple Computer, Inc.. *MacTCP Programmer's Guide*. Apple Computer, Inc., Cupertino, California. 1991.

[6] Claris Corporation. *Hypercard New Features Guide*. Claris Corporation, Santa Clara, California. 1991.

[7] *C-Linda Reference Manual*. Scientific Computing Associates, Inc., New Haven, CT. 1990.

[8] Douglas Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architecture (Volume 1)*. Prentice Hall, Englewood Cliffs, NJ. 1991.

[9] Douglas Comer & David Stevens. *Internetworking with TCP/IP: Design, Implementation, and Internals (Volume 2)*. Prentice Hall, Englewood Cliffs, NJ. 1991.

[10] Thomas Cormen, Charles Leiserson, & Ronald Rivest. *Introduction to Algorithms*. McGraw Hill Book Company, New York. 1990.

[11] Donn Denman, Laura Hamersley, et. al.. *Macintosh Apple Event User Terminology Resources*. Apple Computer, Inc., Cupertino California. 1991. (Developer Note–DTS)

[12] Andrew Glassner, Ed.. *Graphics Gems*. Academic Press, Inc., San Diego, California. 1990.

[13] F. Thomas Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, California. 1992.

[14] Gary Little & Tim Swihart. *Programming for System 7*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts.1991.

[15] Przemyslaw Prusinkiewicz & Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York.1990.

[16] Bill Stafford, Malcolm Slaney, & Richard Tsoi. *TcpPlay*. Advanced Technology Group, Apple Computer Inc., Cupertino California. 1990. (Source code)

[17] Andrew Tannenbaum. *Computer Networks*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey. 1981.

The primary reason that comes to mind is in the situation where you may want to use the gateway from a client platform on which the API isn't currently implemented. I would urge you to let me know if you actually do implement a different version as I can act as a clearing house for this kind of work.