TCP/IP and OSI Interoperability with the X Window System

Nancy Crowther, Joyce Graham – IBM Cambridge Scientific Center

ABSTRACT

Network users are faced with the problem of making the transition from TCP/IP applications to the emerging Open Systems Interconnection (OSI) protocols. To accomplish this goal, these users must rewrite their code to use OSI, or switch to new applications, or use a gateway between TCP/IP and OSI based applications. This paper details how this problem was solved in work at IBM's Cambridge Scientific Center for one distributed application, the X Window System, and how the same methods could be used for other applications. The draft ANSI standard mapping X to OSI is explained. The changes that were made to the X Window System to support OSI and an X TCP-OSI Gateway are described. The best method for migrating applications was found to be extensions to the socket to support OSI at multiple layers.

Introduction

Interoperability on a network connecting different types of multi-vendor systems is a feature increasingly demanded by users. The X Window System, Transmission Control Protocol/Internet Protocol (TCP/IP), and Open Systems Interconnection (OSI) protocols all promote this interoperability. X allows the workstation user to display results from applications running on multiple heterogeneous systems. Both TCP/IP and OSI protocols are non-proprietary ways to connect these multi-vendor systems, but users who want to change from TCP/IP to OSI in order to use new OSI applications face the problem of converting their current TCP/IP applications to run on the OSI network.

This paper examines one TCP/IP-based application, the X Window System, and explains how we moved it into an OSI environment in experimental prototypes at IBM's Cambridge Scientific Center. X over OSI is a natural pairing to address requirements for enhanced interoperability, but until recently has not been attempted. We explain the mapping of X to OSI as defined in the draft ANSI standard [ANSI91], and describe how we implemented it in two different ways on a UNIX based operating system, IBM's AIX 3.1. Our experience indicates that there are many advantages to modifying the OSI socket programming interface found in 4.3 Reno BSD [BSD43] to support OSI at the Application Layer. We also describe an X TCP-OSI Gateway, for use in networks containing some TCP/IP-based systems and some OSI-based systems.

X Window System

The X Window System is a portable network-transparent window system, allowing multiple applications, called X clients, to run on varied heterogeneous systems and architectures throughout a network and display on any workstation running an X server application. The X server controls the workstation's bitmap display, keyboard and mouse. IBM ships X on most of its platforms, either client application libraries, server, or both, depending on the system.

Clients and server communicate by means of the X protocol, which consists of requests from the client for various functions such as drawing, replies from the server to these requests, events which notify the client of mouse or keyboard input, and error notifications. The X protocol in turn rides on top of a reliable byte stream between client and server. If client and server are on the same system, this reliable byte stream is simply some local interprocess communication mechanism. When client and server are on different systems connected by a network, the reliable byte stream is provided by some communications protocol, typically TCP/IP. All of IBM's current X products use TCP/IP as the underlying network communication protocol.

Figure 1 shows the various parts of an X Window System.

X client application programs use various "toolkits", or application programming interfaces (API's) which implement graphical functions. The toolkits in turn call routines in the various X libraries supplied by the X Window System. The lowest level X library, referenced eventually by all higher layers, is the Xlib, or X library. Buried in this library are the routines which perform the network communication. These communication routines, as distributed in the X source code, currently support TCP/IP and DECnet only. The window manager is also an X client, and it uses the Xlib to perform network communication to the X server. A client in frequent use is the *xterm* program, a terminal emulator. This program displays in a window as

if it were the system console. All other programs which write to the console and read from the keyboard can be run "in" this window, sending their output to *xterm*, which in turn communicates through Xlib to the X server.

The X clients and the X server may be executing on the same workstation, or on completely different systems from multiple manufacturers. Since the X protocol is system independent, clients may be written with assurance that they can display on any workstation implementing a standard X server. Similarly, a standard X server can expect to be able to be used by any standard X client from any software vendor. This of course contributes greatly to the goal of interoperability. Applications written for a proprietary windowing system or other user interface are very limited in their use.

Although the X protocol is designed to be able to be implemented from its documentation, in practice vendors use the sample implementation source code, which is available for free from the Massachusetts Institute of Technology X Consortium, and modify it for their own systems. The X source code (in the C language) is written in such a way that it can be compiled for many different operating systems by selecting certain compilation options. The X code is truly portable. Although it was implemented first on 4.2 BSD it runs on operating systems as diverse as VM, MVS, and OS/2, as well as UNIX based operating systems such as AIX. In the current Release 5 from the X Consortium, code which specifically supports AIX 3.1 and one of the graphics adapters for the RISC System/6000 is included. (This code was written by IBM and donated to the X Consortium.) This support is thus freely available to RISC System/6000 users even though it is not yet available as an IBM product.

OSI

While TCP/IP protocols, whose development began in the mid-1970s funded by the Defense Advanced Research Projects Agency (DARPA) for its collection of networks, are the current de facto standard, the long-term replacement for these protocols is the OSI protocols. This suite consists of seven layers of international standard protocols being

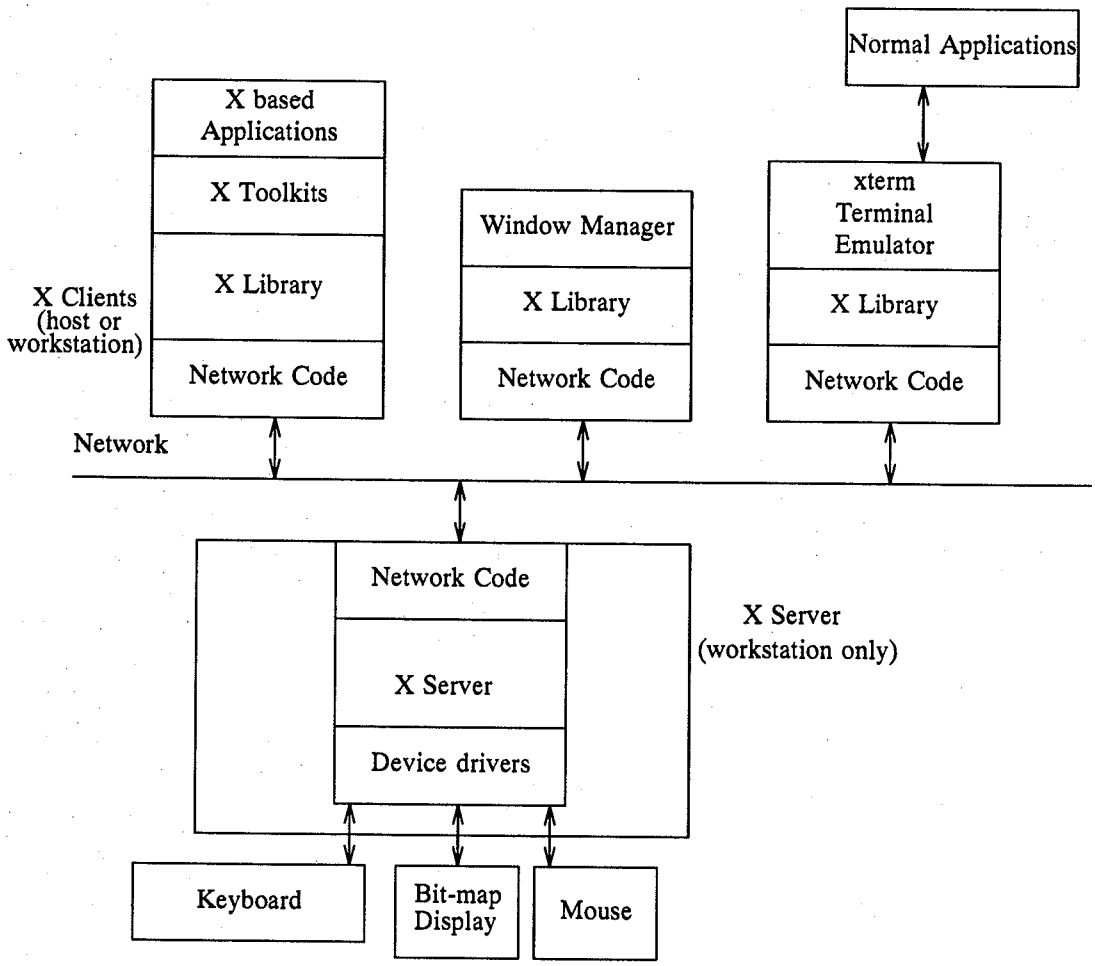

Figure 1: X Window System Parts

developed by the international community to provide both the political and technical solution to world-wide networking. Since August 15, 1990, the US Government has required that new network procurements and major upgrades to existing networks support the Government OSI Profile (GOSIP). GOSIP is a selected subset of the OSI protocols [GOSIP88].

OSI is a complete family of protocols, separating the functions of communication between applications into well-defined layers. Figure 2 shows the OSI reference model [IS7498]. There are two end systems communicating with each other, connected by any number of intermediate systems.

At the highest layer, the Application Layer, applications such as file transfer, mail, and library information retrieval exchange information organized into previously agreed upon data structures. These applications use common application services. One of these is the Association Control Service Element (ACSE). The ACSE manages the connection (called an "association") between the communicating systems. The two sides agree on which application services will be used in the association by exchanging the "application context." The two applications agree on the semantics of the data which is being transferred, but the representation of these data structures on the respective systems may be very different. As simple examples of such differences, one may use ASCII to represent characters and the other may use EBCDIC. One may use the "big endian" method of representing integers (most significant byte first), and the other may use "little endian." The applications are not concerned with these differences; they call on the Presentation Layer to take care of exchanging the data in such a way that the two systems can understand each other. The data structures are specified in a system independent language capable of representing the abstract syntax which defines the data structures exchanged by the communicating applications. The most commonly used abstract syntax language is called Abstract Syntax Notation One (ASN.1), but this is not the only possibility.

The Presentation Layer transforms the local data structures into a system independent syntax (the transfer syntax) which determines the format and ordering of the bits which are actually sent over the network to the other side. The receiving Presentation Layer decodes from the transfer syntax into its own local forms of data representation. The Presentation Layer on each side is able to do this because it has knowledge about which transfer syntax was used to encode the data, and which abstract syntax was used by the Application Layer. The pair of

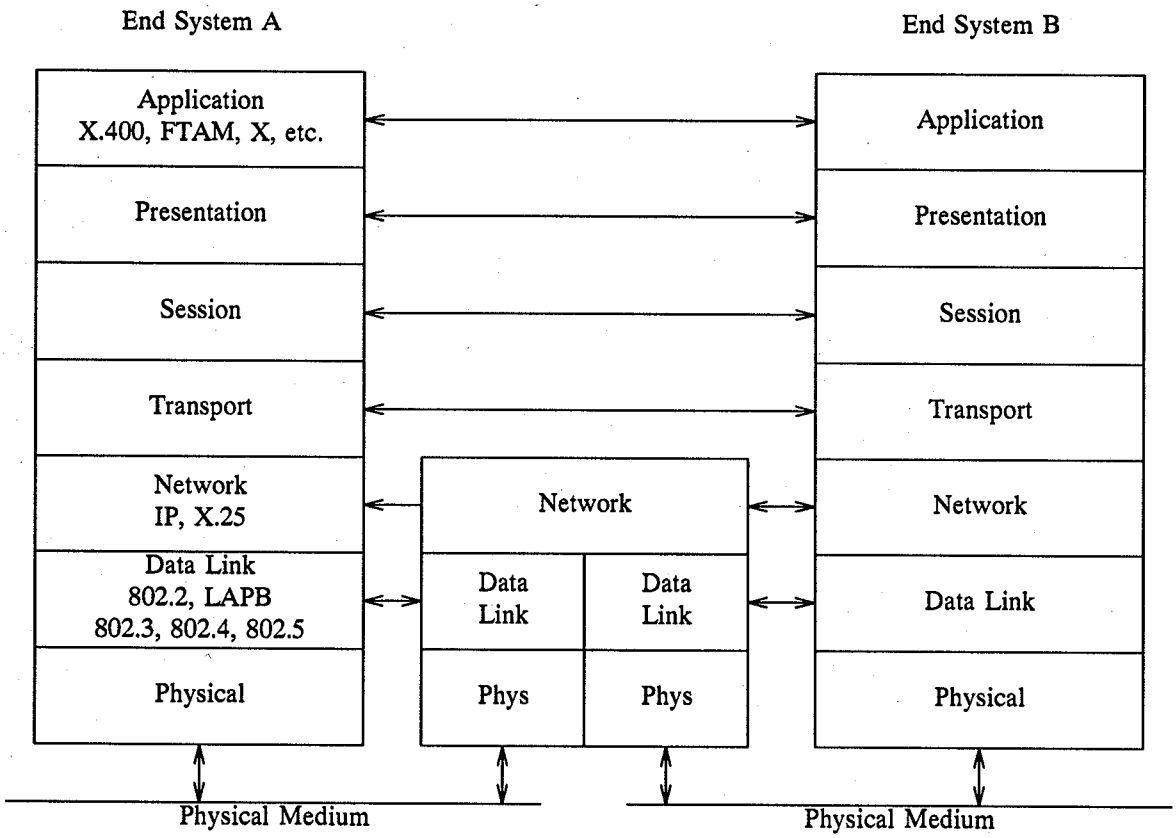

Figure 2: OSI Communication

syntaxes to be used in the communication, called the "presentation context", is negotiated and agreed upon by the two sides of the Presentation Layer. The most commonly used transfer syntax is the Basic Encoding Rules, BER, but it is not necessary to use BER in OSI.

The Presentation Layer calls on the Session Layer to conduct an orderly conversation with the other side, including graceful termination of the conversation and choosing of full or half-duplex communication. (Session performs many other functions, such as synchronization, activity management, and so forth, which are not detailed here.) Session conducts its conversation by means of a reliable end-to-end Transport connection, which in turn uses the Network Layer to route the data through the network. Details peculiar to the medium which is used (Token Ring, 802.3 Ethernet, etc.) are handled in the Data Link Layer, which in turn is dependent on the lowest Physical Layer for the actual electrical and physical connections between systems.

Transition from TCP/IP to OSI

Since TCP/IP is so successful, one may ask "Why bother?" when it comes to conversion to OSI. The first answer is the promise of applications which are much richer in function than those available with Internet protocols. Although only a limited number of these applications are available now, there will be many more in the future as more customers start to use OSI. The Message Handling System protocols (MHS) can send many more kinds of mail than the text-based Internet Simple Mail Transfer Protocol (SMTP) can. It defines a general-purpose third-party transfer facility, with special kinds of structured mail, such as messages with binary, voice or image parts, and Electronic Data Interchange (EDI) for exchange of financial information between enterprises. The OSI global directory service, using the X.500 standard, is already in widespread use. It enables the location of application programs all over the world on any subnet which can connect to a directory service. The new Transaction Processing (TP) standards will allow the customer's TP monitor, database, and application to be purchased from different vendors and still interoperate. The Commitment, Concurrency and Recovery (CCR) protocol provides the two-phase commit needed in TP and database manipulation. The OSI Virtual Terminal protocol potentially will allow a user to log in to any system in the world from any kind of workstation. The File Transfer, Access and Management (FTAM) services provide more than just the file transfer capability of the Internet File Transfer Protocol (FTP), such as remote database access, performing of actions on the remote files, access control, and handling of many different kinds of files.

The second reason for the broad potential of OSI is that it is composed of standards reached by formal, international agreements. They are thus politically neutral and less likely to change. OSI is in widespread use in Europe, and is often the network of choice for large, multi-vendor corporate networks.

The problem of making the transition from TCP/IP to OSI has been attacked in several different ways. See in particular Marshall Rose's book, *The Open Book* [ROSE90], which contains an entire section on *Transition to OSI*. But none of these documented approaches answer the problem of converting the X Window System to run over OSI.

One approach is the Application-Gateway. This is a program which converts analogous protocols from one suite to the other. For example, an FTAM-FTP Gateway can send files between OSI and TCP/IP systems, because it converts FTAM commands and responses into FTP commands and responses and vice-versa. The functions of the OSI FTAM must be truncated into the more limited functions of FTP. In the case of the X Window System problem at hand, the application running on both OSI and TCP/IP is **exactly the same application**. Thus no conversion of functionality is required. Although we did in fact write a Gateway, it does not need to understand the X requests that are made. It just passes the X protocol through to the other side, unexamined. A true Application-Gateway must be able to understand the protocol requests of each domain and translate them to an equivalent, or compromise, request in the other domain.

A second transition approach is to use a Transport-Gateway, Transport Service Bridge, or Network-Service Tunnel. These methods involve running the upper OSI layers on top of either TCP or IP. They are handy when you wish to experiment with OSI-based applications on systems which do not support the OSI lower layers. In our case, however, we had a TCP/IP-based application which needed conversion, and we had OSI products available on all of our systems.

A third approach, the Dual-Stack, is also used for unlike applications to talk to each other, and requires installation of both protocol suites. As with the Application-Gateway method, this method does not apply since we have one application, which we want to talk directly to the protocol stack on systems which have either OSI or TCP/IP installed.

Our approach is a fourth method. We claim that rewriting TCP/IP based applications to run over OSI can be easier than any of the approaches listed above, if a simple, high level interface to OSI is provided. The job is particularly straightforward if the high level interface is the socket interface, since many TCP/IP-based applications are written to this interface.

Standards

As shown above in the case of both X and OSI, in order for true communication to exist, standards are necessary, so independent implementations will interoperate. The X Window System is already a de facto industry standard. The definition of the protocol between X client and X server will soon be published as an American National Standard (ANS) [ANSI91], and will then become an international standard (IS). One of the authors contributed to Part IV of this draft standard, the *Mapping onto Open Systems Interconnection (OSI) Services*.

In this mapping, the X Window System applications, client and server, are found in the Application Layer of the OSI Reference Model. The Association Control Service Element (ACSE) manages the connection between client and server. The client sends the first X data, which is the Open Display request of the X protocol. This request, and all subsequent X requests, replies, events, and errors, are carried as User Data on Presentation Layer P-Data requests.

International consensus has been reached on X over OSI. The European Workshop for Open Systems (EWOS) has published the EWOS Technical Guide 013, *A Mapping of the X Window System over an OSI Stack*. This ETG specifies a minimal use of ACSE, Presentation and Session services, known colloquially as the "skinny stack," for use with X. In a *Guidance for Implementors* section, the ETG suggests that a Transport Layer API be used, and that fixed headers for the upper layers be prepended to the outgoing data and removed from incoming data. In work parallel to ours, the University of London Computer Centre, under the sponsorship of the UK's Joint Network Team, has implemented this "skinny stack" method on two systems [JNT].

While it may seem obvious that X, being an application, should operate in the Application Layer of the OSI Reference Model, this was not always the case. Initial work in mapping X to OSI placed it just above the Transport Layer, since it was felt that this layer was more closely analogous to TCP [BRENN91], [CROWC90]. This placement, however, is not a valid use of OSI. The OSI Transport Layer functions are not completely the same as TCP functions, and the OSI Reference Model assigns functions to the upper layers which are needed for valid communication. It was felt that a Transport Layer mapping would not be approved by ISO committees as an international standard since it violates the OSI Reference Model. In addition, Application Layer placement allows for future changes to the X protocol which can take advantage of the richness of OSI functionality, such as the ability to address multiple servers, OSI security aspects, and the ability to specify multiple transfer syntaxes including one for compressed data.

Implementation

In order to prove the feasibility of X over OSI, we implemented prototypes on AIX and on two other IBM operating systems - VM and OS/2 - which support the socket interface to TCP/IP by means of a user-space application library. IBM ships OSI products on all of these operating systems. On the RISC System/6000, the OSI product is called OSI Messaging and Filing, or OSIMF/6000 [OSIMF91]. For VM and OS/2, the OSI product is called OSI/Communications Subsystem, or OSI/CS [OSICS].

Because the Application Programming Interface to OSI is different in OSI/CS and OSIMF/6000, different approaches were used. This also provided the opportunity of comparing the methods for ease of implementation, minimization of changes to the X code, network management ability, and ease of conformance testing.

The goals of implementation of X over OSI across all three operating systems were the following:

- Enable X server and X client to support both OSI and TCP/IP protocol families at the same time. The reason for this goal is that many potential customers of X over OSI already have TCP/IP installed, and are adding OSI to their repertoire.
- Enable IBM X servers and X clients to support only one of the two protocol families, for systems on which only one is installed. This is done by means of conditional compilation and/or selective linking of the X code.
- Minimize changes to the X code. This was a goal even for the approach in which the changes were concentrated in the X code rather than in the socket layer.
- Maintain existing semantics of the X code. This goal means that, for example, we did not redesign the X client library to do synchronous (blocking) I/O, even though this would have made our job easier. The reason for this is that client applications are allowed to have connections to multiple servers at one time. Thus asynchronous I/O must be used.
- Make it as easy as possible for human users of X clients to specify the OSI addresses of X servers. OSI addresses are long and difficult to type. We wanted to be able to use nicknames, resolved to an OSI address by directory lookup.
- Require absolutely no changes to X clients themselves, except to specify that they must link with a particular Xlib. There is a large body of existing X clients. We did not want to add an invocation parameter to all of these.
- Make no changes to the invocation parameters for X servers, so that switching to the

enhanced X server is transparent to workstation users.

A primary goal of the design of OSI support was to allow the enhanced server and Xlib to support **both** OSI and TCP/IP at the same time. Thus, the server can accept connections from both TCP/IP-based clients and OSI-based clients, if it is running on a system on which both protocol families are installed. The client linking with the enhanced Xlib can connect to either an OSI-based server or a TCP/IP-based server if both are installed on the client host. This raised the problem of how to distinguish which protocol family is desired when the client is invoked. Currently, with TCP/IP, the server desired is specified as follows:

```
-display serveripname:x.y
```

where serveripname is the Internet host name of the system on which the server is running, x is the number of the display desired, and y is the number of the screen for that display. DECnet names are distinguished from Internet names by specifying a double colon, as follows:

```
-display serverdecnetname::x.y
```

The Xlib XOpenDisplay routine knows to attempt a DECnet connection if the server's name is specified with two colons following it.

We decided that a similar syntactical method of identifying the OSI protocol family would be beneficial. We selected the following method:

```
-display serverosiname:ix.y
```

where the *i* identifies the host name as being an OSI nickname used for OSI directory lookup. This method means that the user running the X client can select the protocol family desired, and there is no problem with duplication of Internet names with OSI nicknames. An OSI nickname is a short mnemonic name for a system or X server. This nickname is turned into a full OSI address (which may be quite lengthy and difficult to remember) by the software.

The X Window System code as distributed by the MIT X Consortium uses the BSD socket to access TCP/IP. The VM and OS/2 design consisted of modifying the existing application socket library (now shipped as part of the TCP/IP products) to invoke OSI at the ACSE/Presentation level, using the API provided by OSI/CS. Using this approach we were able to avoid changing the X code very much, and we could take advantage of the OSI upper layers in the product. Because of this approach's advantages, we also implemented an OSI socket library on AIX.

To result in an upper layer OSI socket, two sets of functional modifications were needed on the socket architecture. First, the socket architecture as implemented by IBM was redesigned to match the OSI socket support found in the 4.3 Reno release of Berkeley UNIX [BSD43]. A new family was defined to allow sockets to be declared as belonging to the OSI address family, and new address structures for this family were introduced.

Second, the 4.3 BSD socket design was extended to support OSI at the ACSE/Presentation Layer, so that it could be used by the X Window System. Three new protocol definitions were provided so that an application can specify that either Presentation layer, Session layer, or Transport layer protocols are to be used by the OSI socket being defined. New setsockopt and getsockopt services were added to specify user information, application context name, presentation context definition list, and maximum length of user information [CROWT91]. This new information is stored in the socket data structure.

Skinny Stack Implementation

On AIX, the first method tried was to modify the X server and X client library source code to support OSI by adding conditional compilation of two commonly used interfaces to the OSI Transport Layer. These two API's are the socket API, taken from the 4.3 Reno BSD OSI socket implementation [BSD43], and the X/Open Transport Interface(XTI) [XOPEN88], which is an important new standard for network programming and is the one used in IBM's OSIMF/6000 product.

The goal of this preliminary AIX work was to revise the Xlib and the sample X server code to support the XTI interface to either OSI or TCP/IP, and to support OSI with either the XTI or socket interface. Since X is written in a very clean, modular manner, the changes were confined to a very few existing X routines: four in Xlib and three in the X server, plus a new module for each of client and server. A new module for XTI support, and a module for OSI support, linked by both client and server, were added. A new module for OSI directory support, useable by any OSI-based application on AIX, was written. Since no OSI directory services are available with OSIMF, these routines needed to be written.

Because the server and client library must be able to support OSI, TCP/IP and UNIX-domain connections, a new data structure used by all connections had to be added. This data structure specifies the protocol family and endpoint type (socket or XTI). The *connection.c* module of the server contains routines for each protocol family supported which make the connection. A minor change had to be made to each of these to set up this new data structure. In both client and server, the data structure is tested on each read or write to determine which API to use (socket or XTI) and which domain to use (OSI, UNIX, or TCP/IP).

The OSI and XTI support was added with the following new or modified C language statements: Xlib, 250 statements; X server, 250 statements; new code used in both client and server, 1000 statements[1].

The work for the prototype was done on the brand-new Release 5 of the X11 source code. When an attempt was made to retrofit the X changes to the product release of AIXWindows, which at the time the work was done was Release 3 of X11, the overriding disadvantages of using this approach became clear. It was difficult to do the retrofit work for the server because key modules had been changed by MIT. The prospect of repeating this work for each future release of X was unpleasant. In addition, we had invested a lot of time in this work and the result was only that the X Window System now worked over OSI, but not any other application.

Because of these two problems, we decided to rewrite the code into a user-space socket emulation library. In AIX, sockets are of course part of the kernel. We wanted to get this implementation working quickly, without the complications of kernel modification. AIX does not currently support socket access to OSI at any layer. Our socket emulation library, thus, intercepts socket calls in user-space, determines whether this is a real socket or emulated socket, and makes the real socket call if requested. The emulated socket code issues the XTI call, and translates the return codes to socket return codes before returning to the caller. Much of the code from the embedded method was used intact in the socket library. The per-connection data structure was simply turned into the emulated socket data structure, set up on every call to the new Socket service (even for TCP/IP and UNIX domain sockets). The XTI access routines are now called by the socket emulation library rather than called by X routines.

Since the X code was already written to the socket interface, changes to use this new socket emulation library were minimal. Thus retrofitting these changes to future releases of X will be painless.

The number of lines of code for the changes done by this method was about the same as the previous method, but the number of lines modified in the actual X code was substantially less – under 100 for both client library and server. The bulk of the new code is in separate modules implementing the OSI socket library.

[1]All counts of statements in this paper were arrived at by counting lines of code ending in a semi-colon. This includes data declarations, and does not include lines consisting only of comments, curly brackets, preprocessor statements, or *if (expression)*.

In both implementation techniques, the upper OSI layers were needed, since in OSIMF/6000 the only OSI API is to the Transport Layer. This required implementation of specialized ACSE, Presentation, and Session Layers, thus lending itself to use of the "skinny stack" method envisioned in the EWOS Technical Guide. That is, the layers implement only the simple functions needed by the X Window System. On outgoing data, predetermined fixed headers for the upper layers are prepended to the X data itself, and passed out over the Transport layer. Incoming data is more complicated; it must be parsed to determine the header type and format of the data. The headers are stripped from the data, and only the X data itself is passed up to the caller.

The advantage to this approach is that the OSI Layer implementations were hand-coded for use by TCP/IP-based applications such as X only. Thus tests for the many unused functions of the Session Layer, for example, were eliminated. The Presentation Layer implementation does not need to handle encoding of general abstract syntaxes. In the socket approach, these upper OSI layers are embedded in the socket emulation library, below the socket API.

The initial implementation contains no network management in the upper layers, a disadvantage to this approach. If we had been able to use a complete, general-purpose OSI implementation, network management would have been included. Another disadvantage to this approach is that the specialized upper layer code must now be tested for conformance with standard OSI layers. While the EWOS Technical Guide restrictions permit simplified encoding of outgoing data, all possible forms of incoming data must be decoded, and these must all be tested. If we had been able to use a product level implementation of the OSI upper layers, this testing would have already been done.

A rule of thumb for performance of the X Window System is that the round-trip time for communication between client and server should be 50 milliseconds or less in order for human perception of response time to be acceptable. Performance of the OSI-based X server and X client library was measured by using *x11perf*, which is an X client used for measuring server performance. This client calculates and displays the round-trip time of communication between client and server before going on to execute detailed tests of the server graphics operations. This round trip time was found to be approximately 20 ms, with untuned code, which is an acceptable level. This number was dominated by the performance of the OSIMF lower layers. Running at Transport layer versus running at ACSE layer made very little difference in the round-trip time.

General Purpose OSI Product Implementation

For both VM and OS/2, IBM ships a socket library as part of their TCP/IP products. We obtained the source code for these libraries and modified it to support OSI as well as TCP/IP.

On VM, only X client applications are supported. The availability of X over OSI means that a VM customer running only the OSI communications protocol on their machine can execute X-based applications on their VM mainframe and take advantage of the display capabilities of a bitmap graphical terminal attached to an X server connected to an OSI network.

The goal of the design for VM was to provide the ability for any program using the socket interface for network communications to use TCP/IP, OSI and local sockets simultaneously. In the case of X, this permits a single client application to communicate with multiple remote X servers, some of which might be using TCP/IP transport protocol and others using OSI transport protocols. The socket application library was modified to support the OSI/CS ACSE/Presentation Layer API. This means that the socket library need not contain an implementation of the OSI upper layers, as was necessary in AIX.

The TCP/IP socket code was modified to test for the domain address family on each request. For most services, new routines were written for the OSI protocols and added to the socket library; in a few instances existing routines were simply modified to use the OSI/CS API to invoke the appropriate ACSE/Presentation Layer function. A major benefit of this approach is that it will allow any application (not just the X Window System!) which utilizes the socket for network communication to replace TCP/IP with OSI with only minor changes required to handle specification of the OSI protocol and address family. As a result, aside from the socket library changes, only minor changes to the X library were made to set up information required for OSI connections.

The OSI implementation used is an IBM product, and contains network management in all layers, as well as extensive directory services. It allows the clients and server to refer to each other very simply by nickname, so that there was no need to write any routines to perform directory lookup of OSI addresses. In addition, the OSI implementation is already conformance tested. Thus testing required for this work could be confined to the socket changes and X changes only.

The OSI support was provided by adding or modifying 100 C statements in Xlib, and 1600 C statements of socket library code to utilize the API provided by OSI/CS. Note that this is about the same effort as the AIX X code modifications, even though the AIX code is implementing the upper layers themselves, and the VM code is using an API to the upper layers. On first examination the number of lines of code that was added to the socket library to provide these services might seem surprising. However, an examination of the code reveals that there are three factors that account for this size. First, the socket library changes represent a general purpose implementation of services that are not optimized for usage by X. Modifications were made to provide consistency with the framework of the existing socket library structure. Two, because X code handles multiple connections at once, for this implementation most OSI/CS callable services are performed asynchronously and control is returned to the issuing socket routine as soon as the callable service is received by OSI/CS, but before the action is complete. Return indicates only that the parameters of the OSI service have undergone a preliminary stage of validation, and that the call has been accepted (or rejected) for further processing. This means that for each asynchronous OSI/CS service call that was used, an additional test had to be added to the socket code to determine the results of this preliminary validation. And finally, all return codes and error conditions that are received from OSI/CS must be converted to corresponding socket return codes and error conditions. Due to the large number of possible codes and conditions that can be returned from OSI/CS, a significant portion of the new code represents tests that are performed in the event that an error condition is received, and the subsequent conversion of these codes and conditions to socket codes and error numbers.

On OS/2, IBM currently ships an X server, called PMX, and no X client library. Availability of an OSI-based X server means that clients running on an OSI-based system can display on the PS/2 workstation. X server requests that are received from an X client program are interpreted and OS/2 Presentation Manager (PM) requests are used to control the workstation's bitmap display. Keyboard and mouse events, error notifications, and request replies from the server are packaged in X protocol packets and transported over the network to the client.

Since IBM provides OSI support for OS/2 with its OSI/CS product, we originally planned to implement the X/OSI support using an approach similar to that used in the VM implementation; we would modify the socket code allowing application programs to remain largely unchanged. TCP/IP socket source code for OS/2 V2 was obtained, and the implementation designed using this model. However, pending the availability of a OS/2 V2.0 Ethernet device driver for OSI/CS, it was necessary to change the design to use an approach similar to the first AIX implementation – modifications and additions were made to the PMX server. The OS/2 X server code code performs all tests for protocol family determination, and invokes the calls to the OSI/CS API as required.

As in the first AIX implementation, since the PMX server code has been modified this current OS/2 implementation will require that the changes be retrofitted whenever there are changes made to the PMX code on which it is based. Fortunately, modifications to existing server code was confined to two modules with 100 new C statements added. All additional changes were implemented in three new modules, representing an additional 1700 C statements. As in the case of VM, these new modules provide general purpose OSI functional equivalents of socket service requests, and are available to any application. However, since these changes were not made to the socket library (due to the device driver availability problem previously mentioned) applications must explicitly call for the OSI version of requests (e.g., osi_socket, osi_listen, osi_read, etc.).

Finally, as in the case of the VM implementation and in contrast to the AIX implementation, the OSI support is provided by the IBM OSI/CS product, and therefore a complete OSI implementation including network management, extensive directory services, as well as conformance testing are all provided by this product.

X TCP-OSI Gateway

A typical customer who is beginning to install OSI will have some systems running OSI, but some which are still running TCP/IP. The customer will want to use the X Window System to communicate among all of these systems. To satisfy this need we implemented the X TCP-OSI Gateway. This program reads data on one protocol family and writes it out on the other, thus connecting clients and servers running different protocol families. It runs on AIX and VM.

Figure 3 shows a diagram of a mixed network with some systems running TCP/IP and some running OSI. Through the intermediary of the X TCP-OSI Gateway, these systems can talk to each other.

The Gateway appears as an X server to normal X clients, and appears as an X client to normal X servers. It must be able to interpret addresses in both OSI and Internet address domains. It runs on a system which has both OSI and TCP/IP installed. The Gateway was written by adapting some of the network communication routines found in the server, and linking with Xlib to use the network communication routines found in the client. The Gateway does no interpretation of X protocol requests or replies. It simply passes them uninterpreted to the other side. Because of the specific data exchange made when an X client connects to an X server, this Gateway will work only for X. However similar work could be done for other distributed applications.

Future Work

Now that we have OSI socket libraries we plan to port other socket-based applications to OSI. Candidates for this work include TELNET, FTP, NFS, x3270. This will enable users to maintain their

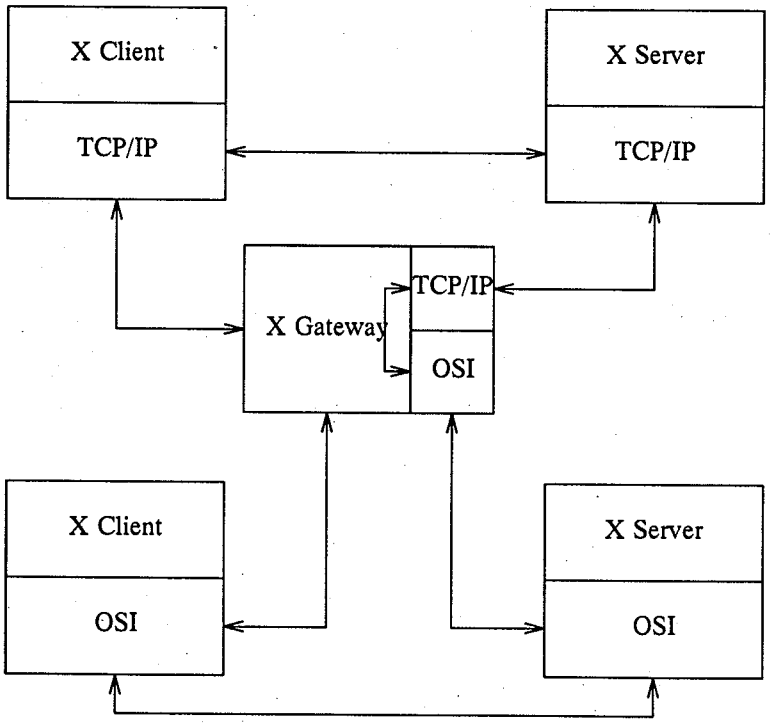

Figure 3: Use of X TCP-OSI Gateway

productivity with familiar network tools, while adding new OSI applications to their repertoire.

Conclusions

The problem of moving a TCP/IP-based application such as the X Window System to OSI was addressed. Different approaches were taken in different operating systems and these were compared. The "skinny stack" approach was used in AIX. Specialized, minimal function OSI layers were implemented by prepending the OSI layer headers to data sent using a Transport Layer API. This approach minimizes the path-length required for OSI, but means that we could not take advantage of upper layer network management or conformance testing.

When the "skinny stack" code was embedded in the X source code, retrofitting problems arose. On systems where the socket interface is found in a user-space application library, the socket was modified to access OSI at the Application Layer. Modification of an existing socket interface to support OSI means that the changes to existing X Window System product code can be confined to under 100 lines of code. This method proved to be so superior in terms of minimization of changes to the X source code and usefulness for other applications, that a user-space "socket" library was implemented on a UNIX system where sockets are found in the kernel. Proper implementation of an OSI upper layer socket clearly requires kernel changes. Once the socket modifications are made and tested, they can then be used by any application written to the socket interface. The socket implementation can use an upper layer access to OSI if this is available to kernel code (often it is not), or can implement the "skinny stack".

Although direct performance comparisons between the "skinny stack" method and general purpose OSI layer method could not be made since both methods could not be implemented on the same operating system, one would suspect that a "skinny stack" would exhibit higher performance. However, a general purpose stack can be implemented in such a way that a fast path is used for applications which do not use the complicated features.

For operating systems which are slow to implement OSI, a Gateway can be interposed between clients and servers in different protocol domains. This Gateway is particularly easy to write if a high-level interface such as the socket is available for both domains.

Based on our experience, we advocate that common network API's such as sockets, XTI, and TLI, should be modified to support OSI at the Application Layer as well as the Transport Layer. The OSI upper layers should contain network management and directory support, and should be conformance tested. The simplifications and fast-path approach highlighted by the "skinny stack" should be used as much as possible. When these high level interfaces are available in operating systems, existing TCP/IP-based application which are now in widespread use can be rewritten with only minor changes to run over OSI.

Although these experimental prototypes are not available to customers, they point the way toward solution of the problem of migrating all TCP/IP-based applications to OSI. An OSI upper layer socket, which implements the most basic, simplified functions of Session, Presentation, and ACSE, can be used by all socket-based TCP/IP applications. Since the more complicated functions of the OSI upper layers were not available to them, they either do not use them, or implement them in other ways. This capability is critical for users who want to change to OSI in order to comply to the Government OSI Profile, or who want to start using the rich functionality of OSI applications, or who want to communicate with European OSI-based networks, but do not want to give up using their TCP/IP applications.

Acknowledgements

We wish to thank our management at the IBM Cambridge Scientific Center, Bob Anderson and Dick MacKinnon, for funding and encouraging this work. Jerry Mouton of IBM Networking Systems had the original idea for the OSI upper layer socket, and both he and Keith Sklower of the University of California made contributions to the design of this socket. We would like to thank Jay Elinsky and Oleg Vishnepolsky of IBM Watson Research, and the PMX team at Cambridge – Allen Springer, Bill Barrett, and Rod Maxwell – for making their socket library source code and PMX source code available to us.

Bibliography

[ANSI91] X3.196-199x, X Window System Data Stream Definition. Part I, *Functional Specification*. Part II, *Data Stream Encoding*. Part III, *KEYSYM Encoding*. Part IV, *Mapping onto Open Systems Interconnection (OSI) Services*. Revised November 20, 1991.

[BRENN91] Brennan, Thompson, and Wilder, *Mapping the X Window onto Open Systems Interconnection Standards*, IEEE Network Magazine, May 1991.

[CROWT91] Crowther, *X on OSI*, Xhibition 91 Conference Proceedings, Integrated Computer Solutions, June 1991.

[BSD43] Manual pages for socket calls in 4.3 Reno release of Berkeley Software Distribution, Sections 2 and 4, Computer Systems Research Group, Computer Science Division, Univ. of California, Berkeley, Calif, May 30, 1990.

[CROWC90] Crowcroft, J., *Experience with mapping the X windows protocol onto ISO transport*

service,", Networks 90 - Network Management. Proceedings of the International Conference, Birmingham, UK June 1990

[DYER90] Dyer,"X Windows over OSI", Report of Joint Network Team, Rutherford Appleton Laboratory, United Kingdom, October 1990.

[EWOS91] EWOS Technical Guide 013, *A Mapping of the X Window System Over an OSI Stack*, European Workshop for Open Systems, 21 May 1991.

[GOSIP88] U.S. Government Open Systems Interconnection Profile (GOSIP), U.S. Federal Information Processing Standards Publication (FIPS) 146, Version 1, August 1988.

[IS7498] ISO 7498, Information Processing Systems – Open Systems Interconnection – Basic Reference Model, International Organization for Standardization, Geneva, Switzerland (1983).

[IS8823] ISO 8823, Information Processing Systems – Open Systems Interconnection – Connection oriented presentation protocol definition, Geneva, Switzerland (1988).

[JNT] Peter Furniss and Kevin Ashley, personal communications with the authors.

[OSICS] *OSI/Communications Subsystem General Information Manual*, GL23-0184, IBM, Palo Alto, California, March 1990.

[OSIMF91] AIX Version 3 for RISC System/6000 OSI Messaging and Filing/6000, User's and System Administrator's Guide, Second Edition, SC32-0012-01, IBM, Palo Alto, California, September 1991.

[ROSE90] Rose, *The Open Book*, Prentice Hall, 1990.

[SCHE90] Scheifler & Gettys, *X Window System*, Second Edition, Digital Press, 1990.

[XOPEN88] X/Open Portability Guide 3, Volume 7 Networking Services, X/Open Company, Ltd., Reading, Berkshire, United Kingdom, 1988.

Author Information

Nancy Crowther has been at IBM for 15 years. Most recently she has been a Staff Member at the IBM Cambridge Scientific Center, where she has worked in the area of networking software. She also worked for Informatics at NASA/Ames Research Center where she did scientific applications for 6 years. She has an M.S. in Applied Mathematics from the University of Santa Clara and an A.B. in Mathematics from Rutgers. Reach her via US Mail at IBM, 101 Main St., Cambridge, MA 02142, or electronically at crowther@cambridge.ibm.com.

Joyce Graham is a Staff Member at the IBM Cambridge Scientific Center. Prior to joining IBM, she worked at United Technologies Pratt & Whitney Division as a research engineer, and at the Massachusetts Institute of Technology as a scientific programmer, user consultant, and systems programmer. Since joining IBM in 1981, she has worked in the areas of operating systems, user interface, and network communications. Ms. Graham received a joint B.A./B.S. (Aeronautical Engineering) from New York University. She may be reached at joyce@cambridge.ibm.com or via US Mail at IBM, 101 Main St., Cambridge, MA 02142.