An Implementation of the OSI Upper Layers and Applications

1 Abstract

Above the transport layer, the open systems interconnection (OSI) basic reference model describes several application standards supported by a common upper layer protocol stack. Digital's high-performance implementation of the upper layers of the protocol stack concentrates on maximizing data throughput while minimizing connection establishment delay. An additional benefit derived from the implementation is that, for normal data exchanges, the delivery delay is also minimized. The implementation features of Digital's two OSI applications-file transfer, access, and management (FTAM) and virtual terminal (VT)-include the use of common code to facilitate portability and efficient buffer management to improve performance.

The open systems interconnection (OSI) basic reference model defined in the International Organization for Standardization standard ISO 7498-1 specifies a layered protocol model consisting of seven layers.[1] By convention, the first four layers-physical, data link, network, and transport-are referred to as the lower layers.[2] These layers provide a basic communication service by reliably transferring unstructured user data through one or more networks. The remaining layers-session, presentation, and application-build on the lower layers to provide services that structure data exchanges and maintain information in data exchanges to support distributed applications. These three layers are known collectively as the upper layers.

This paper first gives an overview of the OSI upper layers and of two application standards-file transfer, access, and management (FTAM) and virtual terminal (VT). The discussion that follows concentrates on the features of Digital's implementation of the upper layers and the two applications, with emphasis on novel implementation approaches.

2 Summary of OSI Upper Layer Standards

The application-independent parts of the OSI upper layers are defined in the following standards:

- ISO 8326 and ISO 8327-Session Connection Oriented Service and Protocol
- ISO 8822 and ISO 8823-Presentation Connection Oriented Service and Protocol
- ISO 8824-Abstract Syntax Notation One (ASN.1)

- ISO 8825-Basic Encoding Rules (BER)
- ISO 8649 and ISO 8650-Association Control Service Element (ACSE)

This section gives an overview of the services defined in these standards. The later sections File Transfer, Access, and Management Implementation and Virtual Terminal Implementation discuss two application-specific standards.

Session Layer

The transport layer service facilitates the exchange of unstructured bytes (i.e., octets) of data. However, exchanges between components of a distributed application are often structured. The function of the session layer is to standardize some of the common exchanges by supplying services that add structure to the transport layer exchanges.

The session-connection-oriented service has the three phases typical of all connection-oriented services: connection establishment, data transfer, and connection release. All structuring of the data exchanges occurs in the data transfer phase and is accomplished by using either tokens or synchronization. Hence, the connection establishment and release phases are not discussed further in this paper.

Tokens are used to control which peer session user of a session connection is permitted to invoke a particular service or group of services. The session layer also provides services to exchange tokens between peer session users. There are four types of tokens.

1. Data, for controlling half-duplex data exchanges

2. Release, for controlling which session user can initiate the release of a session connection

3. Synchronize-minor, for controlling the issuing of the minor synchronization service

4. Major/Activity, for controlling the issuing of major synchronization and activity services

For example, when the data token has been negotiated on a session connection, session data can be sent only by the end that currently has the token. Exchanging the data token between the session users provides a half-duplex data service.

The data transfer phase provides synchronization by allowing session users to insert major and minor synchronization points into the data being transmitted. Optionally, each direction of flow can have its own set of synchronization points.

Figure 1 illustrates a data exchange structured as a single dialog unit. A dialog unit begins at a major synchronization point and terminates

either at a new major synchronization point or by the release of the session connection. Further structure is possible within the dialog unit by inserting minor synchronization points.

The session synchronization services allow applications to insert synchronization points into their data exchanges. These points are application specific. The session service also provides a resynchronization service to allow a session user to request its peer to resynchronize to an earlier synchronization point, for example, to a previous point in a file transfer.

Activities provide an additional structuring service. An activity represents a logical piece of work. At any moment in time, there is at most one activity per session connection. However, several activities can exist during the lifetime of a session connection, and an activity can span session connections. The synchronization services can be used with activities services.

Presentation Layer

Different computer architectures and compilers use different internal representations (i.e., concrete syntax) for data values. Therefore, conversion between representations is necessary when communicating between dissimilar architectures. The intent of the presentation layer is to allow communicating peers to negotiate the data representation to be used on a presentation connection.

The presentation standards, ISO 8822 and ISO 8823, distinguish between abstract syntax and transfer syntax. Abstract syntax is the definition of a data type independent of its representation. Typically, data types are defined using the ASN.1 standard, ISO 8824, which was developed for this purpose. ASN.1 has a number of primitive data types, including INTEGER, REAL, and BOOLEAN, as well as a collection of constructed data types, including SET and SEQUENCE OF. These primitive and constructed data types can be used to define the abstract syntax of complex data types such as application protocol data units.

A transfer syntax is the external communication representation of an abstract syntax. Values from the abstract syntax are encoded according to the rules defined in the transfer syntax. A common way to define a transfer syntax is in terms of encoding rules. For example, these rules may indicate how an 1 INTEGER value is represented or how to encode a SEQUENCE OF data type. A widely used transfer syntax is the basic encoding rules specification, ISO 8825.

An abstract syntax can be encoded using different transfer syntaxes, of which there are many. The role of the presentation layer is to negotiate the set of abstract syntaxes to be used on a particular presentation connection and to select a compatible transfer syntax for each of these abstract syntaxes. This process ensures that both peers agree on the data representation to be used in data exchanges.

Application Layer

The application layer supports distributed interactive processing, that is, the communication aspects of distributed applications such as FTAM (defined by ISO 8571), directory service (defined by ISO 9594), and VT (defined by ISO 9040 and ISO 9041). Unlike for the session and presentation layers, numerous application layer protocols and services exist-at least as many as there are distributed applications.

The application layer structure specified in ISO 9545 defines a model for combining these protocols in the same system. The functions for a particular application are grouped together to form an application service element (ASE). FTAM, VT, and directory service are examples of ASEs and are the basic building blocks of the application layer. One or more ASEs are combined to form an application entity (AE). An AE represents a set of communication resources and can be thought of as a program on a disk. An invocation of an AE (i.e., execution of the program) can contain one or more instances of an ASE with one or more application associations, i.e., application layer connections. The AE specification also defines the rules for interaction between ASEs operating over the same association as well as interactions between associations.

An ASE required by all applications is called the association control service element (ACSE). The ACSE, defined by ISO 8649 and ISO 8650, is the service and protocol required to establish an application association. Therefore, an AE always contains at least the ACSE.

An application association is mapped onto a presentation connection; no other application association can share this presentation connection. In this way, applications gain access to the presentation and session data phase services.

3 New OSI Upper Layer Implementation

Digital's implementation of the OSI upper layers, namely OSAK, includes session, presentation, and ACSE services. Users of OSAK can thus establish application associations and use session and presentation services during the data transfer phase.

Aims

In 1988, when Digital decided to produce a new version of OSAK, three aims were considered paramount: high performance, maintainability, and portability.

Performance High performance of the OSI upper layers is essential to producing competitive OSI products. Because all OSI applications use

these upper layers, the performance of OSAK affects these applications. Therefore, OSAK aims to maximize data throughput and to minimize connection establishment delays. This improved performance is achieved by maximizing the use of the communication pipe and minimizing the local processing requirements. The process involves

1. Amalgamating upper layer state tables. The services provided by the presentation and session layers are similar. Also, connection establishment and release in the ACSE is basically the same as in the other two upper layers. Therefore, the three state tables can be combined into a single state table, thus improving performance by reducing the overhead. This amalgamation eliminates the need to manage links between state tables, requires all predicates to be tested in only one place, and generates only one state transition or action per inbound event.

2. Treating the presentation service P-DATA as a special case. The presentation service P-DATA is the most frequently used service, and hence, its performance has the greatest impact on data throughput. By fast-laning the processing of the P-DATA service, the normal overheads associated with the combined state table processing are avoided.

3. Good buffer management. The new application programming interface (API) to OSAK enables efficient use of buffers. We eliminated all copying of user data within OSAK by taking advantage of user buffers. On an outbound service, an OSAK user is requested to leave space at the start of the user data. If there is sufficient space, we add the OSI upper layer protocol control information (PCI) to the user buffer. This buffer is then sent to the transport provider. Otherwise, we allocate an OSAK-specific buffer using a user-supplied memory allocation routine.

   Before receiving an inbound service, the user must pass at least one user buffer to OSAK. This buffer is used to receive the inbound transport event (both user data and upper layer PCI). The upper layer PCI is decoded before the user buffers are returned. In addition to being extremely efficient, this approach has the advantage of allowing OSAK users to exert inbound flow control; if OSAK is not given any buffers, no transport events will be received. Also, this buffering scheme simplifies resource management in OSAK. As OSAK does not have any of its own resources, they all come from OSAK users. One OSAK user cannot interfere with the operation of another OSAK user by consuming all OSAK resources.

4. Parsing only the upper layer headers. The presentation layer standards model the mapping between concrete (internal) and transfer (external) representation of data values. In particular, the presentation state tables contain predicates to verify that all user data is from a current presentation context. Since the best place for encoding and decoding is in the application itself, OSAK does not implement these predicates. Rather, OSAK assumes that its users have correctly encoded their own protocol and will detect any problems when decoding.

5. Trading memory for performance. All encoding and decoding of upper layer

PCI is done with in-line code. More compact coding is possible using subroutines but at the cost of performance.

6. Minimizing parameter checking. Most parameters are pointers to user buffers. To check the validity of all pointers is time-consuming and, consequently, costly. Therefore, OSAK assumes that the pointers do indeed point to the user's memory.

Maintainability The code for the new version of OSAK is easier to maintain than the previous code. As stated earlier in this section, a major step in improving the maintainability was the use of amalgamated state tables. A single state table eliminates links between tables, reduces the amount of maintenance required, and thus simplifies the code. In addition, using a single table makes it easier to serialize events. With multiple state tables, an inbound transport event can trigger a conflicting state change in the session state table at the same time a user request is changing the presentation state table. Using a single state table for a particular connection ensures that only one event (i.e., either a user or a transport event) is active in the state table at any given time.

The state tables are written in M4 macroprocessor notation. Thus, the OSAK state table definition is similar to an OSI protocol specification; this improves readability. Macros are also used extensively to handle common buffer manipulation and the encode and decode functions. Although macros are preferred over subroutines to improve performance, macros can be converted, at the expense of slower performance, should a more compact version of OSAK be required.

Portability The new version of OSAK is designed to facilitate portability of applications using both the OSAK API and OSAK itself. The new OSAK API is designed to be common across all platforms and thus assists porting applications between platforms. The only major difference between the versions for the ULTRIX and the OpenVMS operating systems is the way events are signaled. The ULTRIX implementation supports both a polling model and an event-driven or blocking model. With the polling model, the OSAK user repeatedly calls OSAK routines to test for completion of an event; the routines used are osak_collect_pb() or osak_get_event(). In the blocking model, the OSAK user blocks awaiting the event, with the osak_select() routine.

These three routines are available to OpenVMS applications. In addition, the OpenVMS implementation supports event notification by asynchronous system traps (ASTs).

Also, the OSAK API is similar to XAP, the X/Open API to the OSI upper layers. To support OSAK on multiple platforms, as far as possible, OSAK code is common to all platforms. The main differences are the interface to the transport layer and the OpenVMS support for ASTs. Over 90 percent of the code is common to the ULTRIX and the OpenVMS versions.

Performance Measurements

Two performance metrics, throughput and connection establishment delay, were measured between two DECstation 3100 workstations connected by a lightly loaded Ethernet communications network. The DECstation machines were running ULTRIX V4.2 with DECnet-ULTRIX V5.1. OSAK accessed OSI transport through the X/Open transport interface (XTI) in nonblocking mode.

For throughput measurements, two programs were used: an initiator and a responder. The initiator

1. Establishes an association.

2. Reads the system time.

3. Transmits 2,000 buffers of data as quickly as possible. These user buffers contain sufficient space for the upper layer headers. When a send request fails due to flow control, the sender waits using the ULTRIX system call select(2) until the flow control is removed. The sender then collects the user buffers with the osak_collect_pb() routine before continuing with the send loop.

4. Reads the system time and calculates the time required to transmit the 2,000 buffers.

5. Releases the association.

The responder

1. Accepts an association request

2. Loops, waiting for a transport event using the ULTRIX system call select(2), and then collects the data using the osak_get_event() routine until all 2,000 buffers have been received

3. Responds to the request to release the association

Table 1 records the throughput measurements for various buffer sizes ranging from 10 to 16,000 (16K) octets per buffer.

Table 1: Throughput Measurements for Digital's OSI Upper Layer Implementation

Buffer Size (Octets)	Throughput (Kilooctets /s)	Number of Send Requests Flow Controlled
10	6.60	2
100	56.80	4
512	216.00	35
1,024	266.60	794
2,048	372.60	862
4,096	453.70	1,151
6,000	507.00	1,217
8,124	528.80	596
8,125	507.10	651
10,000	527.20	751
13,000	522.20	1,101
16,000	505.27	1,279

The data presented in Table 1 indicates that for small buffers, the throughput is poor. This low performance is due to the system associated with processing a send request, independent of the amount of data to be transmitted. However, the throughput rapidly improves until the buffer size reaches 4K octets. From this size on, the throughput measurement is almost flat at between 507K and 528K octets per second. The variation is due to fragmentation in the lower layers. The number of send requests flow controlled represents the number of times a send request was delayed because of flow control by the transport service in the course of transmitting the 2,000 buffers.

We profiled the initiator and the responder. For buffers ranging in size from 10 to 16K octets, the initiator spent more than 90 percent of the time in transport. For the responder, the percent of time spent in transport

varied between 60 percent for 10-octet buffers and 92 percent for 8K-octet buffers. The remaining time was spent primarily in select(2), waiting for and processing the next inbound event. Also, for the small buffers, a significant amount of time is consumed by initializing the user parameter block before returning it to the user.

We also used the throughput program to measure the connection establishment time. The program read the system time before and after the association establishment phase; the average connection establishment time was 0.08 seconds. In addition, tests on the new OpenVMS implementation indicate that throughput improved two to three fold as compared to the OSAK code in the previously existing OpenVMS implementations.

Both the throughput and profile data indicate that the transport performance dominates the performance of OSAK. Therefore, OSAK has met its design goal of reducing the overhead of the OSI upper layers to a very low level. Meeting this goal was necessary because poor OSAK performance would impact all OSI applications supported by OSAK. While further reductions in overhead are possible, such savings would be at the expense of OSI upper layer functionality.

4 File Transfer, Access, and Management Implementation

This section presents a summary of the ISO FTAM standard and details of Digital's implementation of this standard.

Summary of the ISO FTAM Standard

ISO 8571 File Transfer, Access, and Management (FTAM) is a five-part standard consisting of a general introduction, a definition of the virtual file store, the file service, the file protocol definitions, and the protocol implementation conformance statement proforma. The FTAM standard defines an ASE for transferring files and defines a framework for file access and file management.

Initiator and Responder FTAM service and protocol actions are based on a client-server model. In the FTAM standard, the client is referred to as the initiator, and the server is referred to as the responder.

The initiator is responsible for starting file service activity and controls the protocol actions that take place during the dialog (or FTAM association) between two FTAM applications. For example, the initiator has to request that an FTAM association be established, that a file be opened on a remote system, and that a file be read from a remote system.

The responder passively reacts to the requests of the peer initiator. The responder is responsible for managing the virtual file store and mapping any virtual file attributes into local file attributes.

Virtual File Store Many architectures and implementations of file systems exist, and storing and accessing data can differ from one system to another. Therefore, a mechanism is needed to describe files and their attributes independent of any particular architecture or implementation. The mechanism used in the FTAM is called the virtual file store. The FTAM virtual file store model consists of file attributes, activity attributes, file access structure, and document types.

File attributes describe the properties of the file, which include the size and the date of creation. FTAM file attributes also define the types of actions that can be performed on a file. Read access or create access are

examples of file actions.

Activity attributes are properties of the file, which are in effect for only the duration of the FTAM association. Examples of activity attributes are current access request, current initiator identity, and current concurrency control. Current access request conveys the access control applied to the file, e.g., read or write access. Current initiator identity conveys the name of the initiator accessing the virtual file store. Current concurrency control conveys the status of the locks applied by the initiator.

The FTAM file access structure is hierarchical and produces an ordered tree that consists of one or more nodes. This file access structure is defined in ASN.1 and can be used to convey the structure of a wide variety of files.

In the FTAM virtual file store model, document types specify the semantics of a file's contents. The FTAM standard defines four document types.

- FTAM-1, unstructured text files
- FTAM-2, sequential text files
- FTAM-3, unstructured binary files
- FTAM-4, sequential binary files

The virtual file store model provides a framework for defining many different file types, including those not supported by the standardized document types. The U.S. National Institute of Standards and Technologies (NIST) has used the virtual file store model to define document types to support various file types, such as indexed files.

FTAM File Service The FTAM file service is a functional base for remote file operations. Functionality defined by the FTAM file service is broken down into subsets of related services. The subsets of functionality are called functional units. Functional units are used by the FTAM protocol to convey a user's requirements. For example, the standard defines the read functional unit, which allows an implementation to read whole files, and the file access unit, which allows an implementation to access records in the file.

In addition, the FTAM standard defines the following classes of files service: transfer, management, transfer and management, access, and unconstrained. Each service class is composed of a set of functional units. For example, an FTAM implementation that supports the transfer service class will be able to either read or write files.

New FTAM Standard Work Modifications to the FTAM standard are in progress

in the ISO. The most important modification is the file store management addendum, which specifies how wild cards, file directories, and references (links) to files are to be handled in an OSI environment. The addendum also specifies how to manipulate groups of files. In the current version of the standard, only one file can be selected at a time.

Digital's FTAM Implementation

Digital's FTAM products, available for the OpenVMS and ULTRIX operating systems, support FTAM applications in both the role of initiator and the role of responder. The initiator applications allow users to copy, delete, rename, list, and append files. In the OpenVMS version, the initiator applications are integrated into the Digital Command Language (DCL) so that the user can continue to use the COPY, DELETE, DIRECTORY, and RENAME commands. Where the FTAM service and protocol is used to support these commands, the additional qualifier /APPLICATION=FTAM is required. In the ULTRIX version, the same functionality is provided using the set of commands ocp, orm, ols, ocat, and omv. These commands have the same semantics as the corresponding ULTRIX commands cp, rm, ls, cat, and mv, respectively, and are similar to the set of DECnet file transfer utilities of dcp, drm, dls, and dcat. (Note that the set does not include dmv.)

The responder applications allow users to create, read, write, delete, and rename files. File access, i.e., the location of specific records in a file, is also supported by the responder applications. The OpenVMS responder application supports file locking and recoverable file transfer.

Digital's initiator and responder applications support the following FTAM document types:

- FTAM-1
- FTAM-2
- FTAM-3
- NBS-9, FTAM file directory

Programmatic Interface The FTAM API is common across all platforms and shares a "look and feel" with the OSAK API. The FTAM API allows access to all FTAM services and parameters through the use of a single parameter block and five library calls.

- osif_assign_port()
- osif_deassign_port()
- osif_getevent()
- osif_send()
- osif_give_buffers()

The FTAM API can be used to create either initiator or responder applications.

Protocol Gateways Digital's FTAM products support two protocol gateways: an FTAM/file transfer protocol (FTAM/FTP) gateway is available on the ULTRIX version, and an FTAM/data access protocol (FTAM/DAP) gateway is available on the OpenVMS version. The FTAM/FTP gateway supports bidirectional protocol translation. Files on internet hosts can be accessed through the

gateway using FTAM; files on OSI hosts can be accessed through the gateway by using FTP.

Implementation Features Portability, maintainability, and performance were the major goals of the FTAM implementation. To achieve these goals we

1. Created a common code base. The code is implemented using the C programming language. The FTAM protocol machine and the initiator and responder application programs are implemented such that a large amount of the code can be used across multiple platforms. These modules are referred to as common code modules. Any system-specific code, which represents 90 percent of the code, is placed in system-specific modules. All other modules are common to both the ULTRIX and the OpenVMS versions.

2. Hid interface dependencies from FTAM. To aid in the porting of code to different platforms, the FTAM implementation makes no direct calls to system-specific interfaces.

3. Provided good buffer management. The FTAM implementation uses the same buffer management model as OSAK, described earlier in the section New OSI Upper Layer Implementation.

5 Virtual Terminal Implementation

Digital also implemented the OSI virtual terminal application standards. Details of the standards and features of the implementation follow.

Summary of the VT Standards

ISO 9040 and ISO 9041 are the two international standards that define the OSI virtual terminal. ISO 9040 is concerned primarily with specifying a model for a virtual terminal basic class service; ISO 9041 defines the protocol to be used.

OSI virtual terminals are divided into five classes, based on functionality.[3]

1. Basic-data consisting of rectangular arrays of characters

2. Forms-data consisting of characters arranged in fields of variable size and shape, with the manipulation of content controllable for each field

3. Text-data representing document structures as covered by the Office Document Architecture standards (ISO 8613 series)

4. Image-data representing images composed of arrays of dots, i.e., pixels

5. Graphics-data representing computer graphics elements, such as lines and circles

To date, most of the work within the ISO has concentrated on the basic terminal class, i.e., basic class virtual terminal (BCVT). An OSI virtual terminal implementation provides a mechanism that allows a user to interactively access another OSI system, when not directly connected to it. Since a variety of systems and terminals exist that are not necessarily compatible with each other, the ISO VT protocol provides a means by which dissimilar terminals and systems may interact.

An example of a dissimilar terminal and system interacting by means of a VT would be the action of deleting a typed character. Some systems expect the terminal user to enter the <delete> character as an indication of the intent to delete, whereas other systems may expect the user to enter a <backspace> character. VT resolves these differences by translating the local action into a virtual action. The action in our example becomes the virtual actions of decrementing the current cursor position and erasing the character at the current location. A cooperating implementation would then translate these virtual actions into an appropriate local action.

The VT protocol is very powerful in the respect that the protocol definition provides many options and features that allow the support of complex terminal models. During association establishment, cooperating implementations agree on the subset of the protocol and the terminal model to be used. The protocol subset and terminal model are referred to as the profile. In addition, VT provides two modes of operation: asynchronous (A-mode), which may be thought of as full-duplex operation, and synchronous (S-mode), which may be thought of as half-duplex operation.

The ISO base standards define two basic profiles, one for each mode. Additional profiles have also been defined (or are being prepared) by the regional OSI workshops. Currently, the OpenVMS and ULTRIX implementations of the VT protocol both support the following profiles:

1. TELNET-1988, which mimics the basic functionality found in the transmission control protocol/internet protocol teletype network (TCP/IP TELNET) environment

2. Transparent, which allows the sending and receiving of uninterpreted data

3. A-mode-default, which provides basic A-mode functionality

Digital's VT Implementation

Digital's VT implementation provides both initiator and responder capabilities. In addition to describing the features of the implementation, this section compares the VT protocol with other network terminal protocols.

Initiator and Responder The VT implementation for both the ULTRIX and the OpenVMS systems provides the capability to act as either an initiator (a terminal implementation) or a responder (a host implementation). The initiator is responsible for establishing an association with the responder based on information provided by the user, such as the desired profile. The

responder is responsible for accepting the peer association request and for creating an interactive context for the remote peer user.

On the OpenVMS system, the VT protocol initiator is invoked by the DCL command SET HOST/VTP; on the ULTRIX system, the VT protocol initiator is invoked using the ologin command.

Implementation Features The VT implementation uses the OSAK interface outlined earlier in the paper. The goals of the VT implementation were to provide a highly portable, very efficient, and easily extensible code.

To achieve the goal of portability, the implementation was divided into two major components: interface to the OSI environment and the non-OSI interfaces (e.g., to terminals). The OSI component is completely portable to multiple platforms. The non-OSI component is platform specific and must be rewritten for each unique platform. The interface between these components consists of six basic functions, which must be supported on all platforms.

- Attach/detach-to attach and detach the non-OSI environment
- Open/close-to open or close a specific connection into the non-OSI environment
- Read/write-to read or write data between the OSI and the non-OSI environments

Because each function is simple and clearly defined, the amount of platform-specific code required for implementation is minimal. For example, the read function on the ULTRIX implementation is only 10 lines of code. The implementation is therefore highly extensible to different platforms.

Performance of the VT protocol implementation is enhanced by using preallocated buffer pools. This approach to buffer management eliminates the overhead of dynamically allocating buffers.

Our VT protocol implementation not only implements the ISO VT protocol but also provides a gateway to and from other terminal protocol environments. We provide gateways to TELNET and to the Local Area Transport (LAT) on both the OpenVMS and the ULTRIX versions. In addition, we have a VT/command terminal (VT/CTERM) gateway on the ULTRIX version.

Comparison of the VT Protocol with Other Network Terminal Protocols Most comparisons with network terminal protocols deal with echo response time, that is, how long it takes for a character to echo to a display after being typed at the keyboard. VT, like TELNET and CTERM, can operate in two different echo modes: remote, where the echo is achieved by means of the

remote host; and local, where the echo is accomplished through the local host. A number of factors contribute to response time in a remote echo situation, including protocol overhead and line speed. TELNET has little protocol overhead; in fact, for most situations, transferring normal data requires no additional overhead. VT protocol overhead is approximately 30 to 1 for a typical A-mode profile, that is, 30 octets are required to carry

1 octet of user data. VT overhead may seem excessive when compared with TELNET. However, the VT protocol provides many additional capabilities that TELNET does not, such as the ability to accurately model different terminal environments. Additionally, the 30 octets of overhead does not increase significantly when larger amounts of user data are transferred.

The largest gains for the VT are in the area of S-mode profiles. S-mode profiles enable most character echoing to be done locally. By using an appropriate S-mode profile, the VT implementation can provide sophisticated local terminal operations. Thus, it is possible to edit an entire screen of text and then to transmit it all at once to the remote host. The ability to process large amounts of terminal input as batch jobs has many advantages, including reduced network bandwidth requirements, reduced CPU requirements of the remote host (since the remote host is no longer involved in character echo), and increased user satisfaction (since users experience no network delays for character echo).

6 Summary

Goals common to the OSAK, FTAM, and VT protocol projects included good performance and portability of implementation. Performance is especially important for OSAK, because it supports all other OSI applications. Maximizing the use of common code and reducing system dependencies in the three projects significantly reduced the engineering effort to port an implementation from one platform to another. This savings in human resources is necessary, given the growing set of hardware and operating platforms supported by Digital. Equally important is the integration of OSI applications with their non-OSI counterparts, for example, the ocp and ologin functions and the protocol gateways.

7 Acknowledgments

The authors would like to thank their colleagues for reviewing previous drafts of this paper. In particular, we would like to thank Chris Gunner and Nick Emery, who were instrumental in revising the OSAK API, and the OSAK team, who converted the advanced development code into the product.

8 References

1. J. Harper, "Overview of Digital's Open Networking," Digital Technical Journal, vol. 5, no. 1 (Winter 1993, this issue).

2. L. Yetto. et al., "The DECnet/OSI for OpenVMS Version 5.5 Implementation," Digital Technical Journal, vol. 5, no. 1 (Winter 1993, this issue).

3. P. Lawrence and C. Makemson, "Guide to ISO Virtual Terminal Standards," Information Technology Standards Unit (UK), Department of Trade and Industry (March 1988).

9 General References

Information Processing Systems, Open Systems Interconnection, Part 1: Basic Reference Model (International Organization for Standardization, reference no. ISO 7498-1, 1984).

Information Technology, Open Systems Interconnection: Connection Oriented Session Service Definition (International Organization for Standardization, reference no. ISO 8326, 1987).

Information Technology, Open Systems Interconnection: Connection Oriented Session Protocol Definition (International Organization for Standardization, reference no. ISO 8327, 1987).

Information Processing Systems, Open Systems Interconnection, File Transfer, Access, and Management: Part 1, General Introduction; Part 2, Virtual File Store; Part 3, File Service Definition; Part 4, File Protocol Specification; and Part 5, Protocol Implementation Conformance Statement Proforma (International Organization for Standardization, reference no. ISO 8571, 1988).

Information Processing Systems, Open Systems Interconnection: Service Definition for the Association Control Service Element (International Organization for Standardization, reference no. ISO 8649, 1988).

Information Processing Systems, Open Systems Interconnection: Protocol Specification for the Association Control Service Element (International Organization for Standardization, reference no. ISO 8650, 1988).

Information Processing Systems, Open Systems Interconnection: Connection Oriented Presentation Service Definition (International Organization for Standardization, reference no. ISO 8822, 1988).

Information Processing Systems, Open Systems Interconnection: Connection Oriented Presentation Protocol Specification (International Organization for Standardization, reference no. ISO 8823, 1988).

Information Processing Systems, Open Systems Interconnection: Specification of Abstract Syntax Notation One (ASN.1) (International Organization for Standardization, reference no. ISO 8824, 1987).

Information Processing Systems, Open Systems Interconnection: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (International Organization for Standardization, reference no. ISO 8825, 1987).

Information Technology, Open Systems Interconnection: Virtual Terminal

Basic Class Service (International Organization for Standardization, reference no. ISO 9040, 1990).

Information Technology, Open Systems Interconnection: Virtual Terminal Basic Class Protocol (International Organization for Standardization, reference no. ISO 9041, 1990).

Information Processing Systems, Open Systems Interconnection: Application Layer Structure (International Organization for Standardization, reference no. ISO 9545, 1989).

10 Biographies

David C. Robinson David Robinson is a principal software engineer in Network Engineering Europe. He was the architect for the OSI upper layers and designed and prototyped Digital's improved upper layer implementation. He came to Digital in 1988 from the General Electric Co. (GEC) in Chelmsford, Essex, U.K., where he developed a remote procedure call and a distributed computing environment. Dave holds a B.Sc. (Eng) in computing science (1982) and a Ph.D. in management of very large distributed computing systems (1988), both from the Imperial College in London.

Larwence N. Friedman Principal engineer
Lawrence Friedman is a technical leader in the OSI Applications Group. He joined Digital in 1989 and is the project leader for ULTRIX FTAM V1.0 and V1.1. In addition to his project responsibilities, Larry is Digital's representative to the National Institute of Standards and Technologies (NIST) FTAM SIG and editor of the NIST FTAM SIG Phase 2 and Phase 3 documents from 1990 to 1992. He is currently the editor for the FTAM File Store Management International Standard Profile. Larry holds a B.A. (1978) in music from Boston University.

Scott A. Wattum Senior software engineer Scott Wattum is a member of the OSI Applications Engineering Group. He is responsible for the design and development of OpenVMS Virtual Terminal V1.0 and is involved in the ULTRIX and OSF/1 porting efforts. Previously, Scott worked at the Colorado Springs Customer Support Center and provided network support, specializing in OSI protocols and applications. Prior to joining Digital in 1987, he was employed by the University of Alaska Computer Network in various software positions. He received a B.A. (1985) in theatre from the University of Alaska, Fairbanks.

11 Trademarks