# Digital Technical Journal

ALTAVISTA INTERNET SECURITY AND MAIL

ALPHA-BASED WORKSTATIONS FOR NT AND UNIX

digital

Volume 9 Number 2
1997

**Cover Design**
Tunneling and firewalls are two effective technologies for ensuring secure communications between the public Internet and private networks. Our cover depicts encapsulated and cryptographically secured data as "unreadable" numbers traveling in a protective tunnel until reaching the firewall. The firewall functions as a screen that permits only authorized data to pass into the private network where packets of data can be decrypted with a key that is shared between the sender and the receiver. DIGITAL implementations of a tunnel and a firewall are presented in this issue.

The cover design is by Lucinda O'Neill of the DIGITAL Industrial and Graphic Design Group.

The *Digital Technical Journal* is a refereed journal published quarterly by Digital Equipment Corporation, 50 Nagog Park, AKO2-3/B3, Acton, MA 01720-9843.

Hard-copy subscriptions can be ordered by sending a check in U.S. funds (made payable to Digital Equipment Corporation) to the published-by address. General subscription rates are $40.00 (non-U.S. $60) for four issues and $75.00 (non-U.S. $115) for eight issues. University and college professors and Ph.D. students in the electrical engineering and computer science fields receive complimentary subscriptions upon request. DIGITAL customers may qualify for gift subscriptions and are encouraged to contact their account representatives.

Electronic subscriptions are available at no charge by accessing URL **http://www.digital.com/info/subscription**. This service will send an electronic mail notification when a new issue is available on the Internet.

Single copies and back issues can be ordered by sending the requested issue's volume and number and a check for $16.00 (non-U.S. $18) each to the published-by address. Recent issues are also available on the Internet at **http://www.digital.com/info/dtj**.

DIGITAL employees may order subscriptions through Readers Choice at URL http://webrc.das.dec.com or by entering VTX PROFILE at the OpenVMS system prompt.

Inquiries, address changes, and complimentary subscription orders can be sent to the *Digital Technical Journal* at the published-by address or the electronic mail address, **dtj@digital.com**. Inquiries can also be made by calling the *Journal* office at 978-264-7549.

Comments on the content of any paper and requests to contact authors are welcomed and may be sent to the managing editor at the published-by or electronic mail address.

ISSN 0898-901X

Documentation Number EC-P8429-18

Book production was done by Quantic Communications, Inc.

# Contents

# Editor's Introduction

DIGITAL has pioneered many networking developments in its 40-year history. A recent development, AltaVista, has captured the popular imagination, as evidenced by worldwide accesses, averaging 18 million per day, to this Internet search engine. Introduced in 1995, AltaVista indexing of the entire Internet was made possible by 64-bit VLM Alpha technology. The index proceeds today at a pace of more than 6 million pages per day. DIGITAL's Internet developments, however, go well beyond search functions. Business users need greatly improved security and protection to integrate the power of Internet connectivity into their businesses. It is this need that is addressed in the papers on tunnels, firewalls, and electronic mail. Additional papers in the issue feature high-performance, low-cost Alpha microprocessor-based workstations with unique design features, such as a single-chip core logic ASIC.

"Tunnel" and "firewall" are strong metaphors that developers use to connote the kind of security software necessary to protect business communications transmitted over the Internet. Tunneling protects data as it travels in the public Internet by providing secure encapsulation within the standard TCP/IP protocol. However, as Ken Alden and Ted Wobber explain, additional security measures are necessary, specifically, cryptographically secure encapsulated packets. The authors describe how secure network-level routing can be achieved by combining the well-known technologies of tunneling and secure channels. The paper includes their experiences in deploying the AltaVista Tunnel within DIGITAL.

Once data arrives—almost—at its destination, the firewall is a filtering router that determines which data packets will be allowed to pass from the public to the private network. Mark Smith, Sean Doherty, Ollie Leahy, and Der Tynan compare types of firewalls; describe firewall functions such as alarm systems, authentication, and reporting; and present the design of the AltaVista Firewall for DIGITAL UNIX. The AltaVista Firewall comprises both application-level and packet-filtering functionality and implements the principle "that which is not expressly permitted is denied."

The development of the AltaVista Mail product is presented by Nick Shipman as a case study in the issues facing engineers who design products for business users of the Internet. He relates several of the fundamental assumptions about engineering projects that were overturned by the engineering team; for example, product definition had conventionally started with the technical issues to be addressed and now started instead with a product purchase price. Further, in an effort to ensure product simplicity for the target customer, they imposed the principle of simplicity throughout the project—simplicity in presentation, in design, in methods, and in implementation.

A low-cost, high-performance workstation has been designed by DIGITAL's workstation engineering group. In the first of two papers about the DIGITAL Personal Workstations, Ken Weiss and Kenny House discuss the primary reasons for initiating a wholly new design: simultaneously to take advantage of new, high-performance memory technologies and to implement at a low cost. A new, low-cost core logic design was needed to function as the CPU-to-memory interface. The result, described by Reinhard Schumann, was the 21174 single-chip core logic ASIC for the Alpha microprocessor. Designers were able to meet their own aggressive performance goals by focusing on reductions in the main memory latency that was attributable to the memory controller subsystem and by using as much of the raw bandwidth of the Alpha 21164 CPU's data bus as possible.

Subjects for papers in the next issue of the *Journal* include the parallel SCSI technology, shared desktop software, and a high-performance debugger.

Jane Blake

Jane C. Blake
*Managing Editor*

# Foreword

Paul J. Cormier
*Director of Engineering, AltaVista*

In this issue, we focus on Internet software products that are part of the AltaVista portfolio. These particular products are notable because of the fashion in which they have been developed and brought to market.

With the commercial Internet software industry moving quickly from nonexistence to the most competitive and fast-moving industry in existence, engineers have recognized the need for innovation at all stages of product life. To succeed in Internet software, engineers must be innovators both in technology and in product development. Innovation begins with the concept and continues through product development and on to delivery to the end user; and the cycle continues.

From the beginning of the Internet revolution, DIGITAL has been a significant player in Internet software and solutions development. The company's success is attributable to the many diverse and technically talented groups that are focusing their resources on developing software and solutions for Internet users.

The products presented in this issue are good examples of the results that can be achieved in extremely short periods of time—six to nine months—when research, product development, and services groups work together to bring world-class products to market.

In the case of the Firewall product, research took the lead early, while the Internet was still used almost exclusively by the scientific and technical community. As one of the first Fortune 500 companies to connect to the Internet, DIGITAL quickly saw the threat to the security of its network and, in response, members of its research group developed the initial firewall technology. As with many innovations, this technology was recognized by DIGITAL's customers as state-of-the-art and was in turn demanded by them for their own uses. While this complex technology was still in its infancy, DIGITAL Services was able to deliver high-end security solutions to companies that desired to connect to the Internet.

Not surprisingly, these companies also needed to address the same network security issues that DIGITAL faced as a consequence of connecting to the public infrastructure. Starting with the firewall technology, the SEAL Firewall Service was born, and DIGITAL became one of the very first Internet security software providers.

As more and more enterprises connected to the Internet and experienced the same security issues DIGITAL had been facing, it became evident that both firewall technology and the market were beginning to mature. These factors quickly led to the DIGITAL AltaVista Firewall product.

DIGITAL responded to this market demand quickly, initially moving the SEAL technology to a standalone product on DIGITAL UNIX platforms. The same engineering group that developed the SEAL technology for the Services group seamlessly moved to product engineering, first in the Internet Business Group and later to the AltaVista Group. This smooth transfer of experience allowed DIGITAL to go to market after a short, six-month development cycle and to be one of the first vendors to offer a standalone, commercial firewall product.

Engineering has learned from research and carried that knowledge and experience through services and directly to the product engineering community. Moreover, engineering has adapted its process to stay competitive within the Internet market, enabling DIGITAL to be a technology leader with the AltaVista Firewall product on DIGITAL UNIX and, more recently, on the Windows NT platform.

The AltaVista tunnel, or secure virtual private network product (VPN), has similar roots to those of the firewall technology. Tunneling was born in response to a need for visitors at DIGITAL facilities to securely traverse the trusted internal network with *untrusted* packets. Again, the research community took the lead.

With the tunneling concept reversed, that is, access allowed from the untrusted external network (the

Internet) to the trusted corporate network and with encryption added, the rudimentary basis for today's product was put forward.

The newly formed Internet engineering group was ready to take the technology and prototype forward, putting into action a new instance of the research–engineering partnership. As was the case with the firewall, a talented engineering group moved the initial product to market within six months. DIGITAL was once again able to lead in the Internet space and claim the first VPN product to surface in the market, one that currently has many competitors.

As was also the case with the firewall, DIGITAL recognized a good use of this technology to solve one of its own problems. The telecommunications costs of moving the U.S.-based sales force to home offices and connecting it back into the corporate network were becoming excessive. The information services organization ran a pilot with 2,000 sales people, using local Internet connections and the Internet tunnel to authenticate users to the DIGITAL corporate network. The solution was perfect because the tunnel supplies the encryption capability that ensures the privacy of confidential business data as it traverses the public network infrastructure.

The results of the pilot were staggering in terms of the savings in telecommunications costs and keeping our internal network secure. With this pilot in hand, information services moved to offer the tunneling service to other internal groups as a way to solve DIGITAL's mobile-worker problem.

DIGITAL Services has also begun to offer the tunnel product, coupled with information services' pilot experiences, as a solution to its customers—the same model used with the initial firewall technology.

As DIGITAL and the industry move forward in using the Internet as an effective business tool, standards are emerging that DIGITAL is helping to define. Future products are being developed based on the standards and include features that allow other companies, who may have very different security strategies and policies, to take advantage of the Internet in a secure and productive manner.

The model of research, product development, and services working together to deliver innovative, cutting-edge products and solutions that use the ubiquity of the Internet to solve real-world customer problems will continue to expand DIGITAL's Internet capabilities and offerings.

A cornerstone of the research–product-development–services model is the talent and mind-set of the product engineering group. The advantage of keeping intact the core of the Internet and AltaVista engineering groups through the entire technological cycle that I present here has enabled the engineers to react quickly to changing requirements and market conditions. The group has consistently responded with two major product releases per year and some minor releases needed to satisfy a particular, significant demand.

As has been proven with these products, the model that is good for the company and for the customer is one that includes

- Researchers incubating and piloting the technology in the labs
- Engineering groups rapidly prototyping and developing product
- Services groups developing a repeatable solution for customers

DIGITAL will continue to move technologies rapidly from research through products and on to solutions, thus accelerating the use of the Internet as a mainstream business tool and helping businesses take advantage of the Internet and be competitive in their own markets.

[illegible]

Kenneth F. Alden
Edward P. Wobber

# The AltaVista Tunnel: Using the Internet to Extend Corporate Networks

**The public Internet has become a low-cost connection medium for joining remote employees or offices to a private intranet and for permitting impromptu high-speed connections between business partners. This connectivity is offset by a significant loss in security. The AltaVista Tunnel, a DIGITAL product, offers secure network-level routing over Internet connections by combining two well-known networking technologies: tunneling and secure channels. This paper discusses the design and implementation of the AltaVista Tunnel and describes our experience in deploying the product within DIGITAL.**

The public Internet is fast becoming a ubiquitous and inexpensive medium for connecting remote employees or offices to a private intranet or for permitting impromptu high-speed connections between business partners. This gain in connectivity is offset by a significant loss in security, however. The Internet is notorious for electronic break-ins and eavesdropping.

The AltaVista Tunnel, a DIGITAL product, offers network-layer routing over secure Internet connections. This allows, for example, a mobile user to connect securely to his or her corporate network using the Internet. Similarly, a corporate network can employ the AltaVista Tunnel to securely link remote offices with Internet connections. Although our product uses the Internet for packet transport, all traffic is encapsulated within cryptographically secured connections. Because the AltaVista Tunnel is a network-layer router, client applications can run without modification. Moreover, our product is firewall independent and therefore can be used in concert with most common firewalls. The AltaVista Tunnel supports both static connections to remote offices and intermittent connections to single-user machines. Currently, implementations exist for the UNIX, Windows 95, and Windows NT platforms.

In this paper, we begin with an overview of the benefits and pitfalls presented by using the Internet for private network connectivity. Next, we describe the design of the network protocol used by the AltaVista Tunnel, with a particular focus on the security concerns that led to this design. We then discuss how we implemented our design. Finally, we briefly describe our experience deploying the tunnel product in a large corporate network, provide performance data, and discuss some of the security risks this technology entails.

## Overview

Before the Internet became pervasive, corporate networks were built from leased and dial-in telephone lines. Such networks carried substantial costs for both communications equipment and telephone service. Usually, security relied on the inaccessibility of the physical medium, and over the years, the risk of wiretap has proved to be slight when compared to password cracking or other higher-level attack. The reason for this is that most telephone systems are both proprietary and centrally managed, and they are therefore not easy to subvert in the large without a substantial budget.

The Internet brings opportunity and challenge to the modern corporate network designer. Global connectivity makes it possible to replace expensive leased lines and communications equipment with Internet connections. However, such connections lack the physical security of telephone lines. Furthermore, direct connection to the Internet poses numerous, well-documented security problems. Consequently, many organizations find it necessary to isolate their private networks behind firewalls—filtering routers that place constraints on packets allowed to pass between protected and public networks. The policy decisions made in configuring firewalls always involve a difficult trade-off between security and functionality.

Cryptography makes it possible to emulate most of the properties of physically secure wire using Internet connections. When encapsulated at a suitable protocol level, cryptographically secured data can be allowed to traverse firewalls without substantially weakening security policy. However, the encapsulation protocol must require no implicit trust in the router nodes and links that make up the fabric of the insecure network. To solve this problem, the protocol employed by the AltaVista Tunnel uses a synthesis of two well-understood networking constructs: tunneling protocols[1] and secure channels.[2]

## Protocol Design

In computer networks, tunneling is the act of encapsulating one communications protocol within another. For example, a DECnet-in-IP tunnel might transport DECnet datagrams over an Internet Protocol (IP) network using IP datagrams. In this arrangement, IP datagrams act only as a transport mechanism—there is no need for the active nodes in the IP network to interpret or to manipulate the encapsulated DECnet packets. A tunnel alone, however, cannot guarantee that an intermediate node ("man-in-the-middle") will not intentionally read or modify the data portions of tunneled packets. To prevent such unwanted tampering, we cryptographically secure encapsulated packets for passage over the public network. Abstractly, data passed over this secure channel appears once and only once at the receiver as sent by the sender. Furthermore, an attacker observing the public network cannot read this data. Thus, tunnel encapsulation ensures that private-network datagrams cannot interact with the routing algorithms of the public network, whereas secure channels guarantee that the tunneled data arrive intact from an authenticated source and that privacy is maintained.

Figure 1 depicts a secure tunnel in operation. Nodes A and B are tunnel endpoints, that is, packet routers that forward to and from tunneled routes. Node A processes datagrams in private network X and determines which, if any, should be routed to private network Y. Node A then encapsulates all such datagrams and sends them securely across its tunnel connection to node B. Node B checks the integrity of each transmission and then decapsulates and forwards the datagrams to network Y. The process is symmetric, although this is not pictured.

These methods can be used to connect any sort of private network; however, our product is specifically designed to connect IP networks by tunneling IP datagrams. Given the dominance of IP in the network marketplace, the choice of network type is easy. The choice of protocol from which to construct tunnel connections is more difficult. There are three obvious candidates: IP, User Datagram Protocol (UDP), and Transmission Control Protocol (TCP).

Since IP is a network protocol, there is no notion of port-level addressing. This implies that IP-in-IP tunnels must be implemented very close to the operating system, and any multiplexing of tunnel connections must be explicitly added. Since our goal was for our tunneling product to be firewall and operating system independent, we rejected IP in favor of a higher-level protocol.

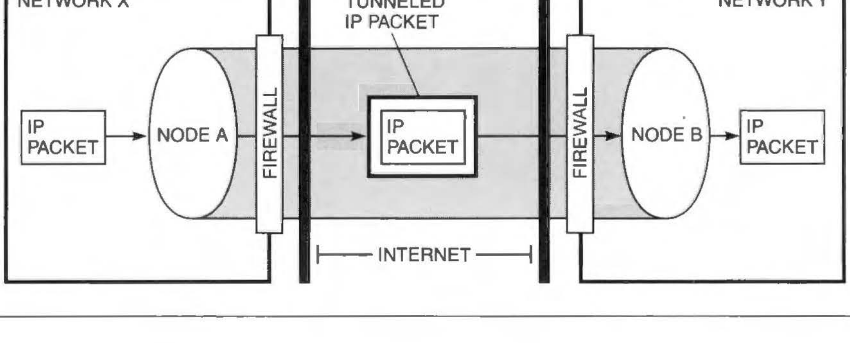


**Figure 1**
A Secure Tunnel in Operation

The choice between UDP and TCP depends on whether datagrams or byte streams best apply to tunneling. Since our application is inherently connection oriented, TCP offers a natural fit, while any UDP design must include an explicit means for reliable connection maintenance. In addition, byte streams eliminate constraints caused by packet boundaries, so fragmentation and maximum size determination pose few difficulties. Furthermore, byte streams enable forms of cryptography and data compression that would be awkward to implement using datagrams. Of course this flexibility does not come without cost. TCP adds an extra layer of reliable transmission, and per-packet headers are large.

The previous discussion lends no clear advantage to either protocol option. We chose to implement the AltaVista Tunnel using IP-in-TCP in order to simplify firewall security policy. As shown in Figure 2, a tunnel connection usually traverses at least one firewall. In practice, a tunnel virtual connection is composed of several distinct TCP connections laid end-to-end. Where TCP connections meet, there is a bidirectional relay process that shuffles packets in either direction. Such a relay service is included with most firewalls.[3] We also offer an intelligent relay that participates in the tunnel connection protocol and therefore allows more flexibility in choosing destination endpoints.

By using TCP connections and relays, we minimize the policy changes required to permit tunnel traversal. All that is necessary is to enable TCP connections between the tunnel endpoint, which is on the private network, and the relay, which is just outside the firewall. (Note that relays are logically outside the firewall, although they might be implemented on the firewall machine.) Whether a generic or an intelligent relay is used, firewall-traversal connections always originate on a locally controlled network. Furthermore, TCP connection requests are infrequent, and therefore TCP traversals are more tractable to log at the firewall than are datagrams. Although the firewall industry has begun to develop standards for IP-in-IP tunnels,[4–6] our choice of IP-in-TCP gives us the clear advantage that tunnel endpoints need not be packaged with or dependent on a specific firewall implementation. Eventually, the emerging standards will probably prevail for static tunnels; however, no standards exist for transient (mobile) users and our solution remains quite viable.

## Implementation

As with many tunnel implementations,[1] we provide tunneling by tricking the operating system's routing layer into forwarding packets to an emulated network device. This device does not transmit packets directly, but rather it encapsulates them as data within a higher-level protocol. The AltaVista Tunnel implementation contains three major components: the tunnel application, the protocol handler, and the pseudo-device driver. The main function of the tunnel application is to interact with the user or system administrator and to modify the system routing tables to make tunneled routes available. This code also maintains a database of acceptable partner endpoints and matching cryptographic keys. The protocol handler implements the tunnel encapsulation protocol and all associated cryptography. The pseudo-device driver is responsible for redirecting packets from the local IP stack to the encapsulation protocol handler and vice versa.

Figure 3 shows how the components of the AltaVista Tunnel cooperate to process tunneled IP packets. The diagram depicts a single-user client and a tunnel server. Although the same basic structure applies to all tunnel endpoint software, there are substantial differences between single-user and server configurations, and between the UNIX and Windows implementations. For example, the single-user version usually runs only while the user is actively connected. On the server side, the tunnel application is a daemon process that continuously waits for connection requests and services existing connections. The following three sections discuss the individual system components in detail and, where appropriate, point out the differences between the various software configurations.

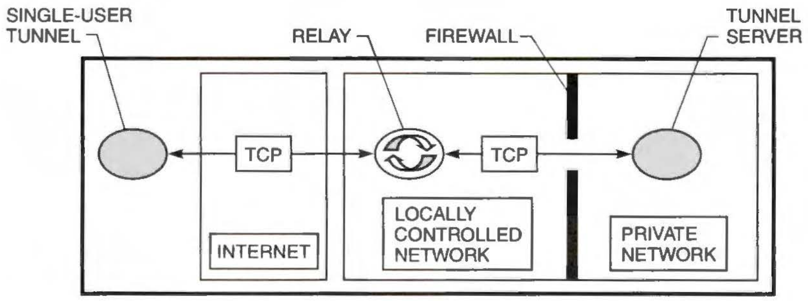


**Figure 2**
Tunnel with Intelligent Relay

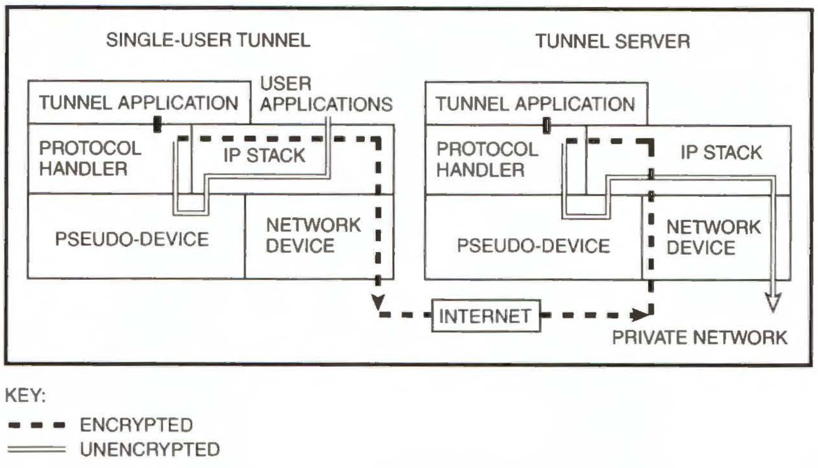


**Figure 3**
System Components and Data Flow

## The Tunnel Application

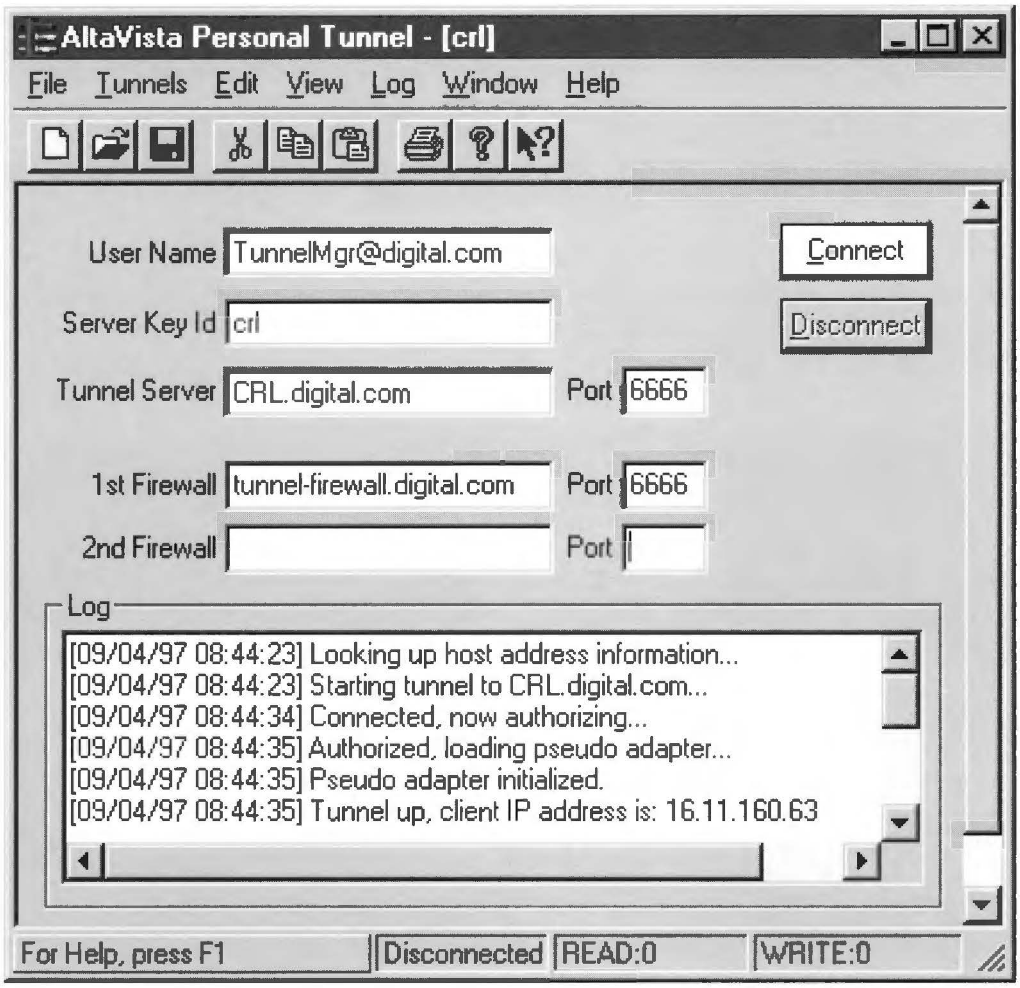
The primary function of the tunnel application is to present a user interface (UI). Although each instantiation of the user interface is slightly different, the function of the application remains the same. The AltaVista Personal Tunnel '97, a single-user configuration, offers a straightforward graphical user interface (GUI) (see Figure 4) that allows the user to register a set of target tunnel servers, select from this set, and then establish and tear down connections. The emphasis is on simplicity. A tunnel connection may be started from either a command line interface or the GUI. If the GUI is used to start a tunnel, the GUI window can be minimized and ignored until the end of the tunnel session. The application logs all interesting events, reflects current state through the user interface, and notifies the user of exceptional events. In this configuration, only traffic from local applications is directed over the tunnel, and no inbound tunnel connection requests are accepted.

In the server configuration, the tunnel application is significantly more complicated. The primary function of the server code is to restrict tunnel access to authorized clients. To achieve this, the server application is also responsible for issuing cryptographic credentials and maintaining an authorization database. In addition to accepting connections, a tunnel server is capable of initiating them. In the "workgroup" tunnel configuration, two servers cooperate to maintain a permanent connection, for example between a corporate network and a remote office local area network (LAN). A tunnel server is a full-fledged router—its job is to forward packets from the protected network into the tunnel and vice versa. We offer servers for both the UNIX and the Windows NT environments.

### *Routing*

As mentioned in the Implementation section, our tunnel works by manipulation of the system routing table. In some environments, such as Windows 95, there is no fully integrated notion of packet routing (sometimes called IP forwarding). However, there is support for multiple network devices. Each network device has a uniquely assigned IP address so that the IP stack can determine which device to use when transmitting packets. The AltaVista Tunnel pseudo-device appears to the operating system as just another network device. There is a one-to-one relationship between tunnel connections and pseudo-devices. During connection establishment, the tunnel application activates a pseudo-device and modifies the routing table to include any newly reachable private network or networks. The application then restores the original state upon termination of the connection.

The tunnel server is implemented in a richer routing environment. Each server typically routes an entire IP class-C subnetwork (254 addresses) but may support partial subnetworks or multiple networks as well. A tunnel server can maintain multiple connections, and this is accomplished by assigning a different IP address to each pseudo-device/tunnel connection. IP pseudo-device addresses at both ends of the tunnel are assigned dynamically or statically from a pool of IP addresses controlled by the server. The operating system, combined with a routing management program such as gated,[7] performs all necessary route propagation. As discussed in the next section, each tunnel user can be restricted to a specific set of IP addresses. This approach allows network managers to establish routing policy based on user class. To obtain fine-grain control over a given tunnel connection, the server can also run a packet-filtering program such as screend[8] to restrict the IP protocols entering and exiting that tunnel.


AltaVista Personal Tunnel - [crl]
File Tunnels Edit View Log Window Help
Connect
Disconnect
For Help, press F1

User Name TunnelMgr@digital.com
Server Key Id crl
Tunnel Server CRL.digital.com Port 6666
1st Firewall tunnel-firewall.digital.com Port 6666
2nd Firewall Port

Log
```
[09/04/97 08:44:23] Looking up host address information...
[09/04/97 08:44:23] Starting tunnel to CRL.digital.com...
[09/04/97 08:44:34] Connected, now authorizing...
[09/04/97 08:44:35] Authorized, loading pseudo adapter...
[09/04/97 08:44:35] Pseudo adapter initialized.
[09/04/97 08:44:35] Tunnel up, client IP address is: 16.11.160.63
```

Disconnected READ:0 WRITE:0

**Figure 4**
The AltaVista Personal Tunnel '97 User Interface

***Key Management and Access Control***

In practice, a secure channel protocol is only as strong as the techniques it employs for naming and key distribution. In the AltaVista Tunnel system, we must name both tunnel servers and human users. (Tunnel users must be authenticated by name, not by IP address, since many users acquire IP addresses dynamically from their Internet service provider.) Because no ubiquitous infrastructure exists to support such a namespace, our software currently assumes a flat, server-specific naming structure, much in the style of PGP.[9] We use RSA public-key cryptography[10] to establish secure connections. Each tunnel endpoint maintains a key file that contains a sequence of names and matching public keys—one (name, key) pair per potential destination. Each key file also contains the password-encrypted private key of its maintainer. The key file is signed by this private key to prevent tampering. Note that the compromise of any given (nonserver) key file does not affect the security of other endpoints. Although we could have obtained a similar result with symmetric key encryption, we believe that the current design will allow our system to scale up gracefully through the addition of public key certification.

When a new user is registered, the tunnel server generates a new RSA key and key file for that user. The user's public key is inserted into the server's key file, and conversely, the server's key is inserted into the user's key file. To obtain enough randomness for key generation, we carefully measure the elapsed time (in machine instructions) to perform each of a sequence of disk seeks. These results are then hashed to provide a seed for a pseudorandom number generator. There is substantial evidence that the air turbulence between hard-disk heads and platters contributes sufficient randomness for such purposes.[11]

Both single-user and server tunnel applications use key files, and the credentials stored therein, as a minimum requirement for successful authentication and authorization. Our server software places additional constraints on incoming connection requests. In addition to recording a new user's public key, tunnel servers maintain a small set of tunnel configuration parameters for each user. These parameters define the range of IP address pairs that can be assigned to the server and client pseudo-device, the set of network routing entries that are passed from server to client at tunnel formation, and the minimum level of encryption strength permitted for a tunnel connection.

Creating or initiating a tunnel connection can be a complex task, considering the network path the tunnel connection might traverse. This path can include two intelligent relays, any number of generic TCP/IP relays, and a final tunnel endpoint. Requiring the user to remember such a path would have made the tunnel exceedingly difficult to operate. Therefore, the AltaVista Tunnel stores this information in an external configuration file. Each new user receives both a configuration file and a key file to initialize a newly installed tunnel application. These files provide all the data necessary to run the tunnel application—the user need only press the connect button.

## The Protocol Handler

The AltaVista Tunnel protocol handler is responsible for establishing secure virtual connections between tunnel endpoints and for encapsulating and transmitting redirected IP packets as data. These connections are virtual in that they are composed of several distinct TCP connections joined by relays. Upon the establishment of each new virtual connection, the tunnel endpoints engage in a dialog to agree on security parameters for that connection. For our purposes, a secure connection must have at least the following properties:

- Authenticity—Data received over the channel originates at a known sender.
- Integrity—Data received over the channel cannot be modified in transit.
- Exactly-once delivery—Each datum is received once and only once.
- Privacy—An attacker may not learn the contents of transmitted data by observing the network.

We use cryptography to provide these properties. As discussed in the previous section, key files form the long-term basis for trust between tunnel endpoints. Prior to transmitting data, the parties must perform mutual authentication and agree on a key length, a set of cryptographic algorithms, and a shared encryption key. It is important that keys be negotiated periodically, since this minimizes the benefit an attacker can gain from breaking a specific key. In the current AltaVista Tunnel, we perform the very simple key exchange shown in Figure 5.[2]

1. A sends to B: A, B, $\{P_{ab}\}$, $K_b^{-1}(S_a)$
2. B sends to A: B, A, $\{P_{ba}\}$, $K_a^{-1}(S_b)$

A and B compute: $P_k = \text{Best}\,(\{P_{ab}\} \wedge \{P_{ba}\})$
A and B compute: $S = S_a \oplus S_b$
A and B compute: $K = \text{Reduce}\,(P_k, S)$

**Figure 5**
Tunnel Key Exchange Protocol

Figure 5 describes our key exchange protocol; $K^{-1}(\ )$ signifies encryption with the public component of an RSA key pair. Suppose tunnel nodes A and B wish to share an encryption key. Both can determine their partner's public key from their local key file. As shown in Figure 5, node A invents a random number $S_a$, encrypts it with node B's public key, and sends it to node B's network address. This message also includes $\{P_{ab}\}$, a set of proposed cryptographic algorithms and key lengths that node A considers acceptable for communicating with node B. Upon receipt of message 1, node B similarly constructs and sends response 2. Now nodes A and B can choose $P_k$, a negotiated choice of key length and algorithm, by intersecting sets $\{P_{ab}\}$ and $\{P_{ba}\}$ and then selecting the best available option using an a priori ranking. Both parties can also compute S, a shared key seed, by decryption and exclusive OR. Finally, nodes A and B can produce a shared key by reducing the shared seed to a key in a manner specific to $P_k$. For simplicity, this protocol is executed for every new connection, and by default, a new connection is established every 30 minutes. This technique guarantees that the active key is updated frequently.

Our protocol succeeds because only node B can decrypt message 1, and only node A can decrypt message 2. As a result, both parties can believe that K is known only to each other. An intermediate node cannot control the negotiated key by intercepting message 1 and then retransmitting a modified version to node B. (Note that this represents a denial-of-service attack.) Both node A and node B, however, must take some care in choosing their algorithm proposals. An intermediate node can force the resultant connection parameters P to be the weakest proposal jointly acceptable to both parties. This problem would be eliminated if messages 1 and 2 were cryptographically signed at their origin.

Once the key exchange is complete, it is easy to see how to achieve the essential properties of secure connections. We sign all transmitted data by appending the output of a keyed hash function under K, where a keyed hash function (such as the one described by

Krawcyzk et al.[12]) is a cryptographic hash of data that incorporates a shared secret. This signature guarantees the authenticity of the data (more specifically, the signer must know K) and ensures that any in-transit modification will be detected. Once-only delivery is guaranteed by including a monotonically increasing sequence number in the keyed hash. Since TCP sequence numbers are not secure, an attacker could otherwise insert previously sent data into the data stream. Finally, privacy is obtained by applying a symmetric cipher, such as the RC4 algorithm,[13] to all tunneled datagrams using K to initialize the keying material. Note that since our transport is reliable, having no missing data or out-of-order delivery, it is easy to use a stream cipher for this purpose. Similarly, compression state can be maintained over the lifetime of a connection. This allows for efficient compression of data prior to encryption, although we have yet to implement this.

To implement virtual connection establishment and data encapsulation, we segment the data stream into typed command frames. Using these command frames, we implement a straightforward protocol for relay activation, connection establishment, key exchange, data transmission, failure detection, and connection teardown. Since the data stream is reliable, this protocol is quite simple. Moreover, the data carried by key exchange packets is opaque from the point of view of the connection protocol, and key exchange can encompass multiple round-trips. Therefore, the basic mechanism we use to establish tunnel connections should support other forms of cryptographic credentials and negotiation as new standards for naming, trust management, and key exchange emerge.

In the UNIX server, the protocol handler is implemented as part of the tunnel server daemon, which runs in user space. In the Windows environment, we found that tunneling could not be implemented in user space. Under certain circumstances, the Windows file system can perform remote operations while holding critical system locks. Since the tunnel application cannot run while these locks are held, deadlock ensues. Therefore, we implement the Windows protocol handler in kernel space, alongside the pseudo-device driver. This approach also improves performance by eliminating the need to copy data to user space.

## The Pseudo-Device Driver

In the AltaVista Tunnel, the pseudo-device driver's sole purpose is to redirect outgoing IP packets to the tunnel protocol handler and to reintroduce incoming packets from the protocol handler to the IP stack. Once the tunnel application has set up and authenticated a tunnel connection, it activates the pseudo-device driver to enable redirection of the tunnel packets into and out of the connection. During activation, the IP stack recognizes the new network device and updates the routing table to reflect any newly available routes.

Because of differences in networking architectures, the implementation of this driver is very simple on the UNIX platform and quite complex on the Windows 95 and Window NT platforms. Our initial attempt to implement the Windows pseudo-device emulated an Ethernet LAN. This design became overly baroque due to the need to emulate LAN services such as the Address Resolution Protocol (ARP).[14] Recently, the pseudo-device in AltaVista Tunnel '97 was redesigned to closely resemble a dial-up network adapter, thereby eliminating the need for LAN emulation. We describe all these implementations in this section.

### *UNIX Pseudo-Device*

On the UNIX platform, the pseudo-device driver is a straightforward emulation of a network device. The back end of this network device communicates with a user-level process through a socket interface. The simplicity of this design comes from the fact that the UNIX IP stack delivers packets to network devices without additional encapsulation. Since the physical device layer takes care of Ethernet Media Access Control (MAC) encapsulation, the emulated network device does not have to deal with complexities such as ARP[14] processing. The UNIX tunnel application uses the ifconfig program to activate the pseudo-device, assign an IP address to the device, and insert the address into the routing table.

### *Windows Pseudo-Device*

The first release of the Windows 95 tunnel pseudo-device was considerably more complex than its UNIX counterpart. Under the Windows operating system, most 32-bit network device drivers are implemented using the Network Device Interface Specification.[15] This application programming interface (API) is tailored to handle physical devices, not abstract IP interfaces. In the Windows environment, the network stack must have considerable knowledge of the physical network. For example, the stack must implement the MAC protocols necessary to transmit a packet on a supported medium. As a result, our initial implementation of a network pseudo-device emulated a complete Ethernet LAN, including a gateway host that provides ARP and dynamic addressing services, as shown in Figure 6.

Every Ethernet device has a unique hardware or MAC address. When IP packets are sent over the Ethernet, they are transmitted using these hardware addresses. IP packets with a destination address off the local LAN must be sent to a gateway host router located on the LAN. The tunnel pseudo-device creates an illusion of the complete LAN, including the gateway host, within the device driver and assigns the IP

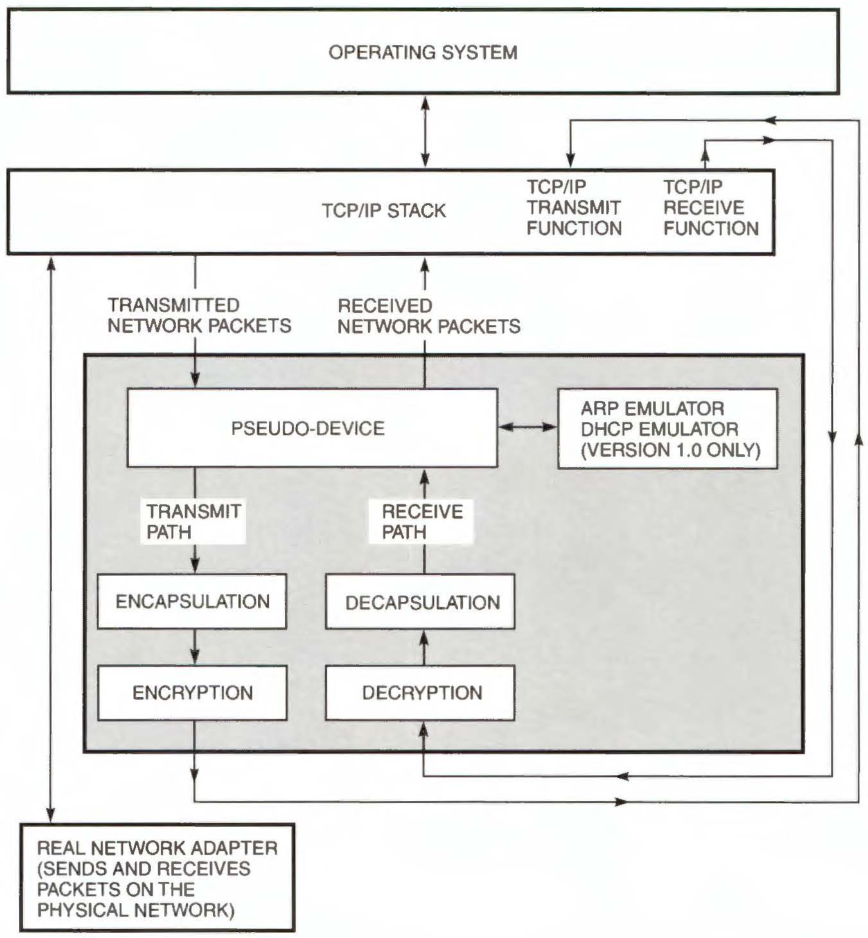


**Figure 6**
Control Flow in the Windows Pseudo-Device

address of the remote tunnel server pseudo-device to this emulated host. The IP stack is fooled into believing there are *two* nodes on the emulated network LAN—the tunnel client, which provides the local node, and the tunnel server as the gateway host to the *real* network or networks on the other side of the tunnel.

When the IP stack prepares to transmit a packet, it must know the MAC address of the destination or the gateway host. If the stack does not know the MAC address, it transmits an ARP packet to the pseudo-device. The pseudo-device responds only to ARP requests for the gateway host MAC address. To resolve this ARP request, the driver includes the functionality of an ARP server. Note that the MAC address must be unique to prevent a conflict with the MAC address of a real device. Clearly, no Ethernet device will ever contain a MAC address of 08-00-2B-00-00-01 or 08-00-2B-00-00-02, which are the first two Ethernet addresses that Digital Equipment Corporation ever assigned. In the AltaVista Tunnel, the first of these addresses always serves as the pseudo-device MAC address; the second serves as the MAC address of the gateway host.

The network pseudo-device in AltaVista Tunnel '97 is simple by comparison. With help from Microsoft, our implementation is now able to emulate a Windows dial-up adapter rather than a LAN. Dial-up adapters are treated specially by the Windows IP stack. No ARP packets are directed at dial-up devices, only one emulated address must be maintained, and information about gateways and dynamic IP addressing can be supplied *after* link establishment. This, of course, perfectly matches the tunnel's operating environment. The control flow outlined in Figure 6 correctly describes the operation of this new pseudo-device implementation; however, as noted, ARP and dynamic address emulation is no longer required.

### *Dynamic IP Address Binding*

Each tunnel is uniquely identified by the IP addresses assigned to the pseudo-device at each endpoint. The tunnel server uses a separate pseudo-device for each active tunnel. The tunnel server implementation could have used a single IP address and pseudo-device for multiple tunnels, because each client is unaware of any other's existence. However, that would have required additional routing complexity at the tunnel server. By using unique address pairs, the routing tables on both the client and server can be maintained easily without platform-specific software. This design also permits conventional packet filtering on the tunnel server. The tunnel address space can be a valid, externally visible or hidden network, thereby supplying an almost unlimited number of addresses.

To facilitate a very large number of registered users for any given tunnel server, we implement dynamic reuse of address pairs. Since dynamic addresses are negotiated at connect time, we need to bind IP addresses to pseudo-devices *after* tunnel connection establishment. For the Windows platform, we use the Transport Driver Interface (TDI)[16] from within the pseudo-device driver to perform both dynamic address assignment and routing table modification.

## Performance and Experience

Tunneling does add overhead to data transmission. This overhead falls into two categories. First, the encapsulating TCP connection adds network overhead by introducing an extra level of framing, plus any acknowledgment and retransmission traffic that is required to support reliable delivery. Second, cryptography adds to the per-packet processing cost, although this does not generally become significant at low speeds. More to the point, transmission of encrypted data defeats the compression present in many modems.

The most significant performance impact is observed when using the Telnet protocol. Because this protocol sends few characters per packet, the encapsulation overhead is quite high. In addition, the overall network path between Telnet client and server can be long enough to make character echoing sluggish. Remember that the round-trip network path may pass through at least two tunnels, a firewall, and potentially several other routers. Thus, to properly support interactive applications, it is essential to choose an Internet path so as to minimize the round-trip latency to the destination network.

The performance is considerably better for noninteractive applications such as File Transfer Protocol (FTP) or Hypertext Transfer Protocol (HTTP) where packets are usually filled to capacity. To prevent IP packet fragmentation, the tunnel pseudo-device reduces the maximum transmission unit size by the amount of the encapsulation overhead. For full packets, the encapsulation overhead is less than 5 percent. We have observed a peak rate for tunneled FTP file transfers as high as 6.4 megabits per second. This measurement was performed over an unloaded switched Ethernet between a 200-megahertz (MHz) Pentium Pro client running Windows NT version 4.0 and a 300-MHz DIGITAL AlphaServer system running DIGITAL UNIX version 3.2. In this case, the FTP client and server programs ran on the tunnel endpoint machines. Without using the tunnel, the same configuration produced throughput averaging 8.8 megabits per second. This translates to a tunnel throughput efficiency of about 73 percent. In both tests, processor usage never exceeded 50 percent.

In another test, we used two 150-MHz AlphaServer systems running DIGITAL UNIX version 3.2 as tunnel endpoints. This tunnel was used to route between two Pentium 166-MHz processors running Windows 95 and LapLink, a popular remote access program for portable PCs.[17] Tunneled file transfers between these computers (with LapLink compression turned off) averaged only 15 percent slower than transfers without the tunnel. Thus, by using the UDP-based LapLink protocol, we were able to achieve a substantially better tunnel throughput efficiency than that reported for FTP. As before, processor usage never exceeded 50 percent. From these simple tests we conclude that tunnel performance is not limited by CPU speed but instead by the networking environment and payload protocol. We also believe that TCP/IP window size may play a role in limiting tunnel throughput.

The cost of cryptography at the client is not a serious performance issue. A 133-MHz Pentium processor can compute RC4 at more than 25 megabits per second and keyed hashes at twice that speed. The cost of server cryptography is mitigated by the fact that most client traffic is bounded by low link speeds and that servers typically run on fast machines.

Digital Equipment Corporation is using the AltaVista Tunnel product to support its mobile workforce and telecommuters. Previously, the company used wide-area, dial-up telephone lines at extremely favorable rates, but more than 30 percent of the IP traffic that used this service had a destination address outside the company. This meant that the company was acting as an Internet service provider (ISP) and was doing so at long-distance rates! A short-term evaluation revealed that users who remotely connected to the company network for more than 10 hours per month would achieve substantial cost savings by connecting through a public ISP and using the AltaVista Tunnel. Local calls are still directly dialed to remote access servers (RAS) located in areas where employee density is highest. In an early pilot, the top 100 RAS users were offered ISP accounts and access using the AltaVista Tunnel. The reduction in monthly telephone costs was dramatic—enough to fund each of the users' ISP accounts for more than a year! In addition, many of these telecommuters can now use higher-speed options such as cable modems or Integrated Services Digital Network (ISDN) to connect to the public network, yielding an overall higher-speed connection into the company than using traditional directly dialed 28.8 Kbps modem access.

At the time of this writing, more than 2,000 DIGITAL employees worldwide use the AltaVista

Tunnel in their daily work. We have observed that a single tunnel server can handle at least 125 concurrent users, although the average number of active users is much smaller. In fact, the ratio of active to registered users rarely exceeds 1 to 10. Currently, DIGITAL offers its employees three tunnel access points within the United States and is planning to deploy additional tunnels overseas.

## Security Risks

Products like the AltaVista Tunnel are not risk free. The most obvious risks have to do with cryptography. Cryptographic security is never absolute. One can only hope to keep the cost of mounting an attack high when compared to the value of a successful attack. Thus, any sensible application of cryptography must be well ahead of any expected attackers in terms of key length and algorithm strength. Barring fundamental change in the science of cryptography, prudent engineering is sufficient to provide this advantage. Since our implementation does not mandate a specific algorithm or key size, the strength of our product's cryptography can improve over time.

As discussed earlier, the tunnel encapsulation protocol protects against many common threats such as eavesdropping, impersonation, replay, and man-in-the-middle attacks. Denial-of-service attacks such as flooding a tunnel server with connection requests remain a problem, however, especially since connection request processing is compute intensive. Newer key exchange protocols, for example, Oakley,[18] preface such costly operations with exchanges of random values that then identify subsequent messages. This technique defeats simple flooding attacks by allowing the server to control the rate at which identifiers are issued. Our product might benefit from this approach, although the benefit would come at the cost of an additional network round-trip.

Attacks on password-protected key files are a greater concern, especially since laptop computers can easily be stolen. If accessed directly, many password-protected containers are subject to dictionary attack, and key files are no exception. If an attacker succeeds in compromising a key file, the attacker can masquerade as the key file's owner. The fact that users are often careless in choosing passwords exacerbates this problem.

Of course, tunnel servers must be carefully protected. A tunnel server not only holds valuable keying information but also enjoys special privileges for firewall traversal. An attacker who can compromise such a machine can compromise an entire network. Therefore, tunnel servers should be handled as carefully as firewalls.

Perhaps the greatest threat posed by IP tunneling is that it extends the perimeter of any firewall it traverses. Because the tunnel traffics in encrypted IP packets, auditing at the firewall is difficult. In addition, there are subtle problems that arise from routine use of remotely connected machines. In the single-user tunnel, we disallow the forwarding of packets not originating on the local machine, but by definition, this cannot be the case for tunnel servers. Consider what happens if a telecommuter uses a tunnel server on his or her home LAN to access a corporate network. The home LAN is then automatically part of the corporate network. Now suppose that a housemate similarly connects to another private network from a machine on the same home LAN. This configuration could allow unintended routing between two private networks! Real-time routing is not the only risk. A computer that is exposed to the raw Internet or to a hostile corporate network could become infected with a virus or Trojan horse program that becomes active only upon tunnel connection establishment.

All these threats are real; however, the benefits to be gained from tunneling are substantial. Any policy that involves the deployment of IP tunnels should carefully counterbalance these risks and benefits. At the very least, machines that use tunneling, especially if IP forwarding is enabled, should be more carefully managed than those directly connected to protected networks.

## Summary

The AltaVista Tunnel was jointly prototyped by researchers at two DIGITAL laboratories, the Cambridge Research Laboratory and the Systems Research Center. The prototype effectively demonstrated that the Internet could be used to reduce telecommuting costs, and within a year, it had grown into a DIGITAL product.[19] Since that time, the AltaVista Tunnel product has evolved to offer support for a variety of client and server platforms, as well as improved performance and enhanced cryptography.

The AltaVista Tunnel is an effective tool for extending corporate networks. By combining tunnels and secure channels, it allows Internet access to supplant leased telephone lines without substantial loss of security. Our deployment of this technology within DIGITAL has cut costs dramatically without substantially affecting network performance. We expect that secure tunneling will play an important role in servicing the telecommuters of the future.

## Acknowledgments

We owe thanks to many people for the success of the AltaVista Tunnel project. Virgil Champlin wrote the IP-in-TCP tunneling code on which we based our current UNIX pseudo-device. Mitch Lichtenberg provided help and guidance concerning all issues related to the Windows operating system. Wick Nichols contributed to our initial design and implementation. The AltaVista Tunnel group turned our research prototype

into presentable software. Our cryptography is implemented using the BSAFE tool kit from RSA Data Security, Inc. Finally, Ed Balkovich got us to work on this in the first place.

## References and Notes

1. W. Cheswick and S. Bellovin, *Firewalls and Internet Security* (Reading, Mass.: Addison-Wesley, 1994): 79–80.

2. B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," *ACM Transactions on Computer Systems,* vol. 10, no. 4 (November 1992): 265–310. This paper is available at ftp://ftp.digital.com/pub/DEC/SRC/research-reports/SRC-083.ps.Z (also available in .ps, ps.gz, ps.zip, and .pdf formats).

3. Generic relays typically forward TCP byte streams from a specific port on the firewall to an (address, port) pair on the internal network.

4. R. Atkinson, "Security Architecture for the Internet Protocol," Internet RFC 1825 (August 1995). This document is available at ftp://ds.internic.net/rfc/rfc1825.txt.

5. R. Atkinson, "IP Encapsulating Security Payload (ESP)," Internet RFC 1827 (August 1995). This document is available at ftp://ds.internic.net/rfc/rfc1827.txt.

6. RSA S/WAN Initiative. For more information about the IPSec Interoperability Forum of RSA Data Security, Inc., Redwood City, Calif., see http://www.rsa.com/rsa/SWAN.

7. GateD—Gateway Routing Daemon, Merit GateDaemon Consortium, http://www.gated.org.

8. J. Mogul, "Using screend to Implement IP/TCP Security Policies," Network Note NN-16 (Palo Alto, Calif.: Digital Equipment Corporation, Network Systems Laboratory, July 1991). Reissued as NSL Technical Note TN-2. This document is available at http://www.research.digital.com/nsl/publications/TN-2.html.

9. P. Zimmermann, *The Official PGP User's Guide* (Cambridge, Mass.: MIT Press, 1995). This book can be ordered at http://www-mitpress.mit.edu/mitp/recent-books/comp/pgp-user.html.

10. R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM,* vol. 21, no. 2 (February 1978): 120–126.

11. D. Davis, R. Ihaka, and P. Fenstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives," *Advances in Cryptology—CRYPTO '94 Conference Proceedings* (August 1994): 114–120. This paper is available at http://world.std.com/~dtd/random/forward.ps.

12. H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," Internet RFC 2104 (February 1997). This document is available at ftp://ds.internic.net/rfc/rfc2104.txt.

13. Information about the RC4 algorithm is available in "Answers to Frequently Asked Questions About Today's Cryptography, Version 3.0, Question 87" from RSA Laboratories, Redwood City, CA at http://www.rsa.com/rsalabs/newfaq/q87.html.

14. D. Plummer, "An Ethernet Address Resolution Protocol," Internet RFC 826 (November 1982). This document is available at ftp://ds.internic.net/rfc/rfc826.txt.

15. Network Device Interface Specification, Windows 95 Networking (Redmond, Wash.: Microsoft Corporation, 1994). This specification is available at http://www.microsoft.com/organizations/corpeval/documents/(48).doc.

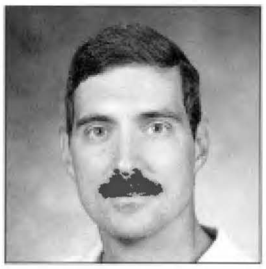
16. Transport Driver Interface Specification (Redmond, Wash.: Microsoft Corporation, 1994).

17. LapLink for Windows 95 software is available from Traveling Software, Inc. at http://www.travsoft.com.

18. H. Orman, "The OAKLEY Key Determination Protocol," Internet Draft (May 1996). This draft is available at ftp://ds.internic.net/internet-drafts/draft-ietf-ipsec-oakley-02.txt.

19. AltaVista Tunnel product information is available at http://altavista.software.digital.com/tunnel/index.htm.

## Biographies



**Kenneth F. Alden**
Ken Alden is a consulting engineer at DIGITAL's Cambridge Research Laboratory. Ken's research interests include novel network appliances, image processing, and digital photography. In previous work, Ken contributed to various other software engineering efforts for the OpenVMS operating system and OpenVMS workstations. He was the lead graphical developer for the VAX/TPU workstation follow-on, the technical director for the European Networking/Workstation Roadshow, and the catalyst for Network Services Integration Services' Internet services development. Prior to joining DIGITAL in 1982, Ken received a B.S. in industrial engineering/software engineering from the University of Michigan in 1981. He has three patents pending, two related to tunneling technology and one on an Internet application that uses audio.

**Edward P. Wobber**
Edward "Ted" Wobber is a consulting engineer at DIGITAL's Systems Research Center. Ted's research interests include computer security, distributed systems, and collaborative systems, and he has published extensively in these areas. Prior to joining DIGITAL, Ted worked for Xerox Corporation in Palo Alto, California, where he designed and implemented networking protocols during the early days of client-server computing. Ted received a B.A from Harvard College in 1975. He holds three patents and has ten patent applications pending.

# Protecting a Private Network: The AltaVista Firewall

J. Mark Smith
Sean G. Doherty
Oliver J. Leahy
Dermot M. Tynan

**Connecting an organization's private network to the Internet offers many advantages but also exposes the organization to the threat of an electronic break-in. The AltaVista Firewall 97 for DIGITAL UNIX protects a private network from malicious attack or casual infiltration by screening all internetwork communication. It enforces the organization's network security policy so that only allowed network traffic can cross the firewall. When installed on a dual- or multihomed host, the AltaVista Firewall applies the principle "that which is not expressly permitted is denied" and uses patented technology to screen each IP packet that attempts to cross it. A highly flexible access control grammar and a comprehensive reporting and alarm system enable the AltaVista Firewall to detect and react to harmful or dangerous events. The AltaVista Firewall also includes an HTML-based user interface to ease configuration and management of the firewall.**

The advent of electronic commerce as a means of conducting business globally has resulted in an increasing number of organizations connecting their internal private networks to the Internet. Most users of the Internet and the World Wide Web (WWW) view the technologies involved as leading edge, but many are unaware that the foundations on which these technologies are built are quite old.

The Transmission Control Protocol and Internet Protocol (TCP/IP) were first developed in 1979. The primary focus then was to ensure reliable communications between groups of networks connected by computers acting as gateways.[1] At that time, security was not an issue because the size of this Internet was small and most of the users knew each other. The base technologies used to construct this network contained many insecurities, most of which continue to exist today.[2,3]

Due to a number of well-reported attacks on private networks originating from the Internet, security is now a primary concern when an organization connects to the Internet.[4] Organizations need to conduct their business in a secure manner and to protect their data and computing resources from attack. Such needs are heightened as businesses link geographically distant parts of the organization using private networks based on TCP/IP.

An organization implementing a secure network must first develop a network security policy that specifies the organization's security requirements for their Internet connection. A network security policy specifies what connections are allowed between the private and external networks and the actions to take in the event of a security breach. A firewall placed between the private network and the Internet enforces the security policy by controlling what connections can be established between the two networks. All network traffic must pass through the firewall, which ensures that only permitted traffic passes and is itself immune to attack and penetration.

This paper comprises two parts. The first part provides an overview of firewalls, describes why an organization needs a firewall, and reviews the different types of firewalls. The second part focuses on the AltaVista Firewall 97 for the DIGITAL UNIX operating system.

It discusses the product's requirements and describes its architecture. It then describes the important aspects of the product's implementation. Finally, future enhancements for the AltaVista Firewall are discussed.

## Firewall Overview

Any organization that connects to the Internet should implement an appropriate mechanism to protect the private network against intrusion from the external network and to control the traffic that passes between the two networks. The mechanism used depends on the value of the asset being protected and the impact of damage or loss to the business involved. Typical reasons for using a firewall to protect a private network include the following:

- To prevent unauthorized external users from accessing computing resources on the internal private network. This is necessary because it is extremely difficult and costly to attempt to secure all the hosts within a private network.
- To control internal user access to the external network to prevent the export of proprietary information.
- To avoid the negative public relations impact of a break-in.
- To provide a dependable and reliable connection to the Internet, so that employees do not implement their own insecure private connections.

A firewall is a device or a collection of devices that secures the connection between a private, trusted network and another network. All the traffic between these two networks must pass through the firewall; this enables the firewall to control the traffic. The firewall permits only authorized traffic to pass between the two networks. The organization's network security policy defines what traffic is authorized. The firewall is immune to attack and penetration and provides a reliable and dependable connection between the two networks. It also provides a single point of presence for the organization when, for example, connecting to a public network such as the Internet.

The fundamental role of a firewall is to provide a control mechanism for the IP traffic between two connected networks. Firewalls provide two types of controls and are categorized either as packet-filtering (packet-screening) or application-level implementations.

Packet-filtering or packet-screening firewalls control whether individual packets are forwarded or denied based on a set of rules.[5] These rules specify the action to take for packets whose header data matches the rule criteria. Typically, a rule specifies source and destination IP addresses and ports and the packet type (for example, TCP and User Datagram Protocol [UDP]). The actions are usually to allow or deny the packet. Packet-filtering firewalls provide a basic level of control over traffic at the IP level. The majority of firewalls of this type are custom-configured routers.

Application-level firewalls disable packet forwarding and provide application gateways (also known as proxies or relays) for each protocol that can cross the firewall. The application gateway relays traffic that crosses the firewall. It can impose protocol-specific and user-specific controls on each connection and can record all operations performed by the user of the connection. This type of gateway therefore allows an organization to control (1) which individuals can establish a connection, (2) when a user can establish a connection, and (3) what operations a user can perform. It also keeps a record of the session for tracking and reporting purposes. Most application-level firewalls are dual- or multihomed hosts that run a modified operating system and special-purpose software to implement the firewall.

These two approaches to implementing a firewall can be compared, using the following criteria:

- Operating philosophy
- Level of control over connections
- Level of logging and reporting
- Ease of use
- Flexibility
- Ease of administration and configuration
- Private network information made available

### *Operating Philosophy*

The operating philosophy that a firewall implements is the fundamental element of the security of the network connection. For maximum security, firewalls must apply the principle "that which is not expressly permitted is denied." That is, unless the firewall permits a connection, that connection is not allowed. Many packet filters allow any connection that is not expressly prohibited (screend is an important exception).[6,7]

Packet filters are unsuitable for use as a firewall because they require the operator to specify carefully what traffic is allowed and what is denied. Application-level firewalls are specifically designed to implement this philosophy, so that each application gateway allows only those connections specified by its configuration and denies all other connections by default.

### *Level of Control over Connections*

Application-level firewalls allow a significantly greater level of control over who can establish a connection and what operations can be performed over that connection. For example, a File Transfer Protocol (FTP) application gateway, with the assistance of a user authentication system, can identify a user who wishes to establish a connection and can control what FTP operations that user performs. For example, the gateway can permit GET operations and deny PUT operations.

Packet filters can only support host-level control over who can establish a connection, with no restrictions on the individuals who can connect and what operations can be performed once a connection has been established.

#### *Level of Logging and Reporting*

Typical packet filters provide basic data logging, and where there is the ability to log traffic, the information available is at the packet level. This makes it difficult to identify and track individual connections when many take place at the same time. No information is available on what operations were performed during a connection. Reporting information is limited to counts of packets passed and dropped and other relatively useless traffic statistics.

Application-level firewalls can log data on each connection. They can identify the individuals who establish connections and what operations they perform. Reports can then list what connections an individual established, what operations were performed, and when they were performed. This facilitates monitoring and maintaining the security of the installation.

#### *Ease of Use*

Packet filters tend to be easier to use because they are effectively transparent for those connections that are permitted. Application-level firewalls often require the user to connect first to the firewall host, and then to their destination on the other network. Recently, transparent application gateways have been developed to address this issue.

When support is needed for a new protocol, it is much easier to reconfigure a packet filter than to develop and distribute a new application gateway. For this reason, users behind an application-level firewall may become frustrated when they cannot connect to the latest Internet developments.

#### *Flexibility*

Flexibility is important for systems integrators constructing custom network security solutions for large organizations and for those involved in electronic commerce. It is important that the individual firewall components can be configured directly.

#### *Ease of Configuration and Administration*

The ease with which an individual can configure and administer a firewall is becoming more important as firewalls are used in organizations that do not possess a high level of technical knowledge. Packet filters are essentially simple but often have a complex rule syntax that makes the task of correctly configuring a packet filter quite difficult. User interfaces could help with the task of configuration, but an expert is usually needed to set up a packet filter properly, because the order of the packet-filtering rules significantly affects the security of the installation.

Application-level firewalls suffer the same disadvantage. The more sophisticated products on the market provide comprehensive user interfaces that guide the user through the task of configuring the firewall, assist in the selection of the appropriate setting for each application gateway, and provide comprehensive firewall management functions.

#### *Private Network Information Made Available*

If information about the private network and the hosts that are on it is available to the external network, an attacker may be able to use this information to subvert the system. Packet filters generally do not hide much information from the outside, which increases the risk of a break-in. Application-level firewalls usually appear to be an end node instead of a router to another network; therefore, they can significantly reduce the amount of information that is made available to a potential attacker.

## AltaVista Firewall Requirements

This section discusses the functional requirements for firewalls and reviews the product requirements that apply to the AltaVista Firewall.

#### *Functional Requirements*

The major functional requirement of a firewall is that it protects a private network from unauthorized external access. A firewall itself must be resistant to subversion and must also ensure that other, less secure hosts within the private network cannot be subverted by an external host.

A firewall must provide a central location for controlling network traffic and for implementing an organization's network security policy for its external network connection. Because many Internet-based services are inherently insecure, a firewall must provide an organization with the means to disable some services and restrict others in accordance with their security policy.

It is also critical that the firewall logs all network traffic, so that a record is retained of all connections established between the private and external networks. Support for the management and retention of network traffic logs is also required to assist tracking of potential and actual break-ins and other security-related activity.

A firewall must be reliable so that users are not inconvenienced by sudden losses of connectivity. As the Internet becomes another means of conducting business, an organization's firewall must ensure that loss of service due to security lapses or other failures is minimized. It must also provide a point of presence for an organization on the Internet.

A firewall must be easy to use. Historically, firewall administrators were required to have in-depth knowledge of Internet protocols; they constructed their firewalls manually. Today's organizations find it difficult to obtain the necessary expertise; they require that their firewall be easy to configure and manage, without sacrificing quality of security and service. Users also require protection without modifying their usage of the Internet. They expect the firewall to protect them without obstructing their work. A firewall must be as transparent as possible to users establishing connections between the private and external networks.

### *Product Requirements*

In addition to firewall products, DIGITAL also offers a custom firewall installation service as part of its network services. The AltaVista Firewall has some of its origins in technology developed for systems integration engineers constructing custom firewall solutions for large customers. These engineers continue to build custom solutions using the AltaVista Firewall to provide the basic firewall technology. The AltaVista Firewall must also be designed to accommodate the needs of integrators delivering custom firewalls.

As described above, there are two types of firewall. Typically, filter-based firewalls perform better. Most detailed product evaluations compare the performance of the products under review, so the AltaVista Firewall must equal or better the performance of its leading competitors.

Most firewalls require that the administrator log on to the host at the console—remote logins are not enabled for security reasons. However, as organizations centralize their network management operations, a key requirement for the AltaVista Firewall is to provide secure remote management functionality.

The AltaVista Firewall must itself be secure, and the host it is installed on must also be secure. It is a product requirement that, while being installed, the product secures the operating system against attack.

## AltaVista Firewall Architecture

This section describes the constraints that applied to the design of the AltaVista Firewall. It then introduces the major product components and summarizes the main functions they provide. Then it lists the steps taken to establish a connection through an application gateway.

The following constraints were applied to the design of the AltaVista Firewall:

- In every design decision, the security of the firewall must be the primary concern.
- The basic installation and use of the firewall must be simple and must be guided by a simple user interface.
- All firewall component functionality must be configured using text-based configuration files so that systems integrators can install custom solutions. These files must be consistent across platforms, so that the same set of configuration files can be used on different platforms to configure heterogeneous firewalls.
- The performance of the firewall is an important concern. Application gateways must run as daemons to improve performance and avoid process start-up delays.
- It must be possible for engineers in the field to extend the functionality of the AltaVista Firewall without requiring code changes to the product.

The AltaVista Firewall comprises the following three major components:

- The graphical user interface (GUI) provides the functionality for the user to configure and manage the firewall and manages the configuration files that specify how the firewall will operate.
- The application gateways and support daemons provide the network security specified by the configuration files.
- The kernel is hardened and modified to provide the extra security functionality required for the firewall's operation.

### *The Graphical User Interface*

The GUI is based on a set of hypertext markup language (HTML) template files, computer graphic interface (CGI) scripts, and an HTML browser. It is menu-driven and guides the user through the configuration and management of the firewall. The design of the GUI involved many trade-offs between simplicity of the UI and access to the firewall functionality. We often chose to maintain the simplicity of the UI at the expense of functionality for two reasons:

1. We must protect unskilled installers from installing the firewall in an insecure manner.
2. Skilled installers must still have access to the functionality through the configuration files.

The CGI scripts are written in the C language and use a set of HTML templates to generate the GUI pages. Few static pages are used, so the firewall can be modified in the field to add more application gateways, authentication methods, alarms, and alarm actions.

Because the GUI is HTML-based, it can be ported easily to any platform that supports an HTML browser.

### *The Kernel*

The DIGITAL UNIX kernel has been hardened to protect the firewall host and modified to add functionality required by the firewall. The following modifications were made to the kernel to improve security and to support the operation of the firewall:

- Two mechanisms have been added for protection against routing attacks.

  We added two mechanisms to the kernel to prevent attackers from using routing information to launch an attack through the firewall. The first mechanism allows the firewall to be configured to ignore requests for it to change its routing tables. These requests come in the form of Internet Control Message Protocol (ICMP) redirect messages,[2] which the firewall ignores. The second mechanism allows the firewall to be configured to drop all IP packets that have source routing options set in the packet header. Source routing can force a packet to take a specified route through the Internet and is not normally used. IP packets with source routing options specified are considered to be evidence of an attack, so it is valid for the firewall to ignore these packets.[8]

- Interface access filtering has been implemented to prevent IP spoofing.

  Interface access filtering provides a mechanism to specify what IP packets are to be accepted on a particular network interface. The source address for each packet is inspected and compared against a filter list. The packet is passed through to the kernel or dropped, depending on the corresponding filter list action. This provides the ability to prevent attackers on the external network from forging IP packets so that they appear to come from trusted hosts on the private network.[8]

- Interface trust group support has been implemented to support packet filtering.

  Trust group support provides a mechanism to specify a color for a network interface. When a packet arrives on a given interface, the kernel marks the packet with the color of the interface. The application gateways and the packet-filtering daemon can use this information to apply filtering rules to allow or deny the packet.

- Transparent proxy support has been implemented to support transparent application gateways. Kernel support for transparent proxying is described in the section on Packet Filtering.

### *The Firewall*

The main functionality of the firewall is provided by the application gateways that provide connection services to users on each side of the firewall. A number of daemons also implement common services to support the operation of the application gateways. These daemons are described here:

- screend: The packet-screening daemon provides packet filtering and transparent proxying functionality.
- alarmd: The alarm daemon provides logging and alarm functionality to the application gateways and other firewall components.
- authd: The authentication service daemon provides user authentication services to the application gateways.
- dnsd: The name service daemon provides the Domain Name Service (DNS) to the application gateways and to the users of the firewall.
- fwcond: The firewall control daemon monitors the application gateway and support daemons that must run on the firewall and restarts any that stop operating.

In addition to these support daemons, the following statically linked libraries provide common services to the application gateways and the other firewall components:

- The access control list (ACL) library allows or denies access to clients based on the appropriate access control list.
- The firewall logging (fwlog) library provides a uniform logging format for all firewall components. This library also provides the interface between the other firewall components and the alarm daemon.

The main features of the application gateways, support daemons, and libraries are described in more detail below. Figure 1 and the following steps show how these components interact when a network connection is established through an application gateway.

1. The application gateway receives the connection request.
2. The application gateway sends requests to the name service (dnsd) to get the host name of the client and the host name or IP address of the server.
3. When the application gateway gets all the information available to it from the name service, it sends a request to the ACL system to ask whether to allow the connection. At this stage, the gateway does not have an authenticated name for the user attempting to connect, so it asks the ACL system whether unknown users are allowed to make the requested connection.
4. If the connection is allowed, the application gateway makes a connection with the server. Data can now flow between the client and the server through the firewall. The gateway also generates a log message for the connection.
5. If the connection is denied by the ACL system, it is still possible that the connection is allowed for certain users, so the application gateway prompts the user for a user identifier so that it can be authenticated.
6. The user specifies an identifier to the application gateway. The gateway sends a request to the authentication service (authd) to authenticate the user and initiates an authentication sequence.
7. The authentication service responds to the application gateway with a challenge for the user. The gateway displays the challenge to the user.

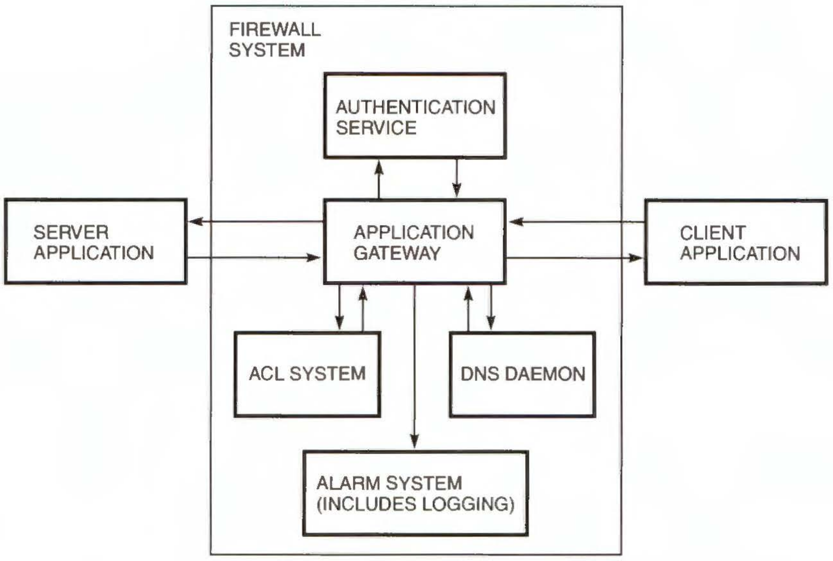


**Figure 1**
Interaction between Application Gateway, Support Daemons, and Libraries

8. The user uses this challenge to generate a response. The user then enters this response. This is passed by the application gateway back to the authentication service. The authentication service uses the response to authenticate the user and confirms or denies that the user is authenticated to the gateway. If the identifier specified is unknown, the authentication service fakes an authentication sequence and then denies authentication. When authentication is denied, the authentication service logs an event that may result in an alarm being triggered.
9. If the user is authenticated successfully, the application gateway sends another request to the ACL system, this time with the additional information of the authenticated user identifier.
10. If the connection is denied, the application gateway logs an event, which may result in an alarm being triggered. The user can try to authenticate again.
11. If the connection is allowed, the application gateway logs a message specifying that the connection has been accepted and makes a connection with the server. The client and server may now exchange data.
12. When either the client or the server terminates the connection, the application gateway logs two messages: one specifies that the connection has terminated, and another reports statistics such as duration and amount of data exchanged.

### *Packet Filtering*

The AltaVista Firewall for the DIGITAL UNIX operating system provides both application-level and packet-filtering functionality. Packet-filtering functionality is provided by the well-known packet screening daemon, screend,[6,7] which is shipped with DIGITAL UNIX. The AltaVista Firewall replaces the standard DIGITAL UNIX version of screend with a modified version that extends its packet-filtering functionality and provides support for transparent proxying. This section provides an overview of how screend operates as part of the packet routing in a dual- or multi-homed host. It then outlines why a firewall requires transparent proxying and describes how screend is used to implement transparent proxying.

Screend is a daemon that runs in user space. It examines every IP packet that is being forwarded by a firewall and decides whether that packet should be forwarded or dropped. Screend bases its decision on a set of filter rules specified in a configuration file that it reads at start-up. When an IP packet is received by a DIGITAL UNIX host, the first check that the DIGITAL UNIX kernel performs is whether or not the packet is destined for this host. If the packet is destined for another host, then most hosts discard the packet. However, a DIGITAL UNIX host can be configured (by using iprsetup to switch on ipforwarding) to route packets that are destined for other hosts. This is useful when hosts that are not on the same physical network need to communicate. In this case, a number of routers must exist between the communicating hosts that can pass packets from one physical network to another. These routers have at least two network interfaces, as shown in Figure 2. The router can then be configured to pass packets between the networks on each of its interfaces. In Figure 2, for example, if the

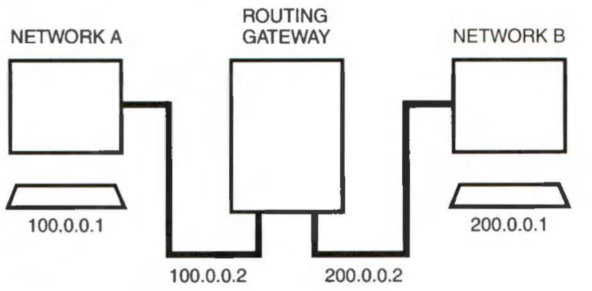


**Figure 2**
Routing Packets between Networks

host on network A with address 100.0.0.1 wants to send a packet to a host on network B, the packet is initially sent to the routing host. The router receives the packet on interface 100.0.0.2. The routing software in the kernel then sends the packet to the host on network B.

When a DIGITAL UNIX machine with ipforwarding switched on receives a packet that is not destined for itself the packet is passed to a forwarding module in the kernel. This module decides whether the packet can be forwarded and what network interface to send it to. When screend is running on the host, a third operation is inserted after the host has decided that the packet is not for itself and before the packet is sent to the forwarding module. This intermediate function sends the packet to the screend process, which decides whether the host is allowed to forward the packet, based on its set of rules. If screend rejects the packet, it is discarded. If screend accepts the packet, it is sent to the forwarding module as normal.

The AltaVista Firewall is primarily an application gateway firewall. This means that all hosts interacting with the firewall route packets to the application gateways on the firewall. The application gateway then opens a connection to the remote service if the requested connection is allowed. Because the firewall never has to forward packets, IP forwarding could be switched off at the firewall, and screend need not run. If a user attempts to connect to a server on the other side of a firewall, however, the user must understand that the application gateway is present. Instead of connecting directly to the server wanted, the user must first connect to the firewall and then request an application gateway to establish a connection to the remote server. This can be especially awkward with some GUI-based clients.

Transparent proxying was developed to overcome this problem. When an application gateway that supports transparent proxying is running on the firewall, the user can make what appears to be a direct connection to the remote server. In reality, the connection from the client machine is routed through the firewall, which intercepts the connection and redirects the packet to the appropriate gateway. Then, if the gateway allows the connection, it builds a new connection between the firewall and the server. From the user's perspective, the firewall is transparent and has played no part in the establishment of the connection. However, the gateway processes all the user's requests. When a connection is made by a transparent gateway between a client and a server, two TCP connections exist. The first is between the client machine and the firewall. The client host address and the server host address are the source and destination addresses of packets on this connection. The second TCP connection is between the firewall and the server host. The firewall and server host address are the source and destination addresses of packets on this connection.

The AltaVista Firewall uses screend to implement transparent proxying. To do this, changes were made to the DIGITAL UNIX kernel and to the screend application. The most significant change to screend was to add a third option to how screend deals with a packet. Although the standard screend implementation can only accept or reject packets, the implementation of screend that is shipped with the AltaVista Firewall can also proxy a packet. In this case, the packet is not passed on to the IP forwarding module in the DIGITAL UNIX kernel on the firewall. Instead, the kernel sets a flag in the packet to indicate that it is being proxied, and the packet is sent back into the IP input module. When the IP input module detects that the packet is being proxied, it treats the packet as if it were addressed to itself, and passes the packet to its own TCP input module. This then passes the packet on to the application gateway that is listening on the appropriate port. The application gateway determines what the intended destination for the packet is and, subject to the access control specified for the gateway, creates a new connection to the requested server.

Screend imposes an overhead on packet forwarding. Most of this overhead occurs because of the need for a system call from screend to deal with every packet traversing the AltaVista Firewall. The designers of the AltaVista Firewall decided that this overhead would seriously degrade performance, and a cache for screend was implemented in the DIGITAL UNIX kernel. This cache maintains the ways in which screend deals with most open connections, so that only new connections suffer the overhead of a system call from screend.

Screend can also be used as a packet filter for connections to pass through the firewall at the IP level. Packet-screening routers are not as safe as application gateways, so the AltaVista Firewall does not use screend to filter packets through the firewall and does not provide a feature in the user interface to configure screend. Experienced firewall administrators, however, can configure screend to allow IP-level connections to pass across the firewall when they need to support protocols for which application gateways are not available.

### *Application Gateways*

As previously stated, the AltaVista Firewall is primarily an application gateway firewall. As the name implies, application gateways operate in user space at the application layer of the open system interconnection (OSI) model, controlling the traffic between directly connected networks. A separate gateway listens on the appropriate TCP/UDP port on the firewall for each protocol that the firewall relays. This approach provides a high level of control over all major TCP/IP services and allows extensive logging, neither of which packet-filtering techniques can provide. For example, it is possible to allow only authenticated users to copy files, using FTP, out of a private network using the FTP PUT command, while also allowing anyone to copy files from external servers into the private network using the FTP GET command. This high level of control is also apparent in the log files, which show the source and destination addresses (both in IP format and as a fully qualified domain name [FQDN]), as well as the commands executed, names of files transferred, and the number of bytes transferred in each direction across the firewall.

By default, the AltaVista Firewall is configured without any screend packet filter rules that might allow network traffic to cross the firewall at the IP level. Therefore, traffic traveling in both directions must be directed to the firewall where it will be relayed by the appropriate application gateway. This highlights another significant advantage of application-level firewalls—the ability to perform network address hiding. This allows customers to use RFC 1918 (Address Allocation for Private Internets)[9] addresses on their internal network, since the gateway hides the IP addresses of the hosts on the private network inside the firewall.

Historically, application-level firewalls performed poorly, because a new process was forked to handle each connection. For FTP and Telnet, this is not a serious issue; however, with Web traffic, a single URL request may result in numerous other requests for in-line images. The overhead involved in forking a new process for each connection is not acceptable. The application gateways used in the AltaVista Firewall have been designed to address this limitation, without sacrificing security.

Each application gateway is implemented using a single process to handle all connections. To process each connection, the gateway multiplexes between open I/O file descriptors using the select() system call, performing nonblocking reads and writes, and buffering all data until it can be passed on to the client or server. This nonblocking functionality allows the gateway to handle other connections without having to wait for the client/server to process the data already sent to it, or to wait for the name service daemon or the authentication service daemon to return with a response.

This requires that each of the support daemons, such as the alarm daemon, the name service daemon, and the authentication service daemon, must also process requests from the application gateways without blocking. This design resulted in significant improvements in performance and in memory usage. Performance has also been improved by caching name service lookups in the name service daemon. Independent tests have demonstrated that the AltaVista Firewall 97 performs better than packet-filter firewalls, even at Fast Ethernet speeds.[10]

The AltaVista Firewall currently provides the following application-level gateways:

- WWW (including Hypertext Transfer Protocol (HTTP), gopher, FTP, and secure socket layer (SSL) forwarding)
- Simple Mail Transfer Protocol (SMTP)
- FTP
- Telnet
- RealAudio/RealVideo
- SQL*Net
- Finger
- Generic TCP
- Generic UDP

Web traffic comprises the majority of activity on a typical firewall. The WWW application gateway includes specific functionality for

- Blocking Java class files to protect users
- URL filtering to deny access to certain Web sites
- Caching Web documents to improve performance

### *Alarm System*

The AltaVista Firewall uses a dedicated alarm system to monitor the security of the firewall and to respond to attempts to circumvent the security of the firewall. It can also respond to less serious anomalies such as incorrect passwords. The alarm system is implemented by the alarm daemon alarmd, which monitors events generated by application gateways or other servers on the firewall. These are communicated to alarmd by means of log messages.

The alarm system associates one or more alarms with each event. Each alarm includes one or more actions to take when the event occurs. The alarm that is triggered when an event occurs depends on the state of the firewall, that is, the current level of security awareness of the firewall. The state of the firewall is represented using the colors green, yellow, orange, and red. Each color represents a different security awareness level, ranging from under no threat to under serious threat.

The following list describes each state and the level of awareness associated with the state.

- Green: The firewall has not detected any events, and all appears normal.
- Yellow: The firewall has detected one or more events that may indicate a malicious attempt to compromise the firewall or the private network. If no further event occurs in the next two hours, the firewall returns to the green state.
- Orange: The firewall has detected events that are construed as malicious. Events that cannot be created by harmless or accidental operation cause escalation to this state. For example, if the DEBUG command appears during a mail (SMTP) session, the firewall cannot revert from the orange state to the yellow state; the firewall administrator must explicitly set the state back to a lower level once the threat has been analyzed and appropriate action taken.
- Red: The firewall has detected events that may result in a breach of the security of the firewall or of the private network if not addressed immediately. When an escalation to this state occurs, the interface to the external interface is usually disabled immediately, preventing any further IP traffic from that network.

The current firewall state is constantly displayed in the background of the GUI and on the console monitor. The alarm system can be configured manually or using the GUI and allows the firewall administrator to specify the alarm to be triggered for each event occurring at one or more firewall states. Each alarm specification includes one or more actions. For example, an alarm can advance the firewall state, shut down an application gateway or the firewall, and notify the firewall administrator. The administrator can write shell scripts for additional specific actions to take. This can enable the firewall to actively counter a threat to its security or that of the private network.

### Authentication Service

When users log on to most computer systems, they specify a password to prove their identity to the system. User authentication is the process by which a computer system verifies the identity of a user or entity through a unique password.

The authentication service on the firewall provides a mechanism by which the identity of a user can be verified. This allows organizations to implement security policies that specify that only certain users are allowed to use particular services to access resources through the firewall. When users wish to establish a connection, they must first identify themselves so that the firewall can associate them (via their identifiers) with the connection. This identifier is then presented by the application gateway to the ACL system, which then decides if the connection request will be allowed or denied.

Although users logging directly into a computer system or an internal network run certain risks of having their passwords discovered by another person, most organizations are happy to use what are termed reusable passwords to protect access to computer resources within a private network. However, when a user wishes to gain access from an external network through the firewall to an internal network, the use of reusable passwords presents an unacceptable risk due to the availability of IP packet sniffers. A packet sniffer is a tool that allows an attacker to read the data in an IP packet and identify a user's identifier and password without the user becoming aware that this has happened. It is clear that a stronger form of user authentication is required for use with a firewall that controls access between two networks.

Several authentication mechanisms have been devised that are based on the concept of a one-time password. In the one-time password scheme, a particular user has a mechanism to generate a new password each time he or she is requested to respond with a password, and the authentication system has a similar mechanism to validate the one-time password as correct for that user. Most one-time password systems are based on a shared secret that is used in conjunction with some software or a hardware device (commonly known as a handheld authenticator [HHA]). Typically, the system challenges a user, often with a value. The user then uses an HHA to generate a response. The authentication system receives the response, and determines if it is the one expected from the user. Further attempts to authenticate involve a different challenge and a correspondingly different response.

The AltaVista Firewall supports a wide range of user authentication mechanisms and can be easily modified to support additional mechanisms if necessary. The authentication mechanism used to authenticate a user can be different depending on whether the access requested is inbound or outbound. This allows an organization to apply less stringent controls to accesses from the internal network. Users are registered with the authentication system through the GUI, and their details are stored in a database. This database includes the inbound and outbound authentication mechanisms that apply to the user and an expiration date for the record.

The authentication service is implemented using an authentication service daemon (authd) and a set of authentication server daemons. Authd runs continuously on the firewall and accepts connections concurrently from application gateways requesting authentication of users. The authentication server daemons implement support for each authentication mechanism. Each of the authentication servers implements a challenge-response authentication service and is started by inetd when a connection is made to the port on which the service is provided.

When an application gateway on the firewall requires a user to supply an identifier, a sequence of actions occurs as follows:

1. The gateway connects to authd and specifies the identifier for the user.
2. Authd accesses the user database and, after checking that the user is registered and that the user record has not expired, determines what mechanism to use to authenticate the user.
3. Authd contacts the relevant authentication server and passes the user identifier to it.
4. The authentication server generates a challenge for the user. Authd passes this challenge to the user.
5. The user generates a response, which the gateway provides to the authentication server, and validation proceeds.
6. If validation fails, the gateway is informed and access is denied. If validation succeeds, the user may proceed.

In the event a request is received to validate a user who is not registered or whose user record has expired, authd will fake a user authentication sequence. This additional security measure ensures that a potential attacker cannot determine if the specified user identifier is valid.

Adding a new authentication mechanism is simple, because configuration files specify the port to contact for each type of authentication mechanism, and a common authentication protocol governs the interaction between authd and the authentication servers.

The authentication protocol used between authd and the authentication servers is modeled on the FTP protocol but is much simplified. All user authentication sequences involve an initial step in which the client (authd) specifies the user's identifier to the authentication server, followed by one or more challenge-response cycles in which the server requests a reply from the client (by challenging the server for some information), and the client responds. The server processes each response and will either issue a further challenge to the client or issue a result indicating if the user is authenticated or not.

The authentication service supports the following authentication mechanisms:

- SecureNet Key
- CRYPTOCard
- WatchWord key
- SecurID card
- S/Key
- Reusable passwords (for outbound connections only)

### *The Name Service*

Unlike the traditional model of the Internet in which all DNS[11] information is available to all hosts, certain organizations may decide to prevent external hosts from viewing information about internal hosts. Restricting this type of information allows an organization to prevent a would-be attacker from gaining a foothold to launch an attempt to subvert the security of a private network. For example, an attacker may support a claim to be an employee of an organization by showing knowledge of the network structure, or a recruitment agency may identify personnel engaged in development activities.

Traditionally, hiding information about internal hosts required providing two separate name servers. An internal name server ran on a server behind the firewall and handled internal DNS queries. The firewall ran the second name server and handled DNS queries from external hosts. In this way, the complete name service database was available to internal hosts, and the restricted database (which usually contains records for the firewall itself and any external WWW or FTP servers) was available to external hosts. The drawback of this approach is that a second host was required to run the internal name service.

If an organization is not concerned that its internal name service is available to the outside world, the firewall can act as a name server for both internal and external queries. This approach raises some questions regarding how information is delivered. For example, the mail exchange (MX) records specified for internal hosts will be incorrect for hosts on the external network. If the proper hierarchy is shown for mail delivery (that is, the internal host has the lowest MX priority, followed by the local mail hub, then the main corporate mail hub, and then the firewall itself), external hosts will attempt to connect unsuccessfully to the three internal hosts before eventually sending the mail to the firewall. Although this does not consume considerable bandwidth, it does cause a noticeable delay in mail delivery to hosts on the private network.

The AltaVista Firewall name service is implemented using a name service daemon (dnsd). Dnsd acts as a gateway accepting DNS queries from both the private and external networks. It also accepts DNS queries directly from application gateways and supports daemons running on the firewall. Two name servers based on the standard Berkeley name daemon (bind) also run on the firewall in a protected environment. One acts as an internal name server containing information concerning the full database of internal hosts; the other acts as an external name server containing only selected information. Both name servers are configured to accept requests from dnsd only.

The name service daemon dnsd considers both the origin and the type of each DNS query it receives. DNS queries that originate on the private network are treated differently from queries that originate on the external network. Requests received by the firewall are forwarded to one or the other of the name daemons, based on the form and origin of the question.

Table 1 shows the relationship between queries originating on the external and internal networks, as well as queries by gateways and other daemons running on the firewall. An authoritative query is a query that concerns a host within the domain of the firewall. Queries from external sites, for domains other than those for which the firewall is authoritative, are rejected by default, though this behavior can be disabled, allowing such queries to be forwarded to the external name server.

The main advantage of this dual-DNS system is that it removes the requirement for a second host to run the internal name service. It also allows precise control over the dissemination of name service information to hosts on the external network. The AltaVista Firewall GUI allows an administrator to enter a host name and specify whether or not that particular host name is visible to the external network as well as to the internal networks. Previously, it was possible to specify only whether or not all internal hosts were visible, but the AltaVista Firewall now allows this decision to be made for each host. The MX records are correctly generated so that external hosts do not see MX records for hosts or mail hubs on the private network. Similarly, if the installation includes an external mail hub, this hub will not hold an MX record for internal machines when viewed from the internal network.

The firewall does not permit zone transfers to external secondary name servers unless they appear in a firewall list of secondaries. A zone transfer involves transferring the full information for a domain from the primary domain server to one or more secondary servers. Any external secondary name server that is authorized to receive zone transfers can still only see the domain information as it exists from the perspective of the external network. This means that hosts on the private network that cannot be resolved from the external network will not be included in any zone transfer initiated from the external network. Zone transfers from secondary name servers on the internal network do not need to be authorized.

### *Access Control System*

Access control is a core function of a firewall. The access control system in the AltaVista Firewall provides a powerful, flexible, and secure means for administrators to define who can use the application gateways on their firewall. The ACL system is implemented using a rule definition language, a library of functions used by all the application gateways to load and interpret the rules, and a comprehensive user interface to allow firewall administrators to exploit most of the power of the language.

The architecture of the ACL system is very simple. It is implemented as a static library. This static library is linked to an application gateway or any server that controls access. The API to the library is minimal; it consists of a function to load a rule set from an ACL file and a function that takes the details of a user's request as parameters and returns a decision to deny or allow the action based on the rule set. The AltaVista Firewall has been designed so that each application gateway or server that uses the ACL system has a separate rule file. However, the architecture of the ACL system does not require this, and ACL file sharing is possible.

In the rest of this section, we describe the requirements for the ACL system and how they were addressed.

- The access control system must be reliable. Since access control is a core function of a firewall, the design team considered the reliability of the ACL system as primary. The ACL system was designed as a separate library that is statically linked to the application gateways. Static linking is used to ensure that this component cannot be easily replaced by an agent attempting to subvert a gateway. Implementation as a separate component means that all gateways and other servers that require access control use a common service, and that this component can be tested thoroughly and independently. This approach also had the advantage of allowing more project resources to be allocated to the design, implementation, and testing of the ACL system.
- It was clear that a powerful language was required to define a wide range of security policies. A firewall administrator must be able to grant and deny access to individual users, to hosts, and to groups of users and hosts. The administrator must also be able to specify times at which access is allowed.
- Because the access control language was extremely flexible, it was considered that policies must be fail-safe and tend to deny rather than allow access. Several features of the language implement this requirement:
  - An implicit default rule states "If there is no rule granting access to a request, then that request is denied." This default rule cannot be altered.
  - The order of rules in the ACL file is not important. If there is a conflict between rules, a deny rule takes precedence.

**Table 1**
Handling DNS Queries

| | **Queries for which the firewall is authoritative** | **Queries for which the firewall is not authoritative** |
|---|---|---|
| Queries from internal network | Internal name server | External name server |
| Queries from external network | External name server | Reject the query |

- A blacklist rule has been defined. This is a simple statement that takes a list of host names or addresses. If a host appears in a blacklist statement, then no user can access that host using the firewall, and no user can access the firewall from the blacklisted host.
- Time-based rules can be specified to deny or allow access for particular periods during a day or week.
- A GUI interface to the ACL system provides an interface to most features of the system without requiring the firewall administrator to generate the ACL files manually.

- The interface to the ACL system is very simple. The API consists of just two functions that an application gateway must call. The first function loads the rule set that the gateway will use, and the second function takes the details of a request a user is making as parameters and returns a deny or allow decision.
- The ACL system is also well integrated with the other systems in the firewall. For example, the ACL system performs its own logging of each user request and the resulting decision to deny or allow access.

The ACL grammar is a simple rule-based language that supports the definition of a list of deny and allow rules for application gateways. Each application reads an ACL file to load the rule set it uses to make access control decisions.

The core of the ACL system is the algorithm used to allow or deny access through the gateway. Each time the user makes a request to an application gateway, the gateway builds a data structure describing the user and the requested action and makes a call to the ACL system. The data passed to the ACL system includes the following information:

- The canonical form of the FQDN of the host from which the user is making the call.
- The IP address of the host from which the user is connecting.
- The user identifier, if the user has been authenticated.
- The action the user is requesting.
- The server that the user wishes to access. Note that this could be the firewall itself.

The ACL system uses the following algorithm to decide whether to grant or deny access to a user request.

- Search through all the time-based rules. If any time rule denies access to the requested operation, log that a request was denied because of a time rule and set a flag to indicate that the request was denied.
- Search through all the blacklist rules. If the source or target host is blacklisted, log that a request was denied because of a blacklist rule and set a flag to indicate that the request was denied.
- Search through all the deny rules. If access is denied because of an explicit deny rule, log that a request was denied because of a deny rule and set a flag to indicate that the request was denied.
- If the flag to indicate that the request has been denied is set, return 'DENY' to the gateway.
- Search through all the allow rules. If access is granted because of an explicit allow rule, log that a request was granted and return 'ALLOW' to the caller.
- Log that access was denied because no rule matched the request and return 'DENY' to the caller.

Note that denying one request can generate up to three log entries: access can be denied because of a time rule, a blacklist rule, and a deny rule. If all three rules are triggered, they are all logged. The decision to implement logging in this manner was made to ensure that logs were complete and to ensure that all applicable alarms were triggered. Note also that if any deny rule is triggered, the ACL system does not even look at the allow rules. This means that the deny rules take precedence over the allow rules. Also note that if no rule triggers that the request will be denied, the request is denied. This implements the requirement that any request that is not explicitly allowed is denied.

### *Reporting and Logging*

It is a primary function of a firewall to keep an audit trail of all network traffic for both allowed and denied connections. Most authors on the subject of firewalls stress the importance of maintaining comprehensive logs so that break-in attempts can be identified quickly, and the necessary actions can be taken against the attacker. These actions can include contacting the firewall administrator of the host the attacker is using, blacklisting the host the attacker is using, or temporarily shutting down some of the services that the attacker is trying to exploit.

Logging in the AltaVista Firewall is implemented using a common library. This library is statically linked to all the firewall components and provides a uniform logging interface and uniform entries in the firewall's log files. Static linking is used to ensure that this component cannot be easily replaced by an agent attempting to subvert a gateway. Each firewall component initializes the logging function on start-up, passing it an acronym that the library uses to tag all subsequent entries in log files. When a firewall component calls the logging library to write an entry to the log files, it specifies a flag indicating the type of the log message and the log information. Log messages are of the following types:

- FW_LOG. The message is informational.
- FW_WARN. The message is a warning; there may be some security implications. This level usually

indicates that some configuration information expected by the component is not present or cannot be accessed.

- FW_EVENT. This message is a security event. Event messages are detected by the alarm system as described above.

All logged messages include the following information in the log entry written to the log file:

- The current date and time
- The component generating the message
- The log message type
- The log message information

As well as ensuring that all firewall activity is logged, it is also critical that the firewall manages the logs files that are generated to ensure that the log information is retained for later analysis. The AltaVista Firewall automatically archives logs files on a monthly basis and includes features to allow for the retention and/or deletion of log archives after one year. The firewall also monitors log disk space usage and will automatically inform the firewall administrator if the amount of space available falls below a specified size. If the log disk is full, the firewall will disable itself so that no network traffic can occur that is not logged.

The AltaVista Firewall also provides comprehensive traffic-reporting facilities, including an automated reporting service and a GUI-based custom report generator. The firewall's reporting system allows a firewall administrator to choose from a number of report types, including summaries or detailed information, on individual gateways or on all firewall services. Reports can be generated for daily, weekly, and monthly periods. These reports can be mailed automatically to a specified distribution list.

### Remote Management

Firewall management has rapidly become a key issue for organizations implementing connections to the Internet, or for organizations required to protect several parts of their internal private network. Typically, firewall products allowed operator access only at the firewall system's console to ensure the security of the installation. Most organizations today prefer to centralize their network control operations and expertise. Firewall products that require console-only access are often not considered. Also, many organizations place their firewalls at separate locations where console access for monitoring and control may be restricted for various reasons. The AltaVista Firewall's remote management provides a mechanism by which a central network management body can control every firewall within an organization.

The key requirements for remote management of a firewall are as follows:

- A secure channel between the firewall and the remote management client
- A consistent user interface for both local and remote management
- Support for multiple administrators with the ability to control the tasks each administrator can perform

Clearly, any firewall remote management capability must be completely secure, offering no possibility of compromise. If an attacker can break into a firewall's remote management channel, then the complete subversion of the firewall is likely.

For ease of management, it is also important that the user interface that is available remotely is the same as the one available locally, so that all firewall monitoring, control, and configuration facilities are available from the remote host.

Finally, the ability to support multiple administrators is required when a firewall is being managed remotely, so that the organization that operates the firewall can control who has access to the firewall and what operations they can perform. Typically, an organization restricts the administrators who can configure or reconfigure the firewall so that changes to the security policy the firewall implements are tightly controlled.

As described previously, the user interface for the AltaVista Firewall is implemented using an HTML-based approach, thus enabling any platform capable of supporting a Web browser to manage the firewall. The key requirement is to secure the connection between the firewall and the remote host the GUI traffic travels over, because all management actions are performed by the GUI Web server running on the firewall. The need to authenticate both the firewall and the remote host is critical to this. Although SSL-based solutions can easily deliver firewall authentication, it is less easy to deliver remote host authentication, without providing an appropriate key generation mechanism for potential clients within the firewall product. A solution is required that provides an encrypted channel for the GUI traffic and includes a mechanism that allows for authentication of both parties.

The AltaVista Tunnel product was selected to provide a secure channel for remote management for two reasons: (1) It delivers a mechanism by which traffic between the host running the AltaVista Firewall and a remote host is secured through encryption, and (2) It provides an easy-to-use mechanism for authentication of the two parties. For remote management, a tunnel is established between the remote host and the firewall. This tunnel provides the secure channel through which the firewall can be managed from a remote host. The firewall acts as a tunnel server (running the

AltaVista Workgroup Edition Tunnel software), while the remote host may be either a tunnel server or a tunnel client (running the AltaVista Personal Edition Tunnel software). In this way, an administrator can manage a firewall from any platform that supports the AltaVista Tunnel. A modified version of the AltaVista Workgroup Edition Tunnel software is shipped with the AltaVista Firewall, as well as AltaVista Personal Edition Tunnel software for Windows 95 and Windows NT platforms. The firewall tunnel software is modified to restrict the number of concurrent tunnels that can be started to one (for licensing reasons) and to operate the tunnel server on a nonstandard port. This avoids clashes with organizations that are relaying tunnels by means of the firewall.

A Web browser running on the firewall connects to the GUI Web server using the local host's address. In contrast, a Web browser running on the remote host connects to the GUI Web server using the IP address of the tunnel endpoint. The endpoint is the pseudo-IP address allocated to the tunnel on the firewall. The GUI Web server on the firewall is configured to allow only connections from the local host (the firewall) and from the tunnel endpoint (that is, the pseudo-IP address) of the remote host. The Web server automatically manages the GUI universal resource locators (URLs) generated for each user interface component, depending on whether the connection originates locally or through a remote management channel. Because the AltaVista Tunnel product adds a route on the host at one end of a tunnel to the pseudo-IP address of the tunnel endpoint on another host, traffic between the browser and GUI Web server is automatically routed through the tunnel and is secure from interception. The GUI Web server rejects any attempt to connect from a host that is not the local host or does not have a remote management channel configured for it.

The implementation of remote management support in the AltaVista Firewall includes GUI functionality that allows a firewall administrator to add, view, and delete remote management channels. When adding a new channel, the administrator specifies the pseudo-IP addresses for both tunnel endpoints, the name of the channel, and the type of tunnel to be used (Workgroup or Personal Edition). The GUI automatically configures the AltaVista Tunnel software, sets up the tunnel configuration required, and generates the necessary key and connection files for the remote host. The administrator is not required to perform any direct AltaVista Tunnel management activities on the firewall.

The implementation of multiple administrator support in the AltaVista Firewall includes GUI functionality that allows an existing administrator to add other administrators, change their GUI login passwords, specify what GUI tasks they can perform, and delete an administrator. Although the GUI restricts administrator logins from a particular source to one at a time, it is possible that separate administrators can log in locally and remotely. Administrators are granted privileges to monitor, control, and configure the firewall for each GUI subsystem. One administrator may be able only to monitor the firewall status, while another administrator may have the necessary privileges to configure the security policies for application gateways or to manage the user authentication system. In addition, GUI privileges are allocated separately for local and remote access, providing further flexibility for administrator privilege control.

## Future Enhancements

Large organizations that have local private networks in several geographies currently link these networks using expensive private connections. The growing availability of inexpensive, high-speed Internet connections and the development of secure IP tunneling software products, such as the AltaVista Tunnel, is prompting many of these organizations to construct virtual private networks (VPNs). Since each Internet connection must be secure, the next logical step is to integrate the IP tunneling capability into the AltaVista Firewall.

Larger organizations are moving away from the idea of having one firewall as a single choke-point connection to the Internet. Instead, multiple firewalls may be dispersed at several locations throughout a private network, perhaps on different continents. It will therefore be necessary to ensure that the network security of each firewall remains synchronized with the others. The ability to provide secure enterprise management of several firewalls from a single location is a major challenge for the AltaVista Firewall.

IP multicast technology[12] is now becoming a core component of the Internet and private corporate networks. Multicasting is the ability to distribute data packets to a group of one or more hosts, as opposed to unicasting, which refers to normal point-to-point Internet communications, and broadcasting, which refers to one-to-all communication. IP multicast has enormous potential, most notably in low-cost, real-time conferencing (for example, video and audio), where each host must send data to all other conference participants and other Internet multimedia applications. Multicast datagrams, however, may pose security vulnerabilities to machines that receive them. The AltaVista Firewall must address the need to relay multicast packets while continuing to ensure the security of the private network.

The deployment of IP networks based on the next-generation Internet Protocol Suite, IP version 6 (IPv6),[13] will address many of the structural and security issues that currently exist with IPv4. IPv6 will provide many advantages, including

- Scalability. Rapid growth in the Internet has resulted in the available IP address space being consumed at an alarming rate. IPv6 provides a much larger address space.

- Security. IPv6 addresses authentication, integrity, and confidentiality issues. Because IPv6 corrects many of the threats and vulnerabilities associated with IPv4, the architecture and security policy of an IPv6 firewall will significantly differ from those of an IPv4 firewall. The integration of IPv6 support into the AltaVista Firewall will require researching any outstanding security threats, as well as addressing the practical issues of performance slowed by cryptography.

## Summary

The emergence of new technologies and the growth of electronic commerce on the Internet means that network security will continue to increase in importance. The AltaVista Firewall addresses customer requirements for securing their Internet connections by providing a powerful and flexible firewall product that includes application-level and packet-level network traffic control functions, traffic logging, and security monitoring capabilities, together with comprehensive firewall configuration and management support through a GUI.

The AltaVista Firewall 97 for the DIGITAL UNIX operating system is now in its third release since its introduction in September 1995 and has achieved market recognition for its high performance, its comprehensive firewall features, and its ease of use. At the same time, the product provides systems integrators with a comprehensive set of firewall features and functionality to enable them to provide customized network security.

## Acknowledgments

We would like to acknowledge the following people for their contributions to the design and development of this product: Frank O'Dwyer, for his contributions to the access control system; Chris Evans and Eugene O'Connor for their extensive user interface and documentation work; Sarah Keating, whose testing contributed to the security and robustness of the product; Tim Shine, engineering manager, who ensured the timely delivery of the product; Brendan Noonan, product manager, whose market knowledge was invaluable; Jeff Mogul, for his advice on packet filtering; Dan Walsh and Ken Alden, for their advice and assistance with the AltaVista Tunnel product; Kevin Carey and Tony Pitt, for their help with beta testing; and the members of the AltaVista Internet Security Product Group.

## References

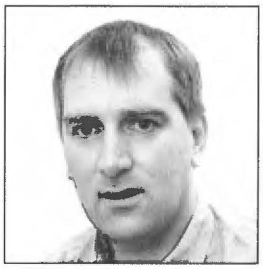
1. W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols* (Reading, Mass.: Addison Wesley, ISBN 0-201-63346-9, 1994).
2. S. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *Computer Communication Review,* vol. 19, no. 2 (1989): 32–48.
3. R. Morris, *A Weakness in the 4.2 BSD Unix TCP/IP Software* (Murray Hill, N.J.: AT&T Bell Laboratories, February 1985).
4. *CERT Coordination Center 1996 Annual Report* (http://www.cert.org/cert.report.96.html).
5. J. Mogul, R. Rashid, and M. Accetta, "The Packet Filter: An Efficient Mechanism for User Level Network Code," *Proceedings of the 11th Symposium on Operating Systems Principles,* AGM SYSOPS, Austin, Texas (November 1987).
6. J. Mogul, "Simple and Flexible Datagram Access Controls for UNIX-based Gateways" (Digital Equipment Corporation, Western Research Laboratory Research Report, April 1984).
7. J. Mogul, "Using screend to Implement IP/TCP Security Policies" (Digital Equipment Corporation, Network Systems Laboratory Technical Note TN-2, July 1991).
8. CERT Advisory CA-96.21, "TCP SYN Flooding and IP Spoofing Attacks," September 1996.
9. Y. Rekhter, B. Moskowitz, D. Karrenberg, G.T. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC 1918 (February 1996).
10. D. Newman, H. Holzbaur, and K. Bishop, "Lab Tests: Firewalls: Don't Get Burned," *Data Communications* (March 21, 1997) http:www.data.com/cgi-bin/dynamic/lab_tests/firewalls97-extras1.txt.
11. P. Albitz and C. Liu, *DNS and BIND,* 2nd Edition (Sebastopol, Calif.: O'Reilly & Associates, Inc., ISBN 1-56592-236-0, December 1996).
12. IP Multicast Initiative (http://www.ipmulticast.com).
13. S. Deering and R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 1883 (December 1995).

## General References

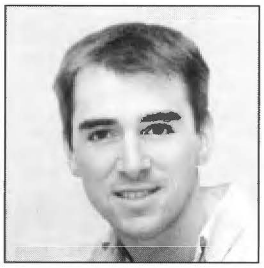
W. Cheswick and S. Bellovin, *Firewalls and Internet Security, Repelling the Wily Hacker* (Reading Mass.: Addison Wesley, ISBN 0-201-63357-4, 1994).

D. Chapman and E. Zwicky, *Building Internet Firewalls* (Sebastopol, Calif.: O'Reilly & Associates, Inc., ISBN 1-56592-124-0, 1995).

## Biographies



**J. Mark Smith**
Mark Smith joined DIGITAL in 1982 and is a principal engineer in the AltaVista Internet Security Product Group. In his current position, he has contributed to the design of the AltaVista Firewall product since its inception. In previous projects, Mark worked on the development of DIGITAL's Firewall Service and on the POLYCENTER Security Intrusion Detector product. Mark was also project leader for a number of publishing products, including DECwrite version 3.0, and was chair of the Technical Committee for the ODA Consortium. In addition, Mark has represented DIGITAL within several publishing-related standards groups. Mark received a B.E. in electronic engineering (1982) from University College Dublin, Ireland and an M.Sc. in computer systems design (1990) from the University of Dublin, Ireland. He has lectured in computing at University College Galway, Ireland and holds one patent.



**Sean G. Doherty**
Sean Doherty joined DIGITAL in 1993 and is a senior software engineer in the AltaVista Internet Security Product Group where he has designed and developed components of the AltaVista Firewall since its inception. Previously, Sean contributed to the design and development of the POLYCENTER Security Compliance Manager for NetWare product. Prior to joining DIGITAL, Sean was responsible for the development of system management software for Third Wave Ltd. Sean received a B.Sc. in computer applications from Dublin City University, Ireland (1992).

**Oliver J. Leahy**
A principal engineer in the AltaVista Security Products Group, Oliver Leahy has worked in security engineering for several years. He led the POLYCENTER Security Compliance Manager (PSCM) for NetWare project and worked on the development of PSCM for OpenVMS. Prior to this, he worked in the Publishing Technology Group and in the DIGITAL Semiconductor Acquisition Group, developing quality control and semiconductor test software. He holds a B.Sc. (1982) from Trinity College Dublin and an M.Eng. (1988) from the University of Limerick.

**Dermot M. Tynan**
A principal engineer in the AltaVista Internet Security Product Group, Dermot Tynan is the architect of the AltaVista Firewall product. Prior to joining DIGITAL, he worked as a UNIX kernel engineer for Altos Computer Systems, Unisys, and ICL. He designed software and hardware components for a real-time FFT system for Ultrasystems Space and Defense, Inc. and developed interprocess communication systems for British Telecoms' Work Management System. Dermot is an active member of the Internet Engineering Task Force (IETF), the ISO Motion Picture Experts Group (MPEG), and the Internet Society (ISOC). He is currently working on three Internet draft standards and is coinventor on a firewall patent.

Nick Shipman

# Developing Internet Software: AltaVista Mail

**The emergence of the Internet as a place where people can conduct business prompted DIGITAL to investigate the development of products specifically for use in this environment. Electronic messaging systems based on Internet technologies provide the communication medium for many businesses today. The development of AltaVista Mail illustrates many of the concerns facing engineers who are designing products for this new customer base. The results of our experience can be helpful in many ways and should be of interest to those involved in designing technologies for running Internet applications.**

In late 1993, the Mail Interchange Group (MIG) within DIGITAL started the AltaVista Mail development program. At that time, the members of MIG had substantial experience in the development of electronic mail (e-mail) technologies; however, the new products were being targeted for use on the Internet in an environment that was quite different from the one for their previous products. In an effort to satisfy the new customer base, the members of MIG reexamined their design and development process.

The AltaVista Mail product emerged from efforts to improve MIG's support for Internet-based e-mail technologies. Our previous products were electronic mail and directory servers for network backbone use, based on the most recent X.400 and X.500 standards. Products suitable for the Internet environment would clearly be quite different.

This paper begins by presenting our analysis of Internet services and support software and describing the transmission of e-mail on the Internet. The paper then discusses the implications of developing a product for the Internet environment and explains the impact of those implications on the design and implementation decisions that defined the AltaVista Mail product. The paper concludes with the engineering assumptions and habits that had to be overturned to build the product set.

## Internet Services and Software

During our initial analysis of the product possibilities, we made several interesting observations about Internet services and software, particularly in comparison to the mission-critical products in MIG's existing portfolio. (Our observations could more accurately be called assertions—it was and is remarkably difficult to get hard information about Internet use.)

1. The academic/research/technical community determined the nature of the Internet's service offerings. Most of the software defining the Internet's services was generated by and for this community.

2. The real market opportunity would not be among the academic/research/technical community but would be drawn from ordinary businesses.
3. Much of the service software on the Internet was unsatisfactory for routine business use, either because it was unreliable or because it was difficult for end users to deal with it. Even though free software was abundant, much of it did not work. Support for the free software was a risk: some software had excellent peer support; unfortunately, not all end users were aware of its existence or were able to access it.
4. We judged the operating system platforms commonly used for service software to be unsuitable for a large part of the business community. The various UNIX platforms need skilled local staff; corrupted or poorly configured Windows version 3.1 and Macintosh run-time environments would be difficult to diagnose and expensive to support.

Expanding into support of the Internet environment would require us to build native equivalents for some of our existing server software. We believed that the Windows NT platform offered a good framework for systems that would work well in any business environment and be easy to support.

Initially, we required the following products: a server that supported the Simple Mail Transfer Protocol (SMTP) and the Post Office Protocol version 3 (POP3); a gateway to Lotus cc:Mail post offices; and a gateway to Microsoft Mail post offices.

The SMTP/POP3 server is the mail system component responsible for accepting messages from mail client programs. It transmits them toward the recipients' SMTP servers and performs local delivery using the POP3 protocol. This process is described in more detail in the next section.

## E-mail on the Internet

This section briefly describes how Internet e-mail is transferred from the originator to the recipient.

The originator's mail client program constructs a message according to the rules described in the Internet standard, RFC 822.[1] The RFC 822 standard defines a message as a sequence of short lines of 7-bit ASCII text, each terminated by a carriage-return and line-feed sequence (CRLF). The first lines are header fields; these are extensible but typically include the originator and recipient e-mail addresses, the date, and the message subject. The header ends with a blank line, and the remaining lines constitute the body of the message. Figure 1 shows an SMTP dialogue that includes an RFC 822 message.

Where appropriate, the mail program can also follow the Multipurpose Internet Mail Extensions (MIME) rules in RFC 1521 and RFC 1522[2,3]; these describe

```
SMTP command/response                            Comments
                                                 Caller opens connection
220 server1.altavista.co.uk AltaVista Mail       Server's welcome message
  V1.0/1.0  BL22 SMTP ready
helo client1.altavista.co.uk                     Client gives its own host name
250 OK                                           Host name was acceptable
mail from:<Fred@altavista.co.uk>                 Identifies return path for nondelivery reports
250 OK                                           Return path was acceptable
rcpt to:<Bill@altavista.co.uk>                   A recipient for this message
250 OK                                           Recipient was acceptable
data                                             Message follows
354 Start mail input; end with <CRLF>.<CRLF>     OK to start message header
Date: Mon, 7 Jul 1997 08:30:13 +0100             Message's date
From: Fred <Fred@altavista.co.uk>                Originator field
To: Bill <Bill@altavista.co.uk>                  Recipient field
Subject: Example message                         Subject field
                                                 Blank line ends message-header fields
Hi Bill,                                         Content lines...

This is a test message.
It's not very long.

Fred
.                                                ...End of content
250 OK                                           Message has been accepted
quit                                             No more messages; signing off
221 redsvr.altavista.co.uk closing connection    Finished
```

**Figure 1**
Example SMTP Dialogue

how to construct a message body to transfer typed and structured data and how to pass non-ASCII characters in header fields. Figure 2 shows an example of a message constructed according to the MIME standard.

The originator's client submits the message to a nearby SMTP server using the SMTP protocol.[4] This very simple protocol uses short, CRLF-terminated lines of 7-bit ASCII text to transfer its commands and responses. To submit a message, three commands are used: the MAIL, RCPT, and DATA commands introduce the originator's e-mail address, the recipients' e-mail addresses, and the RFC 822 message data, respectively.

The SMTP server examines each recipient's e-mail address to decide where the message should be sent. Recipients are routed by consulting the Domain Name System (DNS), a distributed directory that associates domain names with sets of typed resource records that denote the published properties of each domain.[5-7] For a recipient, user@domain.name, the target domain.name is looked up and the resource records of type MX (for Mail eXchange) are retrieved.[5] These records list the hosts that the domain nominates to receive its mail; each host has a numeric preference value. Eventually, the mail must be delivered to the most preferred host.

For each target domain, the SMTP server uses the SMTP protocol to transfer the message to the domain's most preferred, reachable MX host. (The most preferred host may be unreachable from the local server: it may be switched off for a while, or it may be behind a firewall, a machine that protects a private network by limiting access from the open Internet to the machines inside the protected network.) The chosen host, if it is not the most preferred, forwards the message to a more preferred host and so on, until the message reaches the recipient domain's most preferred host. That host then delivers the message to an area from which the recipient's mail client program can fetch it.

Fetching a message is often a platform-specific operation, but a standard protocol such as POP3 can also be used.[8] This simple, text-based protocol allows the mail client to list the messages waiting to be fetched, to fetch individual messages, and to delete them from the server once they are safely stored within the client.

Newer, more feature-rich protocols and interfaces, such as the Internet Message Access Protocol version

```
MIME data                                       Comments

Date: Mon, 7 Jul 1997 08:30:13 +0100            Normal RFC 822 header fields
From: Fred <Fred@altavista.co.uk>
To: Bill <Bill@altavista.co.uk>
Subject: Binary attachment
MIME-version: 1.0                               This is a MIME message...
Content-type: multipart/mixed;
  boundary="zzzBoundaryzzz"                     ... consisting of a list of body parts
                                                Blank line ends message-header fields
                                                Start...
--zzzBoundaryzzz                                ... of first body part...
Content-type: text/plain; charset="us-ascii"    ... in ASCII plain text...
Content-Transfer-Encoding: 7bit                 ... using 7-bit encoding
                                                Blank line ends body-part-header fields
Hi Bill,                                        Body-part contents

Here's a binary file.
It's four bytes of all 1's.

Fred
                                                End...
--zzzBoundaryzzz                                ... of first body part and start of second...
Content-type: application/octet-stream;         ... a stream of bytes called foo.dat...
  name="foo.dat"
Content-Transfer-Encoding: base64               ... using base64 encoding
                                                Blank line ends body-part-header fields
/////w==                                        Hex FFFFFFFF encoded in base64
                                                End...
--zzzBoundaryzzz--                              ... of message
```

**Figure 2**
Example MIME Message

4 (IMAP4), do offer certain user advantages; however, they do not perform the basic job of delivering messages any better than POP3.[9] Even though support for IMAP4 was added to AltaVista Mail version 2.0, POP3 remains the method of choice for fetching messages from a remote server: this protocol is so simple that it is hard to implement incorrectly.

## Product Design Decisions

The definition of the AltaVista Mail product set did not start with technical issues. Instead, it started with an assumption about the purchase price of a product. Even though the price we chose was not used for the released product, our assumption turned out to be, perhaps, the most useful and powerful design tool available during development.

### *Product Pricing and Organizational Concerns*

We were interested in exploring the implications of offering a product at a very low price and selling in very large quantities to make the business worthwhile. MIG's previous products had been priced at the opposite end of the price scale: they were expensive, but they were valuable to the relatively few customers who needed their functions.

(Interestingly, as we explored the necessary organizational and technical changes involved in moving to the low end, we realized that they would in no way jeopardize our ability to sell at the high end. The organizational changes improve efficiency no matter what the product price, and the technical changes make for a better, more usable product regardless of the customer profile.)

Our starting point was to investigate the engineering implications of building a product that would sell for $100. We concluded the following:

- To maximize the number of products sold, we would have to satisfy the largest imaginable customer base and not exclude a potential customer for any reason.
- Customers attracted to a low purchase price will also require low running costs. No hidden costs could be associated with running the product.
- Support costs would have to be kept to a minimum. If each customer needed telephone support several times over the life of the product, the $100 price would not cover support expenses. We would have to aim at receiving zero support calls.
- We would have to minimize the implementation and maintenance cost and deliver products and updates as early as possible. The Internet market moves quickly, particularly at the low end, and a long development cycle loses sales.

Our existing approach to development involved obtaining agreement from many groups within DIGITAL concerning the nature of a problem area and the architecture of any solutions, and then implementing product versions against the architecture. This process is slow and expensive with considerable management overhead.

For the AltaVista Mail product, we decided instead to direct a small team to generate a product-quality prototype as quickly as possible and to ship that prototype as a product. In the interests of rapid development, we would deliberately discard much of the traditional Phase Review Process but would use regular, informal monitoring to ensure that the prototype remained acceptable to our target customer.

All design and implementation decisions would be judged by their effect on this target customer, not by their adherence to an architecture. All future development would be guided by customer feedback. This method is far less expensive, delivers a product far sooner, and is more likely to reflect current customer needs.

### *Technological Concerns*

Our ideas on product pricing and the design process led to three initial design decisions.

First, nonexpert users must be able to get the full value from the product. Setting up and configuring the product must involve answering the minimum number of questions. Each question must relate to a topic on which the user can reasonably be expected to have an opinion. The user must not be asked questions about the internal operation of the product, only about topics with an external significance.

The product must offer the minimum number of operational controls. (Some high-end customers demand many controls. If necessary, these controls could be added in a later version; but the product must not depend on them, they should not be presented to the average user, and those users who insist on seeing them should be charged a premium to cover the additional support costs. We would explicitly accept that there are certain customers we should not aim to satisfy and certain features we should never offer.)

Second, the product must never go wrong. The product must never encounter any internal errors, only those caused by failures in its operational environment. Any environmental failure must be reported completely and accurately in terms that the user can understand. After a failure has been fixed, the product must start working again with no further intervention. If an environmental failure or an operator intervention corrupted the software, reinstalling the kit must get the system working again. The product must not depend on any product that does not follow these rules.

Third, the product must be inexpensive to build and maintain and must use a rapid development cycle. The product—and each of its components—should deliver the maximum customer-perceived value for the minimum engineering investment. Although the number of features should be minimized, the functions delivered should be sufficient to be useful to a large customer base.

## Implementation Decisions

The most important decision was to aim for simplicity above all else: simplicity of design, implementation, and presentation. Simplicity delivers reliability and inexpensive implementation and maintenance. It also helps to ensure that a product is comprehensible to its end user and does not behave in baffling ways, even when it is not working due to an external influence.

A related decision was to use no method or tool that might encourage complexity by helping to manage it: no formal design methods, no automated design verification, and no automated system test. The developer should immediately feel the pain of building a complex structure or one that requires an elaborate system test and thus be encouraged to think again.

Of course, proper engineering practice dictated that we should use a repeatable system test with properties that were well understood, but we deliberately never automated the test. We also used remedial tools such as locally developed libraries to look for memory and handle leaks. (These tools do not tempt developers into bad habits.)

### *User Interaction with the Server*

Ideally, the server should perform its job with no comment, and the user should feel no need to think about the server's performance. Product documentation should be minimized, and there should be no printed documentation at all.

We designed the server to make many of the product's operational decisions, rather than leave the decisions to an administrator:

- The administrator cannot control the routing process. Messages are sent to the targets defined in DNS, and no local rules nominating other targets are supported. The administrator can nominate a firewall but cannot say when to use it (the server uses it automatically, when all the other targets have been found unreachable).
- The operational logs are purged automatically. The administrator can only control how long any logged event is guaranteed to be stored before being purged, and this interval cannot be selected on a per-log or per-event basis.
- The server's network connections are scheduled by the server itself. The administrator can only control the minimum and maximum retry intervals for SMTP connection attempts, not the specific times at which the server tries to communicate.
- The server determines if an event is relevant to system security and responds according to its own rules. Repeated authentication failures result in mailboxes and originating host addresses being locked out for a time; the administrator can manually reset the lockout but cannot control how long it lasts, nor how many failures are judged to be an attack.

Unfortunately, user interfaces cannot be avoided entirely. Therefore the goal must be to minimize the amount of user interaction required with the server and to make user interaction easy to perform and as reassuring as possible to the user. We were able to design the SMTP/POP3 server to be easy to set up and use, and we reduced the user interaction with the gateways to the minimum. Gateways are notoriously difficult to set up; we simplified the process of installation to the extent that a central cc:Mail or MSmail administrator can talk an inexperienced user through the installation.

The SMTP/POP3 server requires an administrator to perform only the following four actions:

- Load the software to a chosen volume
- Tell the server about the local network configuration
- Set up mail accounts (mailboxes) for the local users
- Check that the server is not experiencing problems

The AltaVista Mail product implements these tasks through three user interfaces: the setup (installation) procedure, a Windows-based administration graphical user interface (GUI), and World Wide Web forms.

**Setup Procedure.** Apart from loading the software, the setup procedure has several other functions. If the software has been corrupted, the setup procedure repairs the service by resetting the server's entire environment to a known state. The server's account privileges are reset, a new password is generated, and the database directories and files have their file protection reset.

Setup continues by asking the minimum number of questions required to allow the server to work. The administrator responds by naming any firewall used, or if in dial-up mode, naming the remote mail server. Finally, it runs the server's self-test, which is described in more detail later in this paper. It is important that self-test run during setup. If the system is not working, the administrator needs to be told the precise causes of the problem immediately, before he or she can become confused by subsequent symptoms of system failure.

**Windows-based Administration GUI.** The Windows-based administration GUI controls the AltaVista Mail server through a simple, text-based administration protocol over a Transmission Control Protocol/Internet Protocol (TCP/IP) link. Therefore the GUI can control servers elsewhere in the network.

The GUI has two modes. The default mode is a simple menu with links to functions that invoke the most common administrative tasks: network configuration; adding users and mailing lists; redirecting mail and changing passwords; and self-test. This mode, shown in Figure 3, is a nonthreatening interface for inexperienced administrators; they do not need any other information to use the server successfully. This means that local users with no specialized knowledge can set up small and undemanding sites.

Administrators of large or busy sites need to use the advanced mode, shown in Figure 4. This interface is similar to the style of the Windows Explorer GUI and gives access to every server and mailbox control, the messages in the server, and its operational logs.

Both modes include an on-line help file that gives a brief introduction to the system, reference information on all the visible controls, and assistance in troubleshooting. Apart from the equivalent help text in the World Wide Web forms, this is the only product documentation.

The self-test is one of the most important administration functions for sites at which mail is a mission-critical service. The self-test checks that all aspects of the local machine's environment that are necessary for the server to operate do indeed work; it also checks that the server is responding correctly on all the network ports it serves. In a redundant environment, the self-test checks every element to make sure partial failures are reported. For example, a host generally knows of two or more DNS servers, only one of which needs to be working for the mail server to run. Because the mail server will not see a problem until the last DNS server dies, the self-test must report any partial failure.

The self-test is vital: a regular check is the only way to be sure that a background server is working. A server cannot be guaranteed to inform an administrator of

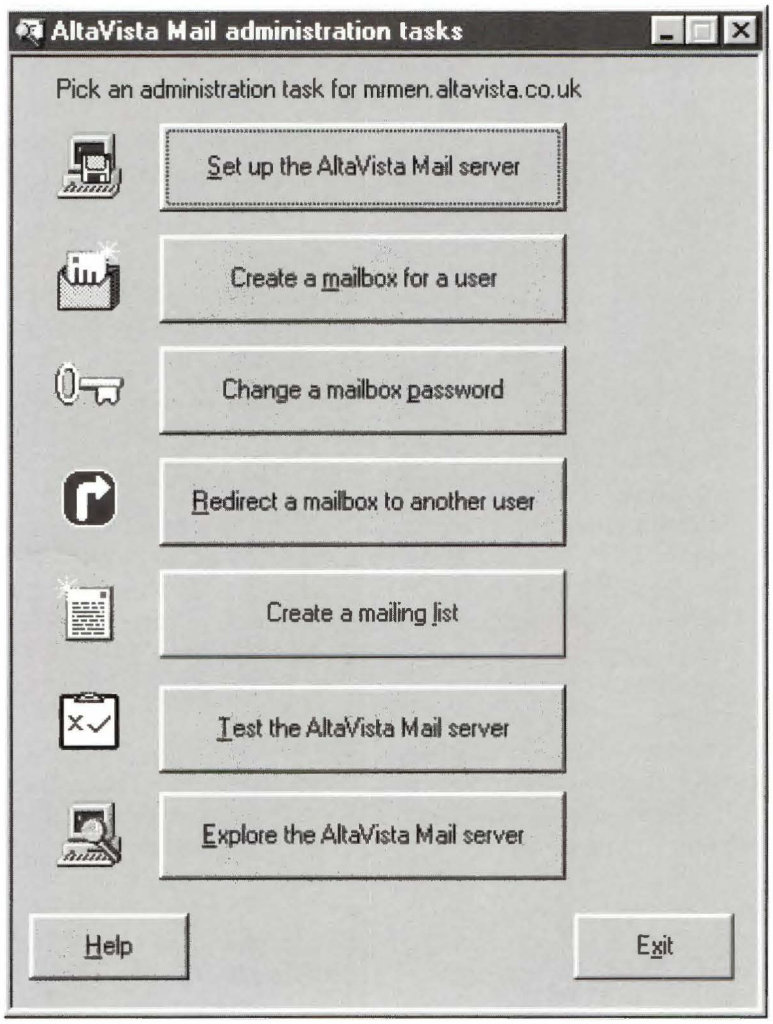
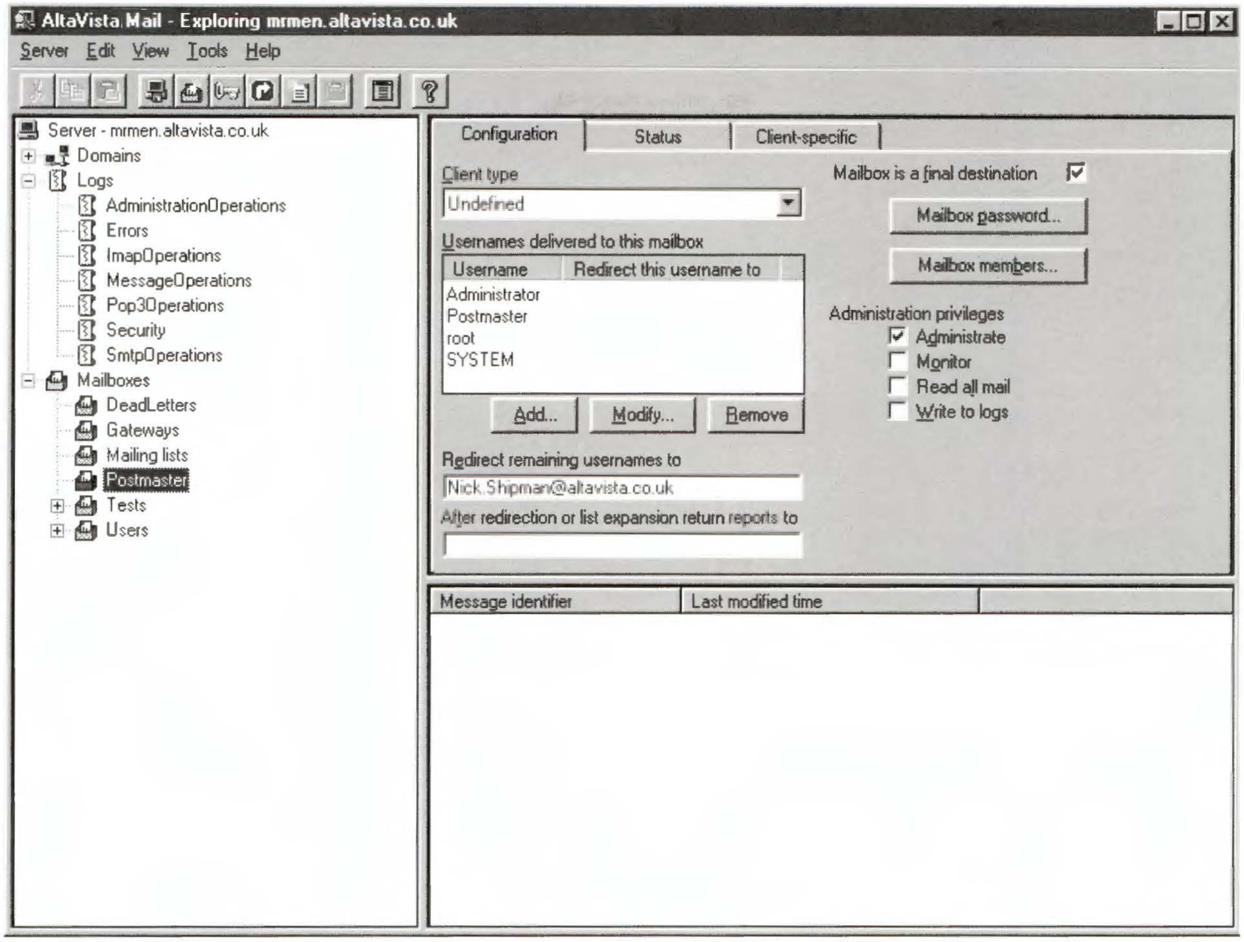

AltaVista Mail administration tasks
Pick an administration task for mrmen.altavista.co.uk
Set up the AltaVista Mail server
Create a mailbox for a user
Change a mailbox password
Redirect a mailbox to another user
Create a mailing list
Test the AltaVista Mail server
Explore the AltaVista Mail server
Help
Exit

**Figure 3**
Windows-based Administration GUI, Simple Menu



Client type: Undefined

Mailbox is a final destination [✓]

Usernames delivered to this mailbox

| Username | Redirect this username to |
|---|---|
| Administrator | |
| Postmaster | |
| root | |
| SYSTEM | |

Administration privileges
[✓] Administrate
[ ] Monitor
[ ] Read all mail
[ ] Write to logs

Redirect remaining usernames to: Nick.Shipman@altavista.co.uk

After redirection or list expansion return reports to:

**Figure 4**
Windows-based Administration GUI, Advanced Mode

problems because the problems may affect the notification path used. The AltaVista Mail self-test allows an administrator to perform the necessary check with a single "button click." The self-test also runs as part of a regular cleanup procedure. Errors are reported to the server's error log, so a less active administrator can monitor the system by reading this log every few days.

**World Wide Web Forms.** The World Wide Web (WWW) forms interface is provided by a built-in Hypertext Transfer Protocol (HTTP) server. This interface offers the same facilities as the Windows administration GUI. Because a Windows Explorer-style GUI is more difficult to present using a Web browser, we implemented the "power user" options on the top-level task menu of the WWW form. These options make the initial interface somewhat intimidating, because they include controls whose function may not be understood by an inexperienced administrator. The familiar controls, however, are grouped together. The Web GUI is shown in Figure 5.

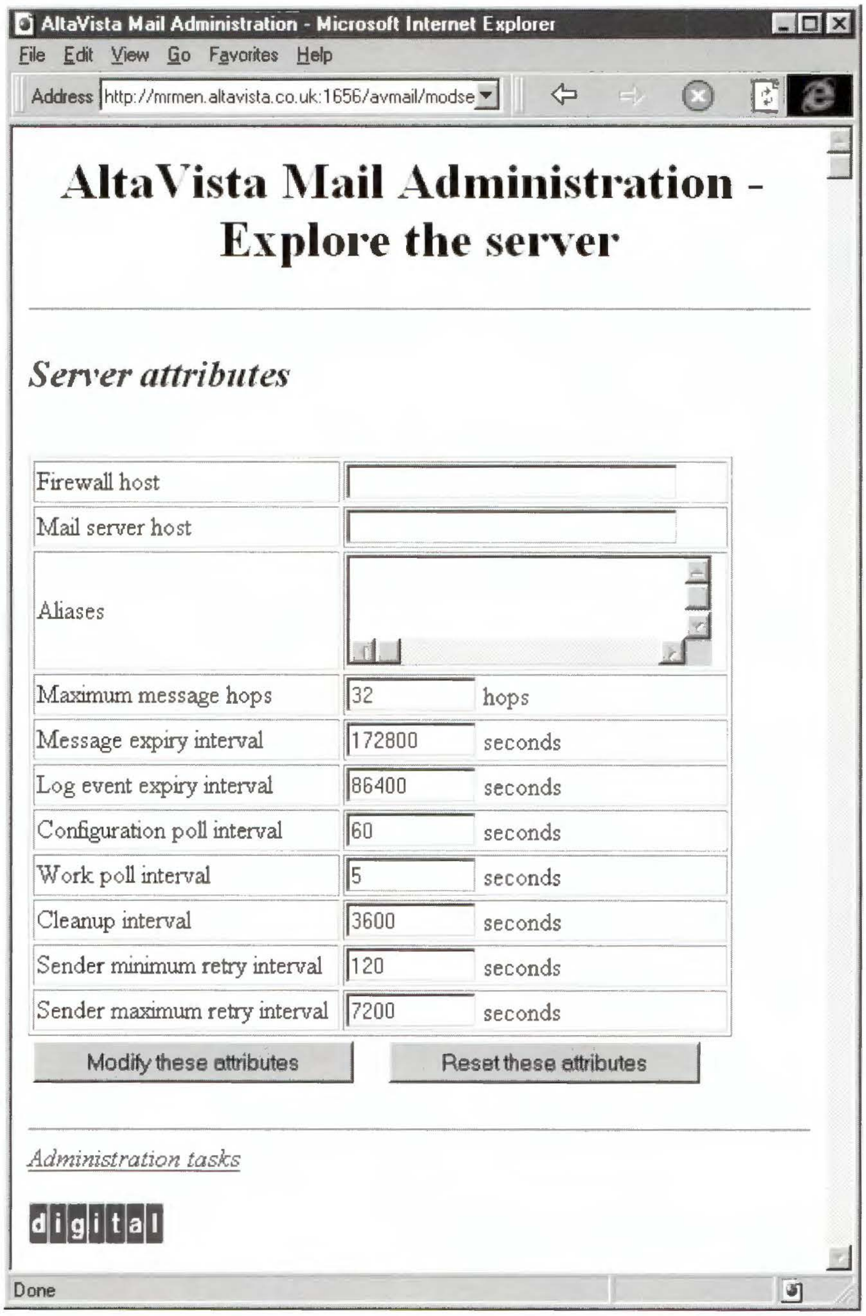
Several details help make the three user interfaces approachable and nonthreatening.

When the software requires the user to answer a series of questions, it presents a dialogue box chain (sometimes known as a "wizard"). Used properly, this technique allows the user to concentrate on one thing at a time, with all distracting material hidden.

Every question in a dialogue box chain gives an explanation of what information is needed, any suitable defaults or examples, a suggestion of whom to contact to find the answer, and a safe way to abort the process. If the user knows the answer, he or she will be able to recognize it in the example. Users who do not know the answer will not be intimidated by the wording.

The logic works in terms the user understands, not in terms of the software's operation. The gateways, for example, contain a question that the software does not use other than to make the text of the succeeding dialogues relate to the user's environment.

Some controls can easily be invoked in error but cannot be redefined to make the error less likely. In


AltaVista Mail Administration - Microsoft Internet Explorer

File Edit View Go Favorites Help

Address http://mrmen.altavista.co.uk:1656/avmail/modse

# AltaVista Mail Administration - Explore the server

*Server attributes*

| | | |
|---|---|---|
| Firewall host | | |
| Mail server host | | |
| Aliases | | |
| Maximum message hops | 32 | hops |
| Message expiry interval | 172800 | seconds |
| Log event expiry interval | 86400 | seconds |
| Configuration poll interval | 60 | seconds |
| Work poll interval | 5 | seconds |
| Cleanup interval | 3600 | seconds |
| Sender minimum retry interval | 120 | seconds |
| Sender maximum retry interval | 7200 | seconds |

Modify these attributes Reset these attributes

*Administration tasks*

digital

Done

**Figure 5**
WWW Forms Administration GUI

these cases, the resulting dialogue box confirms the control's function and offers the opportunity to try again. For example, it is easy to hit the add-username-to-mailbox control instead of the add-mailbox control, and this confusion cannot easily be eliminated with a revised definition. The add-username dialogue therefore warns that it does not add a mailbox but offers a route to the dialogue that does.

The mechanical operation of all controls is smooth. Appropriate default functions are always active; for example, when an input field is empty, the default function might be "Next" or "Skip"; but the moment any text is entered, the default function changes to "Add" (or whatever normally happens to the input text). This helps the user ignore the interface and concentrate on the meaning.

### Performance

In a mail server product, performance, measured as the number of messages processed per unit time, is usually a major concern. In previous products designed by MIG, performance was among the handful of top-priority goals, and from these we had learned a great deal about designing for the highest possible performance. We had also learned that the single-minded pursuit of performance is expensive, disruptive to implementation, and prone to error.

This experience yielded a set of informal rules for cost-effective design regarding application performance. These rules proved to be effective during the development of the AltaVista Mail product set. (Remember, these rules relate to the performance of typical applications: they would not apply to writing an operating system or other low-level code.)

1. Estimate the minimum disk I/O that the product operations will require, but treat this value only as a sanity check against gross waste of resources. Do not insist that this minimum be achieved.
2. Avoid checking special cases in a task's input data to avoid processing steps. Such optimizations are very likely to cause maintenance errors to go unnoticed and greatly increase the cost of the system test.
3. Never optimize tasks that consume only CPU time; examine only those algorithms suspected of being high-order consumers. It is almost never worth optimizing error cases.
4. Do not use complicated buffer management schemes to avoid copying data. Complicated code is prone to maintenance errors, and performance will not be helped much. Modern computers are very good at copying buffers of data but relatively slow at executing the complex branch logic that might be required to avoid copying data.
5. Take advantage of the high-performance system routines in modern operating systems. Do not build a memory allocator or a disk cache: the operating system developer has already spent far more on performance than an application developer can afford to spend. (Even if the operating system routines do not perform well, the problem will be common to all applications that use the platform.) Spend time solving the customer's problem, not repeating operating system development.
6. Use the operating system disk cache. It can be worthwhile to read even quite large amounts of file data in multiple passes if that will simplify program logic: the data will normally stay in memory for a subsequent pass. If the data has been flushed from the cache for the next pass, then the system had a better use for the memory, and any attempt to avoid multiple passes by remembering substantial state will degrade performance, not improve it.
7. Never offer an asynchronous application programmer interface (API), and avoid using asynchronous modes of otherwise synchronous services, even if that technique is presented as the way to obtain good performance. When more than a tiny region of code is directly affected by asynchronous events, it becomes difficult to be convinced that the code works. In addition, the code is unlikely to keep working as it is maintained. Synchronous methods—multithreading with thread-synchronous calls, polling, and timeouts—will normally yield perfectly adequate performance; they need only be avoided for very low-level code such as GUI window procedures.
8. Where latency requirements permit, polling for new work or configuration changes can be a better solution than an active notification path. Polling is easy to implement and robust. There is no need to ensure that an idle program is completely inactive: since modern operating systems page rather than swap, minor CPU attention every few seconds imposes no noticeable performance penalty.

Our approach to performance—and its nearly complete subordination to simplicity—can be seen in several aspects of AltaVista Mail's operation.

The server's function is to switch messages, so its database is a set of messages organized by target. Each message is held as a single text file; the text consists of the SMTP commands that would introduce each message to the server.

A message file is held in a directory that denotes its target, whether that target is a remote domain, a local mailbox, or a thread of the SMTP server itself. When the router decides on the target, it copies the message to its target directory and deletes the original. When a message splits to multiple targets, no attempt is made to share the common message data.

Although we could have designed a far more efficient way of representing the message database and splitting the message data as it flows through the server, our experience with previous products suggested that it was not necessary. Because the storage system we chose was a simple one, we could afford to throw it away if it did not work under load. Happily, it did work, and we gained a highly robust storage system with excellent performance for a trivial investment.

When a message is passed from one thread of the SMTP server to another, it is left in the target thread's input directory. There is no notification path; the target thread discovers the message by polling.

To make the source code as comprehensible as possible, the server uses extremely simple protocol parsers. The source code is organized in terms of a programmer's understanding of the protocol, not some abstraction that might be more efficient. The parsers scan the input data as many times as is convenient to extract the data they need at each level. The result is secure and reliable protocol machines that are easy to verify and modify when necessary.

Despite the apparent lack of care for conventional performance concerns, extreme workloads must be supported. All the server's components must support huge numbers of messages, messages of huge size, messages with huge recipient lists, and messages bound for huge numbers of targets. Ideally, the system should impose no limits below those imposed by the underlying machines. An extreme workload should cause no problems or substantially reduce the work rate.

The POP3 server has to be able to support tens of thousands of messages in a mailbox. On connecting, most POP3 clients ask for a mailbox listing, which involves counting the size of each message in the mailbox. If thousands of messages are present, this can take so long that the client disconnects, believing the server has failed. Subsequent reconnections see exactly the same problem, and no mail flows. To avoid this problem, the server returns a partial listing to the client if it believes the full listing is taking too long. When the full listing is finished, it is written to disk to be returned to the client the next time it connects.

SMTP rules state that servers must support at least 100 recipients per message, and that ideally there should be no limit. Our existing customer base expects no limit. Arbitrarily large recipient lists can be allocated in virtual memory simply by configuring a page file of sufficient size. However, very large lists then impose a severe and highly variable load on the system. To keep the system load within reasonable bounds, we placed an upper bound on the amount of virtual memory claimed. The server limits the number of splits a single message can make while being routed. (A block of virtual memory is required for each split, not for each recipient; however, in the worst case, each recipient requires its own split.) Once a message's recipient list has split to more than 64 targets, subsequent recipients are moved to a new copy of the message that will be routed separately, regardless of whether the recipients could have been served by an existing split.

### Error Handling and Error Reporting

The server has to work reliably, even in the face of errors in the local environment. Experience suggests that code containing many error paths does not work. If a function can fail for any of several individually identified reasons, and the calling code has to handle each reason separately, sooner or later some of those error paths will be incorrect. Checking that each path continues to work as development and maintenance continues makes the system test very expensive; furthermore, it is difficult to cause every possible error to arise when testing.

To ensure its reliability, the server uses the following implementation rule: any function is allowed to fail, but its calling code is not allowed to distinguish between the various reasons for failure. Indeed, it cannot make a distinction, because only one failure return code is defined. Functions were reworked as necessary, so the rule could be observed.

(A related observation is that the code will be more reliable when only one kind of success outcome is allowed. We found this to be true to some extent: it is equivalent to saying that a linear, nonconditional flow of control is more reliable than a highly conditional one. However, as long as every success outcome does occur during the product's normal operation, the code that handles it will probably continue to work, and the system test will have reason to check that it does.)

The server also has to report any errors it encounters: it must say which server operation could not be performed, precisely what system condition caused it to fail, and what to do about the problem. This apparently conflicting requirement is handled with a second rule: error handling must be completely separated from error reporting.

Error reporting works with the structure of the server. The server's flow of control and breakdown into threads were designed specifically to support precise error reporting. Each thread has a well-defined purpose that can, if necessary, be explained to the administrator, or at least named in a diagnostic report without causing confusion. Because a thread is fully in charge of its task, it is able to report the significance of any failures it encounters to the administrator.

A routine uses a simple stack-based error-posting module to report an encountered error. The report includes a description of the failed operation, the operation's parameters, and other helpful information such as the name and return status of any failed system call. The routine then returns the standard error status code. Its caller sees that this routine has failed and generates a report, logging the failure and adding any relevant parameters it holds. The caller then releases any resources associated with the failed request and returns the standard error status code. Eventually a high-level routine handles the failure, typically by logging the stack of error reports and continuing with its next item of work.

Our approach to error handling and reporting yielded extremely good results, but it had two serious implications. First, we could not import any source code from existing systems: all the code in the server had to use the error handling and reporting methods just described. Often, we could not use an established API definition for common functions. Second, the product and each of its components needs built-in knowledge of its function as perceived by the user, so it can report the true status of any problem and ideally give suggestions for fixing the problem. It needs to report the implications of the problem, not merely the facts of the problem. For these reasons, we could not use the powerful UNIX-style approach of building complex systems out of small, general-purpose tools.

### *Additional Necessary Features*

In general, we tried to avoid adding features and keep in mind the server's one basic function: to move messages from place to place with minimal user intervention. When forced to add a feature, we aimed to keep it as simple and inexpensive to implement as possible, yet ensure that the feature offered the greatest real value. Obviously, this involved a trade-off, but it was usually clear how far to go.

The server needed a log subsystem to report important events, for example, errors encountered and suspected security violations (attempts to break into the server). To gain the maximum benefit from the log subsystem, we also used it to report the normal activities of the server in sufficient depth to perform a complete analysis of its work. This allows the logs to be used for load monitoring, performance analysis, message tracing, and billing and accounting. Enough information is logged to identify exactly what has happened to each message submitted to the server. Header information allows the originator and the recipients to be identified, and checksums for envelope and content allow duplicate messages to be detected. Duplicate detection is useful as a diagnostic aid and to avoid billing multiple times for a single message.

The POP3 protocol does not report a message's actual recipients (as opposed to the To: and Cc: fields, which may not be complete or even related to the real recipients). It therefore cannot be used as a way of delivering messages to gateways: it is only suitable for final delivery to recipients. For this reason, the addition of a proprietary interface could not be avoided. We chose to implement an API because it offers the simplest possible interface to the message data: sequential access to the return path, the recipients, the header fields, and the lines of content data. At the same time, it provides routines that encapsulate much of the complicated and error-prone logic that gateways often need.

For example, the API allows a gateway that is fetching a message to handle each recipient individually: it can accept, nondeliver, redirect, expand, or send for retry each of the recipients it sees. The gateway automatically generates any required new messages, including nondelivery reports, messages containing those recipients sent for retry, and messages with new recipients added by redirection or mailing list expansion.

The use of an API also guarantees that a gateway's operation is fully logged. When message-IDs and originator and recipient addresses are translated between SMTP and the foreign representation, the correspondence is logged. Messages can then be traced, even across gateway boundaries.

In addition to these features, we extended the services of the built-in HTTP server, which offers the administration Web pages. With a combination of server-parsed hypertext mark-up language (HTML) and a low-level attribute handling system to read and write server data, a customer can change both the appearance and the function of the Web pages, simply by editing HTML files.

## Experience with the Product

The AltaVista Mail product set has achieved its design goals and has validated the implementation rules we imposed on it. It has been inexpensive to develop, support, and maintain; is a well-behaved and effective solution; offers excellent performance; and has yielded very few bugs.

Its major deficiency is that AltaVista Mail, by itself, does not form a complete solution. Despite our efforts to ensure that it can be run by inexperienced administrators, it relies on complex external technologies. Dial-up networking, Internet Protocol address assignment, and the other aspects of the interface to the Internet service provider present problems that are not easy to deal with.

Our major problem is the configuration of MX (routing information) records in DNS. Although the product reports misconfiguration accurately, users call us to find out what to do about it. Better integration with DNS would substantially reduce our support load.

Overall, we tried to keep the number of controls to the minimum. In retrospect, we did build in a few that perhaps should not have been provided. For example, the administrator has complete control over the SMTP timeouts. Although this is required by the relevant RFC documents, we should have had the confidence to pick values that worked everywhere rather than provide the control.

On the other hand, providing more control in certain areas would have expanded the range of customers who could use the product. For example, some low-end customers need to control the schedule on which SMTP connections and DNS requests are made, and the dial-up facilities we provide are too crude to do this.

## Conclusions

This section reviews the organizational, design, and implementation rules we found most helpful in building the initial AltaVista Mail products. These ideas are the ones we recommend to engineers who are starting a new area of work.

### *Running a Development Project*

If possible, develop a new application as a product-quality prototype, not as the implementation of an architecture. A prototype can be brought to market quickly and inexpensively and will generate helpful feedback from customers. It is much more difficult to make sure an abstract architecture is not expensively addressing problems that do not need to be solved.

Do not develop prototypes that are not product-quality. A body of unshippable code that has been built as a proof of concept does not demonstrate the practicality of building a product. Take shortcuts, but not in any area that affects the application's fitness for use or maintainability. If the outcome is a shippable prototype, the choice can then be made whether to ship it or not.

Accept requirements input from anyone who will express an opinion but grant veto power only to those who are funding the project. Do not let a search for consensus slow the application's entry into the market but be very clear about why the application is the right thing, keep track of the risks, and be ready to respond to changes in the market. Do not use the beta test (open field test) merely to check if the product works; use it to decide whether the product needs changes. A change of mind at the last minute is not a failure.

Underfund software development. Allow enough time to produce the core of the solution and no more. This helps developers in two ways: they concentrate on what really matters, so they keep looking for the most effective ways to solve the largest possible parts of the problem, and they do not start "knitting." Developers enjoy developing; given adequate time, they will increase the functionality of the product. This will reduce the product's quality, not increase it. Given extra resources, it is unlikely that those funding development would choose the extra functionality a developer would like to build.

### *Defining a Product*

Technical issues are never the most important considerations in defining a product. The most important thing to analyze is the customer profile. Instead of building a product, the goal should be solving problems for the customer. The more accurately the customer's problems can be characterized, the more effectively the product will solve them.

At the beginning, assume that the product will be sold at a very low price. Think through the implications; they will indicate the nature of an acceptable product. Then, if the price is higher, add the extra features that the increased price will require.

Assume that the product will be supported directly from the engineering group. Imagine what would be necessary to support the product and define requirements to impose on the product that will minimize the cost of providing support. Then, if the product is supported elsewhere, add the extra features that the external support structure will require.

### *Design and Implementation*

Only a limited number of clever and difficult components can be designed and built into the product. Make sure they are the ones that will make the most difference, and make sure each difficult part is as small as it can be by encapsulating it in a simple interface. Outside the most critical areas, use the simplest designs possible. Do not simplify to spend less time on design: simplify to improve the quality of the product and to reduce the cost of implementation and maintenance.

Early in the project, determine the performance goals for the product. Choose a small number of areas in which to be careful and ruthlessly simplify the rest. Once learned, design for performance is a skill that is difficult to keep under control. However, time spent on performance is an investment, and a deliberate choice of investments is needed.

## References

1. D. Crocker, "Standard for the Format of ARPA Internet Text Messages," RFC 822 (August 1982).

2. N. Borenstein and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," RFC 1521 (September 1993).

3. K. Moore, "MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text," RFC 1522 (September 1993).

4. J. Postel, "Simple Mail Transfer Protocol," RFC 821 (August 1982).

5. C. Partridge, "Mail Routing and the Domain System," RFC 974 (January 1986).

6. P. Mockapetris, "Domain Names—Concepts and Facilities," RFC 1034 (November 1987).

7. P. Mockapetris, "Domain Names—Implementation and Specification," RFC 1035 (November 1987).

8. J. Myers and M. Rose, "Post Office Protocol—Version 3," RFC 1725 (November 1994).

9. M. Crispin, "Internet Message Access Protocol—Version 4, Revision 1," RFC 1730 (December 1996).

## Biography

Nick Shipman joined DIGITAL in 1982 after earning a B.Sc. in computer science from University College London. As a member of the Mail Interchange Group (MIG), Nick has been involved with the design and development of a variety of e-mail and directory server products and their test tools. He is currently a principal software engineer with the DIGITAL UK Engineering Group, where he is working on advanced development of Internet-based servers.

Kenneth M. Weiss
Kenneth A. House

# DIGITAL Personal Workstations: The Design of High-performance, Low-cost Alpha Systems

**The new DIGITAL Personal Workstations for Windows NT (a-series) and DIGITAL UNIX (au-series) incorporate a 21164 Alpha microprocessor, a highly integrated core logic interface, synchronous main memory and cache, and commodity PC parts. The traditional core logic chip set has been designed as a single-chip ASIC. The high-performance uniprocessor workstation includes a low-cost interrupt scheme, tight timing control of clocks for maximal performance, and a flash ROM interface.**

In 1995, DIGITAL began development of a low-cost system implementation of the 21164 microprocessor, incorporating emerging memory technologies to improve system performance. This paper discusses the major architectural and design features of the DIGITAL Personal Workstation a-series and au-series, low-cost, high-performance systems powered by the 21164 Alpha microprocessor. It focuses on some of the unusual design features incorporated into the new core logic chip, the 21174 application-specific integrated circuit (ASIC). The paper also addresses a novel clock distribution strategy with feedback for skew reduction, a low-cost interrupt scheme, and a capability to boot directly from reprogrammable flash read-only memory (flash ROM).

The original project was to build a platform for the DIGITAL Personal Workstation running the Microsoft Windows NT operating system. This system is designated as the a-series (e.g., 433a, 500a, 600a, etc.). Soon after shipping this product, the same hardware components were qualified and shipped with the DIGITAL UNIX operating system as the au-series (e.g., 433au, 500au, 600au, etc.).

## Motivation

For years, main memory technology has been based on fast-page-mode dynamic random-access memory (DRAM). In the personal computer (PC) market, fast-page DRAMs were soldered onto single in-line memory modules (SIMMs). Although the access times of these memories have been decreasing (i.e., 80 nanoseconds [ns]→ 70 ns → 60 ns), improvements in CPU technology and core logic chip sets have evolved more rapidly. In 1995, the PC market began to move to extended data out (EDO) memories, which were roughly twice as fast as the fast-page memories, and burst EDO was on the horizon. Several DRAM vendors began to talk about new synchronous DRAMs (SDRAMs), which provided even greater performance and the promise of reaching commodity price levels by 1997.

Reduced instruction-set computers (RISC) in general, and Alpha machines in particular, require faster memories. Alpha CPUs run much faster than the leading complex instruction-set computer (CISC) (e.g., Intel x86) processors and need a faster path to memory to keep pace with the demands for instructions and data. DIGITAL and other Alpha system developers have been satisfying this need with wider memory buses; for instance, the AlphaStation 500 and AlphaStation 600 series systems have a 256-bit memory bus, four times wider than the typical 64-bit bus on leading-edge PCs. Also, the larger external caches on DIGITAL's systems reduce the memory bandwidth requirements by keeping frequently used data local to the processor.

This advantage, however, was shrinking for two reasons: faster memories were being introduced for PCs, and low-cost systems with 256-bit memory buses and large external caches are difficult and expensive to build. It became clear that a new system design was required, one that could take advantage of the new memory technologies and one that could be implemented at a lower cost. This, in turn, would require the design of new core logic that would function as the CPU-to-memory interface.

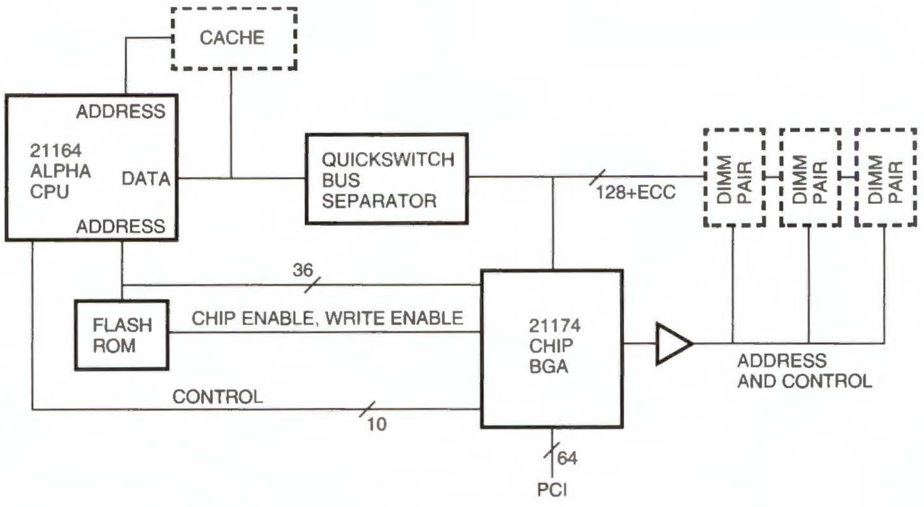
Although the system design was planned to take advantage of higher-performance memory technologies, the primary emphasis was placed on lowering system cost. The design team had a multipronged approach to meeting this goal. In addition to following the established Design for Manufacturing techniques and selecting low-cost components, we focused on creating a highly integrated, low-cost core logic ASIC.

## Design Overview

In traditional system-level designs, the memory data bus goes through the core logic chip set.[1-3] This approach isolates the CPU side of the data bus from the large electrical load of the memory chips and interconnect, thereby allowing the CPU to access its cache more quickly on the lightly loaded bus. This approach also gives the chip set access to the memory data for transfers to and from the I/O bus, while at the same time allowing the CPU concurrent access to the cache. Finally, this design allows the memory and CPU buses to be different widths, with the core logic chip set functioning as a multiplexer/demultiplexer between the two different buses.

However, there are disadvantages to this scheme. For example, each data line requires two pins in the chip set (and more, for the extra power and ground lines), significantly increasing the size and cost of the chip set. Also, memory latency is increased by the additional time needed to get data through the chip set to the CPU, which hurts performance.

In our design, the memory-to-CPU data bus goes *by* the core logic chip rather than *through* it. The data bus attaches to only one pin per bit. Figure 1 shows a simple block diagram of the system. To isolate the CPU and the backup cache (B-cache) from the memory bus, we used a separate bank of QuickSwitch bus separators. These bus separators are large complementary metal-oxide semiconductor (CMOS) pass transistors, which appear as either short circuits or open circuits, depending on the state of their enable pin.



Note: Components with dashed outlines are optional.

**Figure 1**
System Block Diagram

In the open mode, they allow the CPU to access its B-cache in isolation. When closed, memory data travels through the bus separator with a small (250 picoseconds [ps] through 5 ohms) propagation delay, thus avoiding the latency/performance penalty of going through the chip set.

Incorporating high-volume commodity parts into a design is less expensive than adding extra pins to a relatively low-volume, high-pin-count ASIC. The small size of the bus separators allows them to be placed closer to the CPU and cache for a shorter, quicker CPU interface. Finally, a low-end cacheless configuration can be built without them.

Another feature of our CPU-to-memory interface is that the "chip set" is a single chip. Although the original design called for three chips, new packaging technologies were making larger pin-count devices more affordable and manufacturable.[4] We compared various implementations from several ASIC vendors and eventually chose a single-chip implementation. (The Technology Choices section in this paper discusses the details.)

Finally, making the B-cache optional is a new feature for Alpha workstations. Recent Alpha workstations have been hampered by the relatively slow, fast-page-mode main memory coupled to an increasingly high-speed processor. With this imbalance, a large external B-cache has been essential to speed up CPU accesses by reducing average memory latency and memory bandwidth requirements.

We investigated an internal research project in which very fast memory systems were coupled to the 21164 CPU. An experimental cacheless machine used a 512-bit-wide (fast-page mode), low-latency memory system and performed very well on many memory-intensive benchmarks. This work helped inspire the cacheless concept for the 21164 CPU; however, the performance of the experimental system was relatively mediocre on some benchmarks (cache intensive), which led us to conclude we needed to offer an optional B-cache as well.

## Technology Choices

As mentioned previously, system development was concurrent with some significant changes to main memory technology. Our main issue was determining which of the new technologies—EDO, burst-EDO, synchronous, or Rambus—would be the next standard in the commodity PC business. This was a key decision: a wrong choice could lead to prohibitively expensive memory, effectively making the system a high-cost platform.

Our first analysis pointed to burst-EDO, since it was the most compatible with the existing fast-page memories. Several months into the design, however, many of the key memory vendors abandoned burst-EDO in favor of SDRAM. With help and advice from memory component engineers within DIGITAL, we switched over to synchronous memory. (This was a fortuitous decision, since commodity PC platforms are now moving to SDRAM, assuring its place as the next high-volume memory technology.)

From our first look at memory bandwidth with the new SDRAM, we realized that designing a B-cache to keep pace with it would be difficult. The system's memory, running at 66 megahertz (MHz), would have more bandwidth than any other Alpha workstation's B-cache. In fact, it was difficult to design a B-cache that was not the limiting factor in memory bandwidth. From the cache technologies supported by the 21164, we selected synchronous flow-through static RAM cache parts. Although this choice was expensive, an external cache that significantly increased performance was imperative for our product.

Our next major decision was choosing our ASIC technology and package. After evaluating quotes for different chip configurations from several vendors, we selected the single-chip implementation in a 474-pin ceramic ball-grid array package (BGA) offered by International Business Machines Corporation (IBM). The package measures only 1.0 by 1.25 inches and uses flip-chip technology, in which the die is bonded directly to the ceramic package without lead wires. The ceramic carrier is a seven-layer design that connects the die pads to a 50-mil (.050-inch) BGA for assembly onto the printed circuit board. Contrary to our initial concern, the package was not too difficult to route on the printed circuit board. In addition to its small size and reasonable price, the package offered low inductance (short) power, ground, and signal connections, which provided for good signal integrity. Figure 2 shows a photograph of the ASIC package.

Most of the other technology choices we faced do not warrant specific mention, except for the system enclosure. Since our system was destined to become a

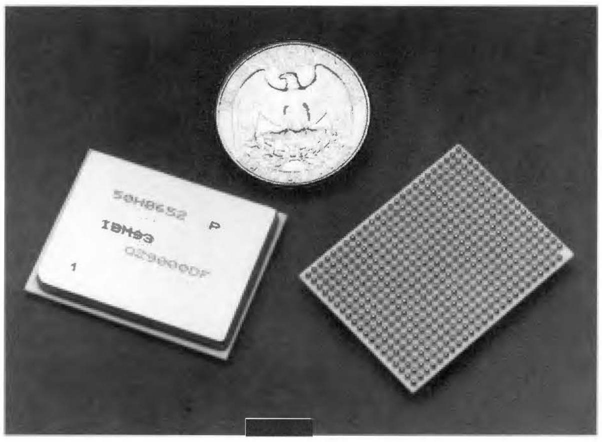


**Figure 2**
Core Logic ASIC Package

DIGITAL Personal Workstation, it had to be highly compatible with DIGITAL's leading-edge Intel processor-based workstation (i-series). The use of common components makes it easier for development, manufacturing, distribution, and service. As a result, we had to fit our system into an existing DIGITAL PC enclosure, use the same power supply, have similar option configurations, and so forth. This constraint had both good and bad ramifications, which we discuss in a later section, Logic Partitioning and Enclosure.

## System Clocks

In *Zen and the Art of Motorcycle Maintenance,* Robert M. Pirsig argues that one can look at a complex system from many different viewpoints.[5] In that spirit, the logic/system design is illuminated by its clocking details. We will touch on the major points of interest in this section.

### *Clocking Overview*

The main memory system was designed to run as high as 66.6 MHz, synchronous to the 21164 CPU. The main clock source is the CPU's internal reference clock, which runs at the full CPU speed. The 21164 CPU provides a programmable, divided clock for the system interface, called SysCLK. At power-on, the divide ratio is determined based on module-level inputs (through the interrupt request [IRQ] lines) and is set so that the system clock runs as fast as possible, up to 66.6 MHz.

For example, a 600-MHz CPU would have its divide ratio set to nine, yielding a 66.6-MHz SysCLK, whereas a 500-MHz system would set the ratio to eight for a 62.5-MHz system clock. A programmable, delayed version of SysCLK, SysCLK_2, also is available from the 21164 CPU and is used as the main clock source in the 21174 core logic ASIC. In the system, SysCLK_2 is delayed nearly a full clock cycle, so that by the time it reaches the core logic ASIC, it arrives roughly coincident to SysCLK.

### *Clock Divisors and Clock Domains*

Figure 3 shows a simplified block diagram of the logic/system clock system. Once it enters the core logic ASIC, SysCLK_2 makes its way to the input of a phase locked loop (PLL) by means of a multiplexer (MUX). SysCLK_2 passes straight through the MUX, which is a power management feature and is not currently used. The output of the PLL goes into two programmable divisors, one to generate CLK, the main clock for the core logic ASIC, and the other to generate PCLK, the Peripheral Component Interconnect (PCI) bus logic interface to the core logic ASIC. Since CLK is the feedback source to the PLL, Sys_CLK and CLK are always at the same frequency.

The external PCI bus is specified to run as high as 33.3 MHz, a limit required for successful operation of industry-standard I/O option cards. The core logic ASIC runs its PCI interface at PCI speeds and thus needs a separate 33.3-MHz PCI clock. Since SysCLK is running at 66.6 MHz, we can simply divide it by two to arrive at a nominal 33.3-MHz PCLK.

In addition, the core logic ASIC has a clocking system capable of running these clock domains at many different ratios. It allows for operation faster than 66.6 MHz and includes divide ratios to generate PCI clocks that do not exceed 33.3 MHz. For example, one interesting frequency for the core logic ASIC is 83.3 MHz. With the CLK divisor set at two ($N = 2$ in the drawing), the output of the PLL will be running at 166.6 MHz. With the PCLK divider at five ($P = 5$), the resultant PCI clock rate becomes 33.3 MHz. The dividers are clocked by both a positive PLL clock and an inverted PLL clock, allowing for symmetry in the output clocks for odd divisors.

Any significant drift or skew in the two major clocking domains, CLK and PCLK, would result in timing problems (mostly hold time) as signals cross from one domain into the other. Therefore, these clock domains had to be in nearly perfect alignment. To achieve this, we ensured that the clock distribution trees had tightly controlled delays, that the core of the clock dividers was implemented structurally (at the gate level), and that all associated logic was balanced in the layout. Then we did a complete timing analysis to ensure proper operation.

So, why bother with all this? An odd frequency mix such as this one is often achieved with asynchronous boundaries and often with much pain. Asynchronous interfaces are difficult to design and to verify. Furthermore, the transitions across these boundaries can often be slow and add data latency. The 21174 core logic ASIC design allows for flexible operation frequencies, while maintaining a synchronous design.

The 21174 ASIC was designed to accommodate several different frequency combinations, and the internal data and control signals that cross the CLK to PCLK boundary were implemented accordingly. As the design progressed, however, we found it was too difficult to run the external memory and B-cache subsystems faster than 66.6 MHz, given the current technologies. As an unfortunate result, these features have not yet been used.

### *SDRAM Clock Generation*

As shown in Figure 3, the memory/DRAM clocks are generated from the 21174 ASIC. Normally, large ASICs—such as this one—have relatively slow I/O cells and significant delay variation from chip to chip. This typically makes large ASICs a poor choice for driving critical clocks (such as the memory clocks) in

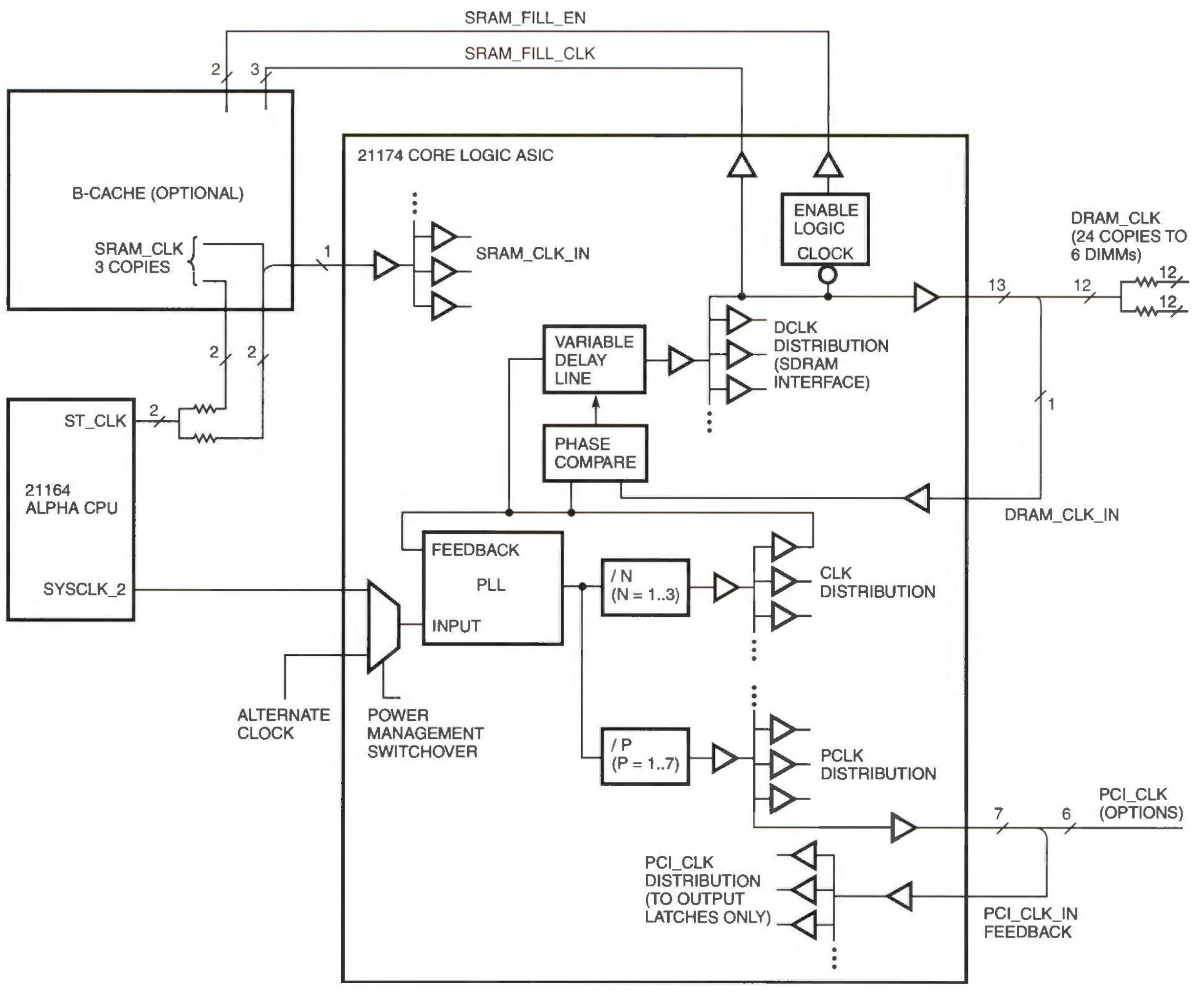


**Figure 3**
Block Diagram of Clocks

high-speed system design, because tight timing control of clocks is a key factor for maximal performance. Therefore, most high-speed designs use costly, skew-controlled buffer chips and elaborate distribution schemes to get tightly controlled clocks. Often, these circuits use PLLs as well to control the absolute arrival time of the clocks. All these decisions result in good performance but at a price, both in cost and board area.

Like other large ASICs, the 21174 has a large timing variation on its I/O cells. Unlike other large ASICs, it employs a novel feedback circuit that automatically compensates for delay uncertainty and ultimately delivers tightly aligned clocks. Figure 4 shows some details of this circuit, which is used to generate the main memory clock, DRAM_CLK. This clock originates inside the ASIC as a copy of the tightly aligned internal clock, CLK, which is then distributed through a variable delay line. The resultant delayed clock is then sent, in multiple copies, out of the chip to the SDRAM memory dual in-line memory modules (DIMMs).

The variable delay line consists of 128 independent delay cells, which have a delay of roughly 200 ps each. Each delay cell maintains the same polarity of the clock but inverts it twice to compensate for different gate rise and fall times. By adjusting the number of delay cells in the path, the 21174 ASIC can control the timing of the output clock. With that accomplished, the core logic ASIC must now determine and control the number of delay elements used.

As Figure 4 shows, the 21174 ASIC generates 13 identical copies of the DRAM clock; 12 copies go to the memory arrays (two per DIMM). The thirteenth copy feeds back into the ASIC. Conceptually, the extra copy goes to a phase detector, which compares it against its own internal clock, CLK. If the sampled DRAM_CLK leads CLK, then the phase detector automatically increases the delay of the DRAM_CLK variable delay line by adding one more element to the delay chain. Conversely, if DRAM_CLK trails CLK, one element of the variable delay line is removed.

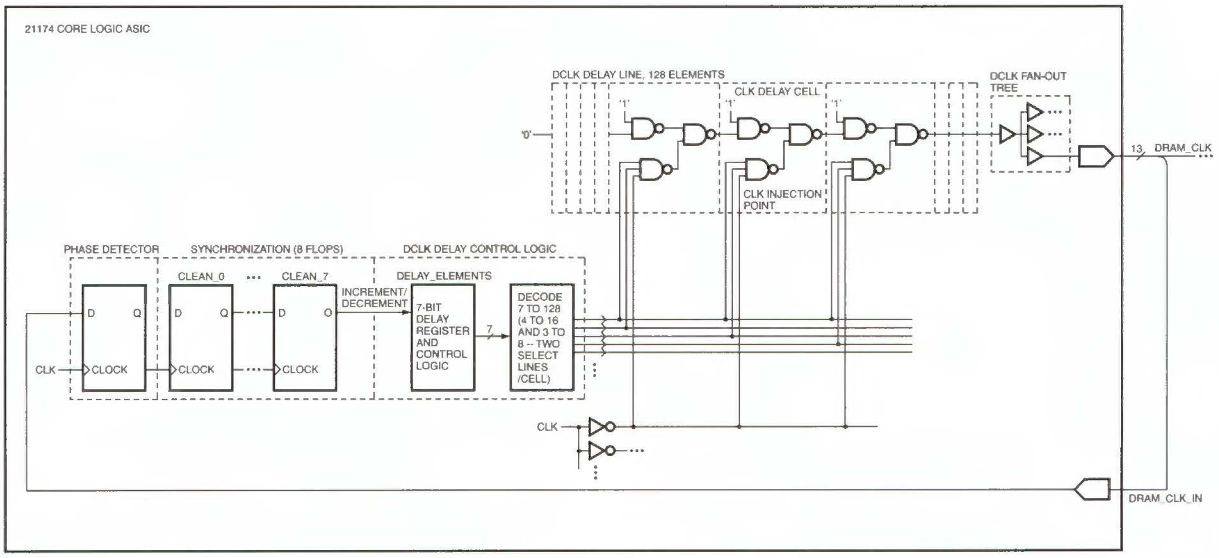


**Figure 4**
DRAM_CLK Clock Aligner

The alignment circuitry continues to add (or remove) one delay element until the clocks are slightly past the optimal alignment; at that point, it then subtracts (or adds) a delay element. The circuit thus brings the clocks close to alignment and then toggles back and forth by one delay element, which adds slight but acceptable jitter to the clocks.

The phase detector is logically a D flip-flop, with CLK driving its clock input and DRAM_CLK as its data input. The two output states thus represent "CLK leads DRAM_CLK" and "CLK trails DRAM_CLK." The circuit pings back and forth between the final two delay elements because the phase detector does not directly indicate that clocks are in alignment. Also, once the clocks are in alignment, timing could drift because of delay changes caused by temperature or voltage variations, which require the circuit to be left on.

The goal of this clock circuit is to precisely align the phase detector's clock, CLK, with its data, the input copy of DRAM_CLK. This nearly guarantees that the flop will be operating in violation of its normal setup-and-hold timing window because its data are changing at the clock edge. (State elements like to have a setup-and-hold window around the clock where the data is stable at either a 0 or a 1.) This violation can cause both uncertainty in its output state, as well as problems with metastability in the flip-flop.

Fortunately, we were able to use a special metastable hardened flip-flop in IBM's CMOS5 library as one of the phase detectors in the 21174 ASIC. (The circuit offers two phase detectors; each is implemented with a different flop and is selectable during power-up.) Also, the output of the phase detector was sent through seven more metastable flops in series. These flops clean up and synchronize any metastable state that may occur and send a clean output to the variable delay line control circuitry.

Although ASICs can have a large variation in output delay from chip to chip and for different voltages and temperatures, cell delays are correlated to a large degree within a chip. In short, if one I/O cell in a chip is running fast, they will all be running fast. Thus, by lining up one of the DRAM CLKs, they are all in rough alignment with it. Although this compensates for the output buffer delay, the rest of the clock feedback path is not automatically correlated.

The external printed circuit board wiring, however, has relatively little skew contribution because propagation delays in etch are relatively constant and easily calculated. (The wiring will correlate to the etch delay of the other clocks going to the DIMMs.) The input buffer cell delay of the feedback clock is not correlated, but this is relatively fast and not consequential.

Some additional aspects of the circuitry are worth mentioning. For example, in the 21174 ASIC implementation, the automatic delay alignment circuitry can be selectively enabled or disabled by software (it powers up disabled). Software can force the number of delay elements. This allows us to derive more elaborate schemes of fine-tuning the SDRAM clocks, such as advancing the clock when there are more DIMMs in the system and retarding it when there is light memory bus loading.

Note that other nonrelated signals that use the same internal DCLK and the same I/O cell type are delay correlated to the DRAM clock signals. Several other signals to the B-cache, which needed a tight timing relationship to the memory, were generated in this fashion. Also, most of the memory interface signals are generated from DCLK, which made the normally difficult job of interfacing to the high-speed memory a relatively easy one in terms of timing.

The clock-alignment circuitry was one of the first things we checked after we powered on the first system. As soon as we wrote the bit to enable the auto alignment feature, the clocks snapped into the calculated/expected positions.

## B-cache Clocks

Each of the two ST_CLK pins on the 21164 CPU are split into two separate copies through a resistor-splitting network. The resultant four copies of ST_CLK route to the B-cache module. Three of these copies are available for the synchronous static RAMs on the cache module, and the fourth copy routes to the 21174 core logic chip. The current implementation of the B-cache module uses only one copy of ST_CLK, which is buffered and sent to the individual synchronous static RAM chips.

The system platform delivers three additional copies of the DRAM_CLKs to the B-cache module, called SRAM_FILL_CLK, which also take advantage of the DRAM_CLK alignment just described. These clocks can be used to clock the synchronous static RAMs during memory fills to give the least amount of clock skew between the memory and the cache. A separate set of time-aligned signals, SRAM_FILL_EN, can be used to switch between the ST_CLK and SRAM_FILL_CLK sources.

The 21174 ASIC implements a victim buffer for dirty (updated) B-cache data being evicted to memory.[4] To optimize victim eject timing to the core logic ASIC, the system uses a forwarded copy of the ST_CLK clock sourced from the 21164 CPU and sent to the 21174 ASIC (see Figure 3). This clock, SRAM_CLK_IN, works similarly to the 21164 CPU's wave pipeline feature and allows ejection of a victim from the cache to the 21174 ASIC in five SysCLK cycles rather than the eight cycles it would have taken without it.[6] For flexibility in future B-cache upgrade cards, this clock routes through the B-cache module, which allows its timing to be fine-tuned for the specific B-cache module parts. Figure 5 shows this timing.

## Flash ROM and Boot

The 21174 core logic ASIC implements the boot code, the console firmware, and the nonvolatile RAM (NVR) interface in the system, another example of a system feature that was integrated into the core logic ASIC. A single, one-megabyte (MB) flash ROM holds the power-on self-test (POST), two 16-kilobyte copies of NVR, and the AlphaBIOS and SRM consoles. These separate components were combined to save board space that would have been used by separate parts: serial ROM, NVR, and two programmable ROMs.

The more highly integrated design also reduced costs, which was consistent with our design goals. The single flash ROM is connected on the address bus between the CPU and the core logic ASIC (see Figure 1). By placing these critical software components logically closer to the CPU, rather than on the I/O bus, we provided a greater ability to diagnose the system when not all of its parts are working. Flash ROM was used

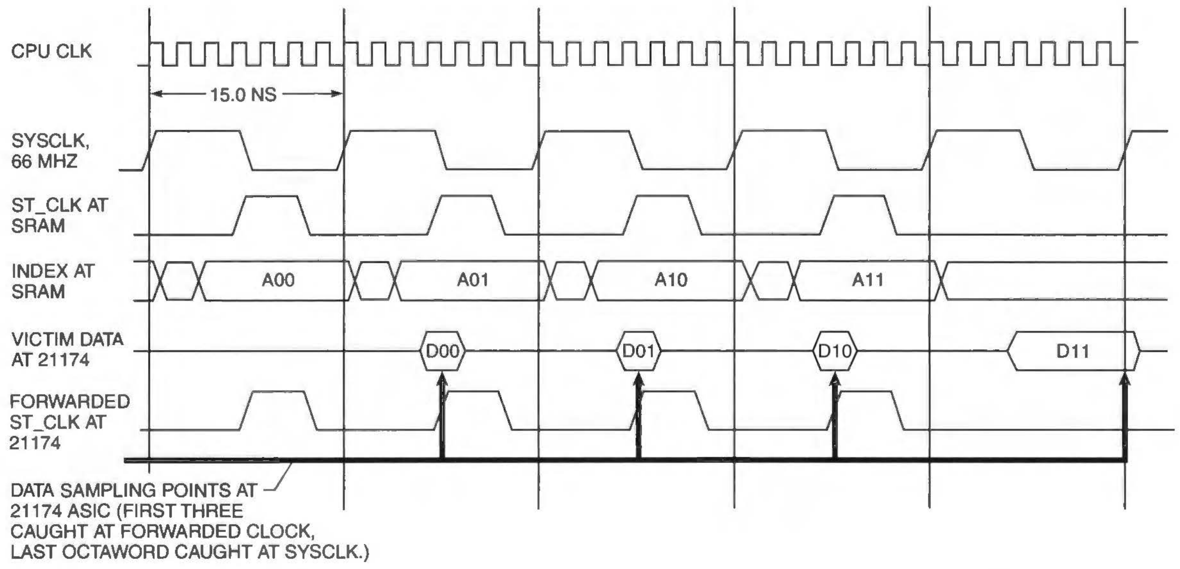

CPU CLK
15.0 NS
SYSCLK, 66 MHZ
ST_CLK AT SRAM
INDEX AT SRAM
A00
A01
A10
A11
VICTIM DATA AT 21174
D00
D01
D10
D11
FORWARDED ST_CLK AT 21174
DATA SAMPLING POINTS AT 21174 ASIC (FIRST THREE CAUGHT AT FORWARDED CLOCK, LAST OCTAWORD CAUGHT AT SYSCLK.)

**Figure 5**
Victim Eject Timing Diagram

to ease updating the firmware in the field, allowing upgrades from a floppy diskette (possibly downloaded from the DIGITAL Web site).

The initial power-on boot sequence on the system is also unique among Alpha workstations. All Alpha CPUs can use a serial ROM to load the primary instruction cache (I-cache) at power-on; once the I-cache is full, execution begins. Alternatively, the 21164 CPU can bypass the serial ROM load and begin executing directly from memory. These instruction stream (I-stream) accesses miss in the CPU's empty internal caches, causing external fills. The system implements this form of power-on, using the core logic ASIC to intercept I-stream fills and retrieve the data from flash ROM. This saves the board space and cost associated with either a specialized serial ROM part or the logic to serialize a standard ROM.

At initial power-on, the core logic ASIC interprets CPU requests for reads from addresses 00.0000.0000 through 00.03FF.FFFF as requests for data from the flash ROM. The transaction begins with a read_block_miss command from the CPU. The core logic ASIC asserts cack_l to acknowledge the command and then asserts addr_bus_req to request the private use of the CPU-to-21174 ASIC address bus. It then issues reads to the flash ROM by passing address and control information on the now-reserved address bus.

Since the core logic ASIC owns the bus, it does not have to adhere to conventional usage: the address bits are free for reassignment. Addr<31:12> is used as the byte address into the flash ROM; the eight-bit datum is returned on addr<11:4>. Sixty-four successive bytes read from the flash ROM are packed into a buffer in the core logic ASIC and returned to the CPU to complete the original fill request.

There are two different address ranges used for flash ROM fills: one starting at address 00.0000.0000 (to allow power-on from code in flash ROM) and another at address 0F.FC00.0000 (above any possible memory). Both ranges access the same flash ROM data. POST code quickly jumps to the high address range, disabling the low range and freeing those addresses for use by memory before it begins to size the DIMMs.

Byte read and write access to the flash ROM is also supported for access to NVR and for updating the firmware. This address range starts at C7.C000.0000. Only two function-specific pins on the core logic ASIC are used for the flash ROM interface, write enable (flash_we_l, deasserted during fills) and chip enable (flash_ce_l). The flash ROM's output enable is controlled through an address line, addr<39>.

A socket is provided on the system mother board to allow for the use of a real serial ROM part. The circuitry automatically detects the presence of a serial ROM and will direct the 21164 CPU to boot from the serial ROM port if the part is installed. This allows a serial ROM to be used in case of damage to, or inadvertent corruption of the flash ROM. This feature is also useful for module-level debug, because serial ROM parts with specific test scripts can be made.

## Interrupt Controller

The system provides separate inputs for every possible PCI interrupt (Figure 6), avoiding the problems that shared interrupts bring: longer latency, unnecessary I/O, configuration restrictions, and so forth. We implemented a serial I/O scheme to handle PCI interrupts and miscellaneous I/O. This has the advantage of bypassing the Industry Standard Architecture (ISA) interrupt handlers entirely, except for true ISA interrupts. Many previous designs run the PCI interrupts through the PCI-to-ISA bridge, which requires additional I/O to determine the device that needs attention. In the design of the system, we tried to avoid the need for time-wasting I/O accesses wherever possible, moving the access closer to the CPU if it could not be eliminated entirely. Further down the hierarchy of buses (CPU to PCI to ISA), latencies increase and the possibility of contention rises. For a high-frequency RISC CPU such as the 21164, stalling the CPU has a significantly higher penalty than for slower processors.

The core logic ASIC uses only three pins to control external shift registers for interrupts and general-purpose inputs and incorporates a fourth pin that handles general-purpose outputs (such as control bits or light-emitting diodes [LEDs]). The external logic is shown in Figure 6, where int_clk is the shift clock into the external shift registers, int_sr_load_l is the load pulse for both input and output, int_sr_in is the serial input stream, and the serial output is on gp_sr_out.

Inside the core logic ASIC, we used several registers in the interrupt controller. All interrupts and general-purpose inputs are readable in the 64-bit int_req register, letting the interrupt dispatch code determine relative priorities. A corresponding int_mask enables or disables each input as an interrupt. To save external inverters, an eight-bit register (int_hilo) is used to change the polarity of individual inputs in the low byte (inputs are assumed to be active-low signals, but some inputs are contrary). Another eight-bit register is int_route, which can selectively route some inputs to different interrupt lines on the CPU; although most interrupts are delivered on the CPU's irq_h<1> line, this feature allows the clock to come in at a higher priority and other inputs to cause a machine check, a power-fail interrupt, or a halt condition.

The interrupt controller can be configured for the number of bits used, in blocks of eight, for as many as 64 bits. We set the interrupt controller for 32 bits, implemented as four, cascaded, eight-bit parallel-to-serial shift registers and two serial-to-parallel parts, which was sufficient for the system. The timing of the shift clock is adjustable to balance the cost of the shift

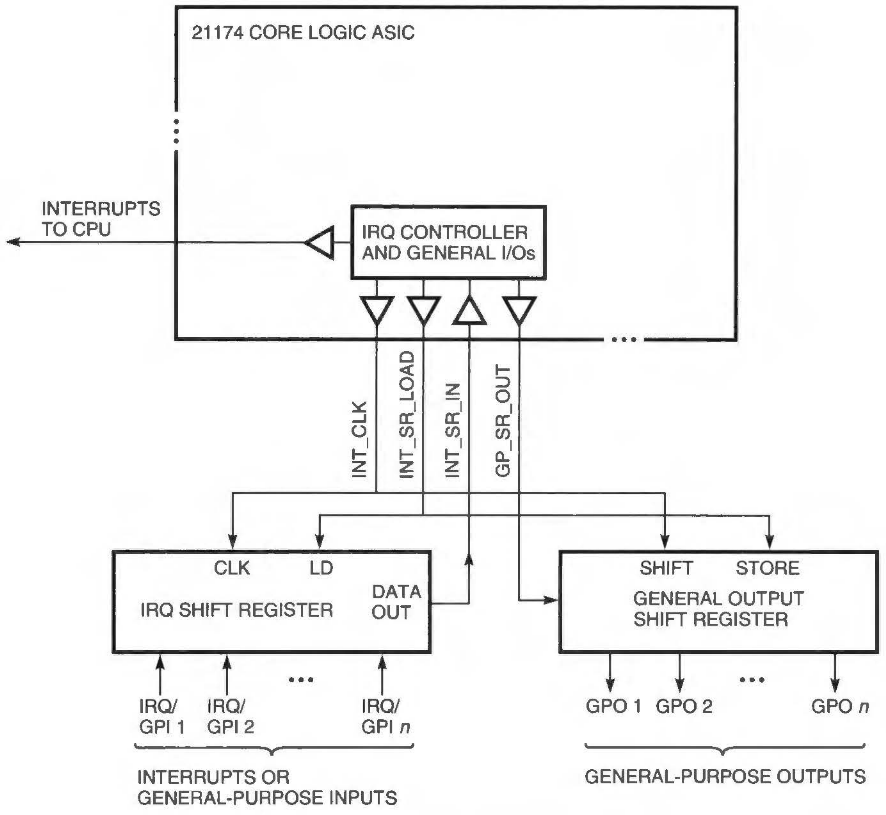


**Figure 6**
Interrupts and General-purpose I/Os

registers against the interrupt latency. These two parameters are maintained in the int_cnfg register.

A small additional interrupt latency was introduced by the serial interface. We judged this to be an acceptable compromise, since we eliminated several I/O accesses per interrupt. Note that when multiple interrupts occur within the same shift cycle, all are available to the CPU's interrupt dispatch routine simultaneously, with only one read of the core logic ASIC's int_req register.

## Soft Reset

The core logic ASIC generates and drives all the system reset lines, including its own. It receives a DC_OK (DC power supply levels charged and okay) signal from an external voltage monitor, waits for its internal PLL to lock, and then delays for a programmable (by means of external pull-up/pull-down resistors) number of cycles before deasserting reset to the system.

Also, the core logic ASIC implements a soft reset feature whereby software can reset the entire machine, with the exception of a few key registers in the core logic ASIC. This allows software to reconfigure certain key CPU and core logic ASIC clocking parameters, hit reset, and acquire the new timing.

For example, the state of the CPU's interrupt pins during system reset determines the external clock characteristics of the 21164, such as the SysCLK divide ratio and the SysCLK_2 delay (mentioned earlier). At power-on, these are set by pull-up and pull-down resistors on the module to the slowest speed. After software establishes the CPU's operating frequency, it can configure the interrupt logic to drive these lines to the correct value and reset the machine. The next time the CPU restarts, software recognizes the right clock parameters and continues with the boot code.

## Embedded Real-time Counter and Timer Interrupt

A 64-bit real-time counter in the rt_count register, ticking with the system clock, is useful for microscopic timing measurements. The core logic ASIC provides an int_time alarm register to interrupt the CPU at specific values of rt_count. The system uses the real-time clock in one of its peripheral I/O chips (super I/O) for the system timekeeper, bypassing the ISA interrupt controller and bringing its interrupt straight to the input shift registers, which saves PCI bandwidth and processor cycles.

## I²C Interface

Two pins on the core logic ASIC were set aside for general-purpose inputs or outputs, under total software control. In the system, these are used by software to implement an I²C (inter-IC) interface for clock and data. This serial interface allows input of static configuration data from the six DIMMs and the B-cache module. At power-on, firmware reads information from small, serial electrically erasable read-only memories (EEROMs) on each of these modules. Thus the interface can easily establish system configuration as well as set up memory and cache timing. The DIMM I²C ROM contents are defined by the Joint Electron Devices Engineering Councils (JEDEC) standards and can be programmed in place.

Since the I²C data is read only once at power-on, it was not important to make this interface fast. Consequently, it was implemented as a "bit-bang" port, in which each transition on each line is controlled by firmware accesses to logic registers. Firmware is responsible for timing of the protocol's clock and for setup and hold of the data. Also, firmware handles deserialization (byte packing) of the incoming data stream.

## Logic Partitioning and Enclosure

Most of the system logic is partitioned onto two boards: a riser card and the mother board (often referred to as the main logic board [MLB]). The riser card is used for all components common across the DIGITAL Personal Workstation line; it includes five option slots, audio, and Ethernet logic. All internal cables connect to the riser, which is intended to be common between the platforms. Figure 7 shows a photograph of the system logic.

Because of the shared riser, the a-series and i-series systems have much in common. For example, they use the same PCI–PCI bridge chip, with identical option slot layouts. The CD-quality audio and 10/100 megabits per second Ethernet logic are common, as are the bulkhead cards for these signals.

The MLB contains the CPU, core logic chip (set), cache, memory, and miscellaneous external connectors. Figure 8 shows the optional B-cache module and a custom-designed memory DIMM. Because there are no internal connections, it is easy to remove an Intel MLB and replace it with an Alpha board.

In a design compromise, we added a PCI–IDE chip on the MLB to match the partitioning dictated by existing i-series machines, which have integrated device electronics (IDE) built into their core logic. The natural place for such a device would have been on the riser card.

A graphics controller was not embedded so the customer could select from adapters of varying cost and performance. For those users with a CPU-intensive application who do not need high-quality, high-performance graphics, a less expensive adapter may be adequate. Demanding users, such as those doing mechanical computer-aided design (CAD), will find that the high-end graphics cards offer a significant boost in quality and performance.

A small computer systems interface (SCSI) controller is provided as a PCI option card for several reasons. First of all, it was more expensive to embed SCSI and provide a bulkhead card for an external connection.

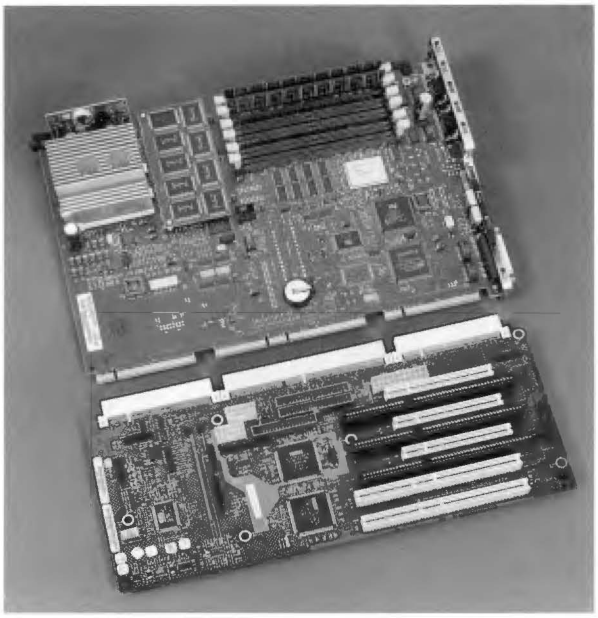


**Figure 7**
Electronics, Mother Board, and Riser for the DIGITAL Personal Workstation, a-Series

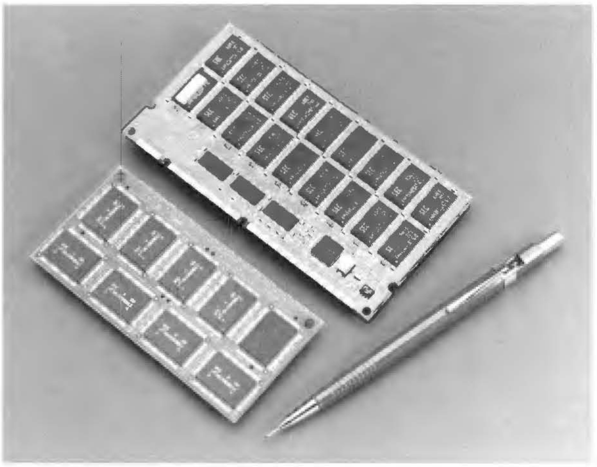


**Figure 8**
Optional 2-MB B-Cache and 64-MB DIMM

Using an option card allows us to move to better SCSI solutions as they become available. Customers with a light disk I/O load may choose to use a less expensive IDE hard disk.

Although there were many benefits to sharing so many components between the a-series (Alpha) and the i-series (e.g., Intel x86), the development process was not conflict-free. The Alpha development team faced several significant problems. Because the i-series development started sooner, much of the feature set and logic partitioning was already committed before we had a chance to participate in the design. The original enclosure proved to be inadequate, and we had to make changes for cooling, FCC containment, and mechanical support of option cards. Also, there were difficulties that reflected the different market requirements for PCs and workstations. For instance, an external HALT button, which forces a running machine to the firmware console, is a standard feature in the UNIX and OpenVMS markets but is missing in the DIGITAL Personal Workstation.

## Results and Summary

The 21174 core logic ASIC was implemented as a standard-cell ASIC design and uses about 320,000 cells (250,000 gate equivalent) on a 7.2-millimeter square die, routed in five layers of metal. The design uses 384 signal pins in a 474-pin ceramic BGA package; the remaining pins are used for power and ground. The DIGITAL Personal Workstation a-series and au-series include the 21164 Alpha CPU, the 21174 core logic ASIC, synchronous DRAM, and a 64-bit PCI bus.

## Acknowledgments

The authors would like to thank all the people who contributed their skills and time to the successful design of the Alpha microprocessor-based personal workstations known internally as the PYXIS/MIATA/MX5 projects. Special thanks are due to the core engineering design, development, and debug team: Arlens Barosy, Frank Calabresi, Jim Delmonico, Jeff Forsythe, Frank Fuentes, Paul Hill, Bob Hommel, Mark Kelley, Yong Oh, Rob Orlov, Rajen Ramchandani, Dan Riccio, Don Rice, Ty Rollin, Arnold Smith, and Dean Sovie. Many thanks also are in order for our qualification team, including Bill Grogan, Bob O'Donnell, Phil Puris, Lynne Quebec, and Matt Twarog; our engineering manager, Fred Roemer; our product manager, Keith Bellamy; and our group's technical writer, Carmen Wheatcroft. We also wish to acknowledge Dave Conroy from the Systems Research Center for his work on the cacheless machine. Finally, special acknowledgment is due to Reinhard Schumann, who contributed an extraordinary amount of inspiration to the design and perspiration to the implementation.

## References

1. S. Nadkarni, W. Anderson, L. Carlson, D. Kravitz, M. Norcross, and T. Wenners, "Development of DIGITAL's PCI Chip Sets and Evaluation Kit for the DECchip 21064 Microprocessor," *Digital Technical Journal,* vol. 6, no. 2 (Spring 1994): 49–61.

2. J. Zurawski, J. Murray, and P. Lemmon, "The Design and Verification of the AlphaStation 600 5-series Workstation," *Digital Technical Journal,* vol. 7, no. 1 (1995): 89–99.

3. *Pentium Processors and Related Products* (Mt. Prospect, Ill.: Intel Corporation, Order No. 241732-002, ISBN 1-55512-239-6, 1995).

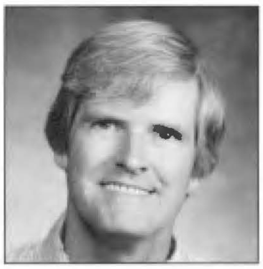
4. R. Schumann, "Design of the 21174 Memory Controller for DIGITAL Personal Workstations," *Digital Technical Journal,* vol. 9, no. 2 (1997): 57–70.

5. R. Pirsig, *Zen and the Art of Motorcycle Maintenance* (New York: William Morrow & Co., 1974).

6. *DIGITAL Semiconductor 21164 (366MHz Through 433MHz) Alpha Microprocessor Hardware Reference Manual* (Maynard, Mass.: Digital Equipment Corporation, Order No. EC-QP99A-TE, 1996).

## Biographies

**Kenneth M. Weiss**
Ken Weiss joined DIGITAL in 1983 as a software engineer developing CAD (computer-aided design) applications for signal integrity analysis and physical design. He later joined the Alpha Workstations Group as a principal hardware engineer and was the engineering project leader for the 21174 ASIC and the DIGITAL Personal Workstation a-series machines. In addition, Ken designed and implemented the ASIC and module-level clocking system and contributed to the signal integrity and timing verification of the system. He received a B.S. in electrical engineering from Cornell University in 1983 and an M.S. in computer science from Boston University in 1991. Ken is now working for Sun Microsystems.



**Kenneth A. House**
Kenny House is a principal software engineer for Workstations Engineering and was the team leader for verification of the 21174 core logic ASIC in the Alpha microprocessor-based DIGITAL Personal Workstation. Prior to this project, he worked for AlphaStation Engineering in software support and I/O integration, for VAXstation Engineering as liaison between operating systems and hardware engineering groups, and for DECmate Engineering on firmware, diagnostics, drivers, and simulation. Kenny joined DIGITAL in 1992 after fifteen years as an independent consultant. He holds a U.S. patent for a SCSI bus extender and was awarded a B.S. in mechanical engineering from the Massachusetts Institute of Technology in 1969.

Reinhard C. Schumann

# Design of the 21174 Memory Controller for DIGITAL Personal Workstations

**DIGITAL has developed the 21174 single-chip core logic ASIC for the 21164 and 21164PC Alpha microprocessors. The memory controller in the 21174 chip uses synchronous DRAMs on a 128-bit data bus to achieve high peak bandwidth and increases the use of the available bandwidth through overlapped memory operations and simultaneously active ("hot") rows. A new prediction scheme improves the efficiency of hot row operations by closing rows when they are not likely to be reused. The 21174 chip is housed in a 474-pin BGA. The compact design is made possible by a unique bus architecture that connects the memory bus to the CPU bus directly, or through a QuickSwitch bus separator, rather than routing the memory traffic through the core logic chip.**

As microprocessor performance has relentlessly improved in recent years, it has become increasingly important to provide a high-bandwidth, low-latency memory subsystem to achieve the full performance potential of these processors. In past years, improvements in memory latency and bandwidth have not kept pace with reductions in instruction execution time. Caches have been used extensively to patch over this mismatch, but some applications do not use caches effectively.[1]

This paper describes the memory controller in the 21174 application-specific integrated circuit (ASIC) that was developed for DIGITAL Personal Workstations powered with 21164 or 21164A Alpha microprocessors. Before discussing the major components of the memory controller, this paper presents memory performance measurements, an overview of the system, and a description of data bus sequences. It then discusses the six major sections of the memory controller, including the address decoding scheme and simultaneously active ("hot") row operation and their effect on latency and bandwidth.

## Project Goals

At the outset of the design project, we were charged with developing a low-cost ASIC for the Alpha 21164 microprocessor for use in the DIGITAL line of low-end workstations.[2,3] Although our goal was to reduce the system cost, we set an aggressive target for memory performance. Specifically, we intended to reduce the portion of main memory latency that was attributable to the memory controller subsystem, rather than to the dynamic random-access memory (DRAM) chips themselves, and to use as much of the raw bandwidth of the 21164 data bus as possible.

In previous workstation designs, as much as 60 percent of memory latency was attributable to system overhead rather than to the DRAM components, and we have reduced that overhead to less than 30 percent. In addition, the use of synchronous DRAMs (SDRAMs) allowed us to provide nearly twice as much usable bandwidth for the memory subsystem and reduce the memory array data path width from 512 bits to 128 bits. Figure 1 shows the measured latency and bandwidth

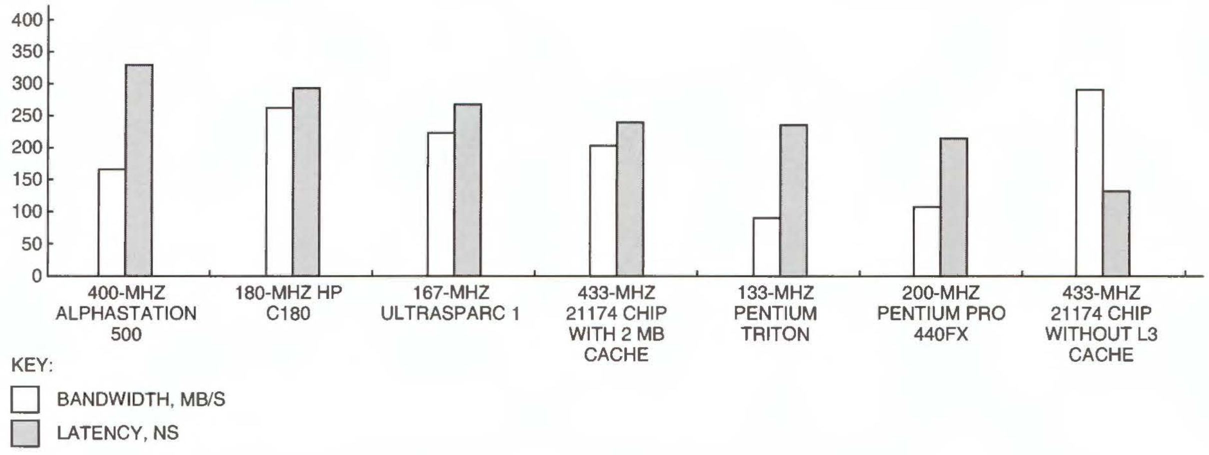


**Figure 1**
Measured Latency and Bandwidth

characteristics attainable with the 21174 chip in the DIGITAL Personal Workstation Model 433a and compares that data with measurements for several other systems. The STREAM (McCalpin) bandwidth measurements were obtained from the University of Virginia's Web site.[4] The memory latency was measured with a program that walks linked lists of various sizes and extrapolates the apparent latency from the run times.[5] Note that the measured latency also includes the miss times for all levels of cache.

Despite our ambitious memory performance goals, our primary goal was to reduce cost. We considered eliminating the level 3 (L3) cache found in earlier DIGITAL workstations from our system design.[6] Previous research indicated that a 21164 microprocessor-based machine without an L3 cache could achieve good performance, given good memory latency and bandwidth.[7] Clearly, elimination of the L3 cache would reduce performance; we chose instead to retain the cache but as an optional feature.

Typically, memory controllers used with Alpha microprocessors have used a data path slice design, with two or four data path chips connecting the CPU to the DRAMs. Early in our design process, we realized that these data path chips could be eliminated if the memory cycle time were made fast enough for noninterleaved memory read data to be delivered directly to the CPU. Previous designs used fast-page-mode DRAMs, but they could not be cycled quickly enough to provide adequate bandwidth from a 128-bit-wide memory. Consequently, a 256-bit or 512-bit memory data path was used in those designs, and data from the wide memory was interleaved to provide adequate bandwidth to the CPU. Recently, several types of higher bandwidth DRAMs have been introduced, including extended data out (EDO), burst EDO, and SDRAMs. All three memory types have adequate bandwidth for direct attachment to the data pins of the 21164, and all three are available at little or no price premium over fast-page-mode DRAMs.

We eventually chose SDRAMs for our design because their bandwidth is better than that of the other two types, and because the industry-standard SDRAMs use a 3.3-volt interface that is fully compatible with the 21164 data pins.[8] Fortuitously, SDRAMs were beginning to penetrate the mainstream memory market at the time we started our design, which provided assurance that we would be able to buy them in volume from multiple suppliers at little or no price premium.

Although we eliminated the data path slices in the path from memory to the CPU, we still needed a data path from the memory to the peripheral component interconnect (PCI) I/O bus. When we started the design, it seemed clear that we would need multiple chips to accommodate the data path because of the 128-bit memory bus, 16-bit error-correcting code (ECC) bus, the 64-bit PCI bus, and their associated control pins. We planned to partition the design to fit into multiple 208-pin plastic quad flat pack (PQFP) packages. Our first design approach used two data path slices to connect the memory to the PCI bus and a third chip for the control logic.

After a preliminary feasibility investigation with two ASIC suppliers, we decided to combine all three chips into one chip, using emerging ball grid array (BGA) packaging technology. Although a single chip would be more expensive, the elimination of two chips and more than 100 interconnections promised a reduction in main logic board complexity and size, thus partially offsetting the additional cost of the single chip design. In addition, the performance would be slightly better because critical paths would not span multiple chips and the design process would be much easier and faster.

## System Overview

Figure 2 shows a system diagram for the cacheless design. Note that the memory array connects directly to the CPU. Figure 3 shows a configuration with an optional cache. In this configuration, a QuickSwitch bus separator isolates the CPU/cache bus from the memory bus. (A QuickSwitch is simply a low-impedance field effect transistor [FET] switch, typically several switches in one integrated circuit package, controlled by a logic input. In the connected state, it has an impedance of approximately 5 ohms, so it behaves almost like a perfect switch in this application.) From an architectural standpoint, it is not absolutely necessary to separate the buses when a cache is provided, but the bus separation substantially reduces the capacitance on the CPU/cache bus when the buses are disconnected, which speeds up CPU cache accesses significantly. As a side benefit, direct memory access (DMA) traffic to and from main memory can be completed without interfering with CPU/cache traffic. The arrangement shown in Figure 3 is used in the DIGITAL Personal Workstation, with the cache implemented as a removable module. With this design, a single motherboard type supports a variety of system configurations spanning a wide performance range.

The memory controller also supports a server configuration, shown in Figure 4. In the server configuration, as many as eight dual in-line memory module (DIMM) pairs can be implemented. To cope with the additional data bus capacitance from the additional DIMMs, the DIMM portion of the data bus is partitioned into four sections using QuickSwitch FET switches; only one section is connected to the main data bus at any one time.

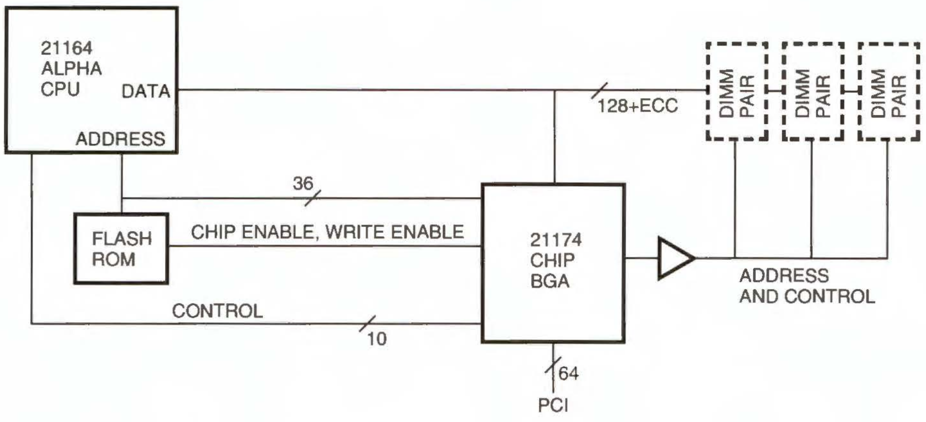


**Figure 2**
Cacheless Workstation Configuration

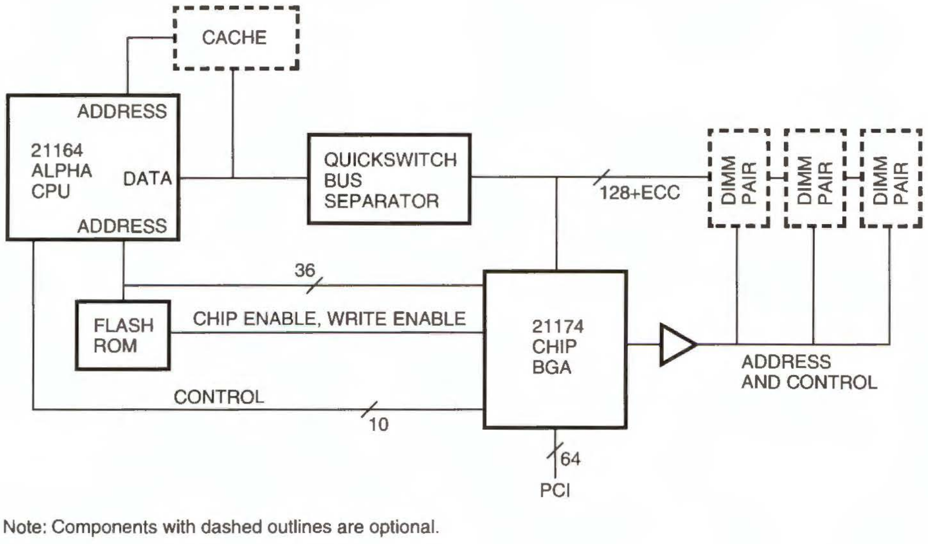


**Figure 3**
Workstation Configuration with Optional Cache

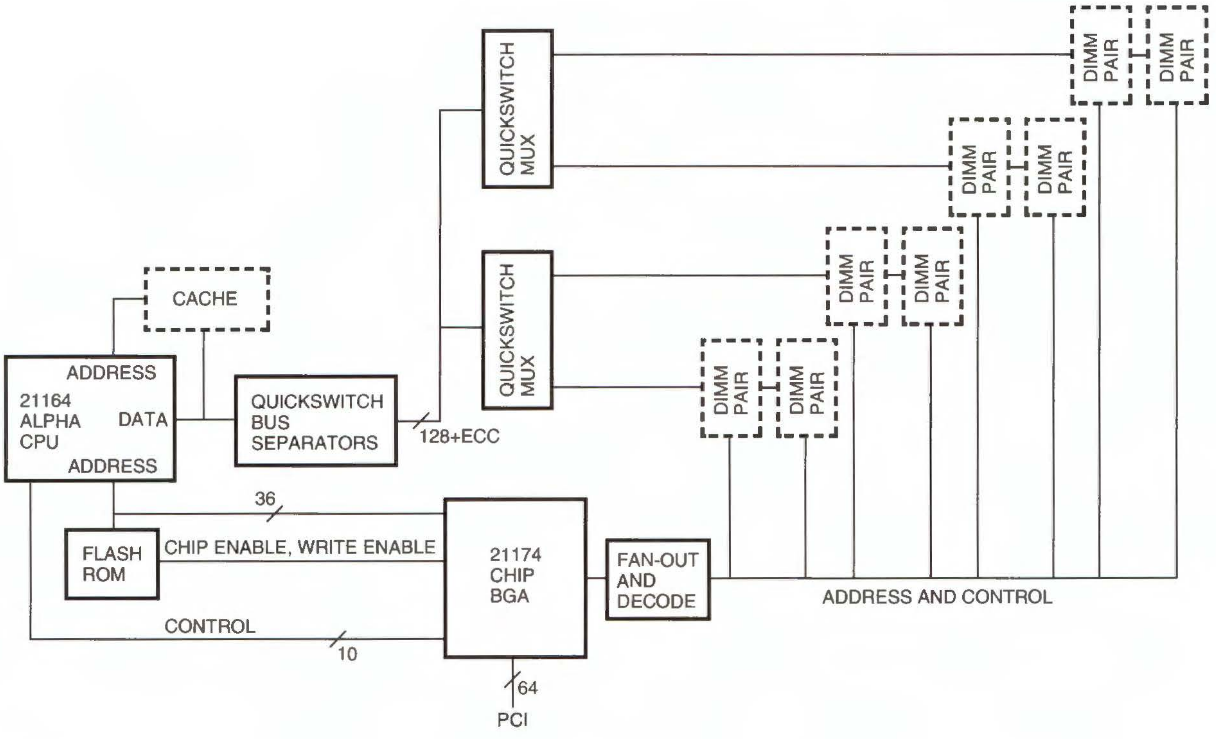


**Figure 4**
Server Configuration

In addition to the QuickSwitch bus separators, the server configuration requires a decoder to generate 16 separate chip select (CS) signals from the 6 CS signals provided by the 21174 chip.

## Data Bus Sequences

The 21164 CPU has a cache line size of 64 bytes, which corresponds to four cycles of the 128-bit data bus. For ease of implementation, the memory controller does all memory operations in groups of four data cycles. Figure 5 shows the basic memory read cycle; Figure 6 shows the basic memory write cycle.

To provide full support for all processor and DMA request types, as well as cache coherence requirements, the memory controller must implement a large variety of data transfer sequences. Table 1 shows the combinations of data source and destination and the transactions that use them. All memory and cache data transfers are 64-byte transfers, requiring four cycles of the 128-bit data bus. All I/O transfers are 32-byte transfers, requiring two cycles.

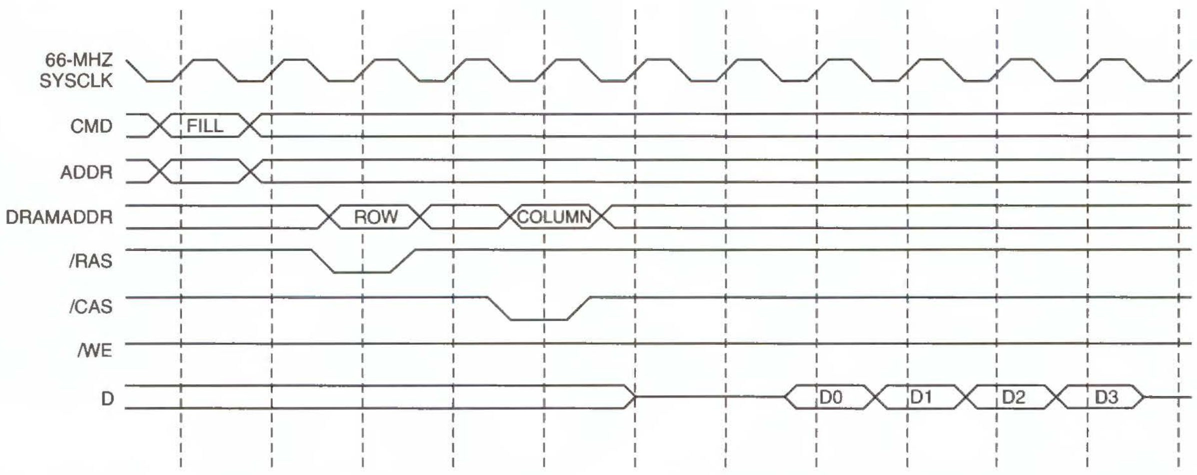


**Figure 5**
Basic Memory Read Timing

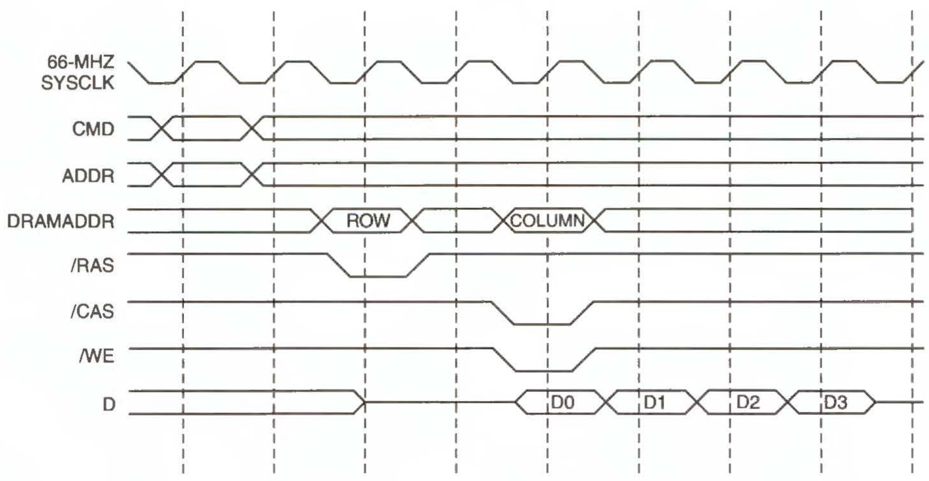


**Figure 6**
Basic Memory Write Timing

**Table 1**
Data Bus Transfer Types

| From \ To | 21164 CPU | Cache | 21174 ASIC | DRAM |
|---|---|---|---|---|
| 21164 CPU | | Private | I/O write | L2 victim |
| Cache | Private | | L3 victim<br>DMA read<br>DMA write[2] | |
| 21174 ASIC | I/O read<br>Fill[1] | Fill[1] | DMA read[1]<br>DMA write[1,2] | L3 victim<br>DMA write |
| DRAM | Fill | Fill | DMA read<br>DMA write[2] | |

Notes
1. Data from victim buffer.
2. Merge data.

The memory controller is designed to permit partially overlapped memory operations. The 21164 CPU can emit a second memory-fill request before the fill data from the first request has been returned. Figure 7 shows a timing sequence for two read commands to the same group of memory chips. Note that the second read command to the SDRAMs is issued while the data from the first read is on the data bus. The two read commands (four data cycles each) are completed with no dead cycles between the read data for the first read and the read data for the second read. For most other cases, the data cycles of adjacent transactions cannot be chained together without intervening bus cycles, because of various resource conflicts. For example, when the bus is driven from different SDRAM chips, at least one dead cycle between transactions is needed to avoid drive conflicts on the bus resulting from clock skews and delay variations between the two data sources.

## Memory Controller Components

Figure 8 shows a block diagram of the 21174 chip. Table 2 gives an overview of key ASIC metrics. A discussion of the entire chip is beyond the scope of this paper, but note that there are two major control blocks: the PCI controller manages the PCI interface, and the memory controller manages the memory subsystem and the CPU interface. The memory controller has six major sections, as shown in Figure 9. The detailed functions of these components are described next.

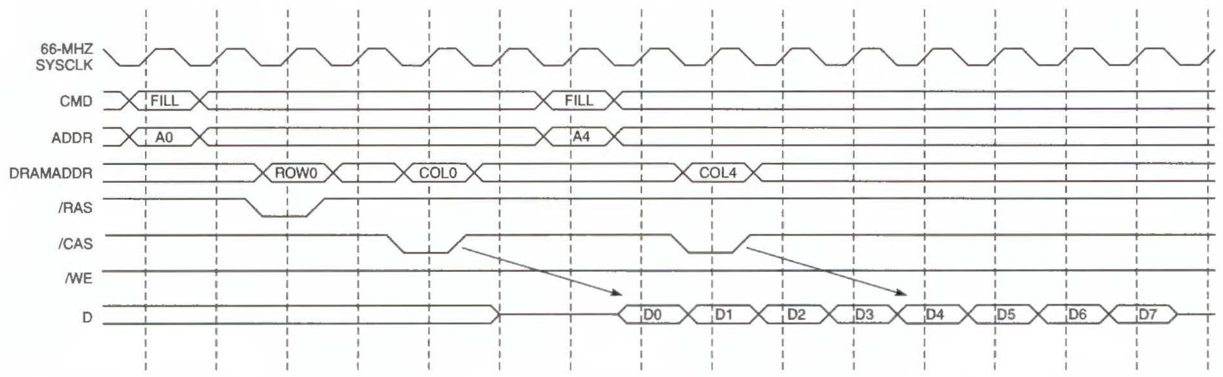


**Figure 7**
Pipelined Read Timing

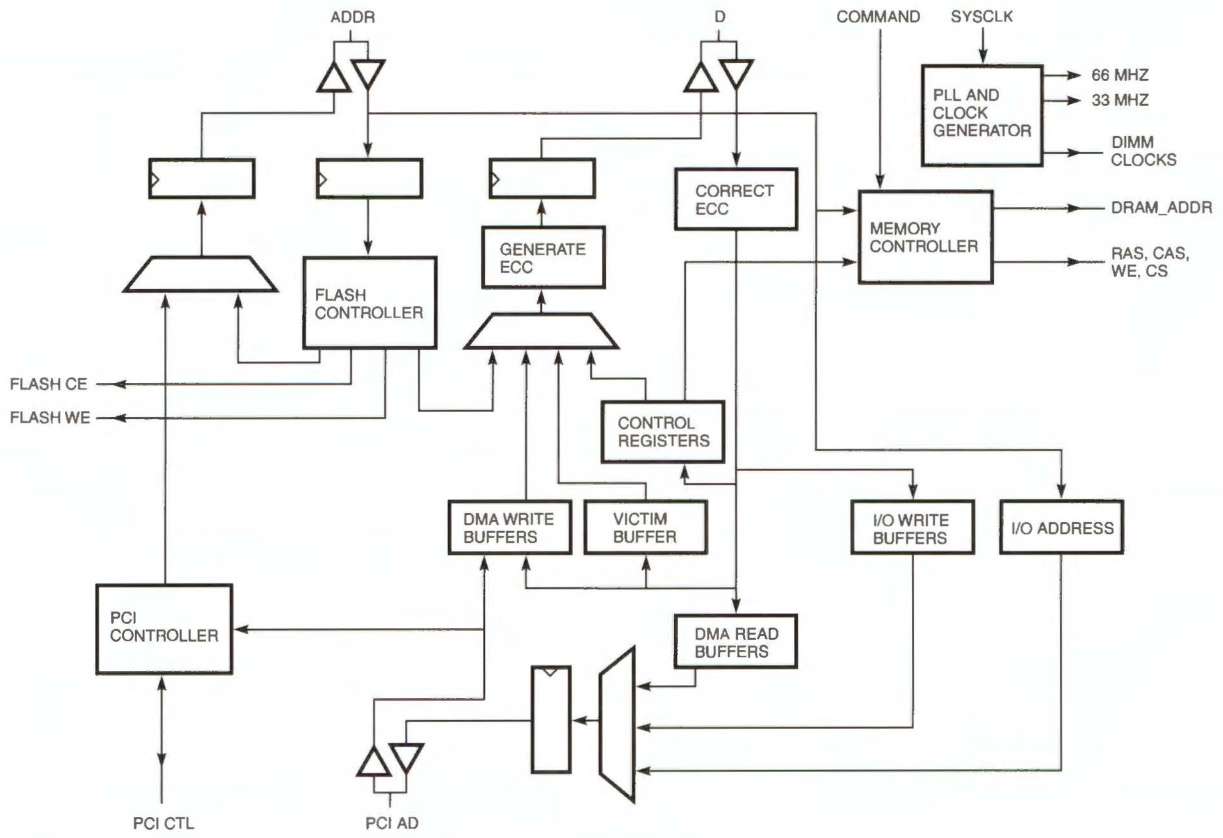


**Figure 8**
Block Diagram of the Logic Chip

### *Memory Sequencer*

The memory sequencer provides overall timing control for all portions of the 21174 chip except the PCI interface. To facilitate partial overlapping of memory transactions, the sequencer is implemented as two separate state machines, the main sequencer and the data cycle sequencer.

The main sequencer is responsible for overall transaction flow. It starts all transactions and issues all memory addresses and memory commands. Table 3 lists the transaction types supported by the main sequencer and the number of states associated with each type.

The data cycle sequencer is dispatched by the main sequencer and executes the four data cycles associated with each memory transfer. Although the data cycle sequencer is simple in concept, it must cope with a wide variety of special cases and special cycles. For example, in the case of reads and writes to flash read-only mem-

**Table 2**
21174 ASIC Summary

| Function | Core Logic |
|---|---|
| Data bus clock | 66 MHz |
| PCI clock | 33 MHz |
| Memory transfer size | 64 Bytes |
| Memory latency | 105 ns (7-1-1-1) |
| Hot row latency | 75 ns (5-1-1-1) |
| Data bus width | 128 bits + ECC |
| SDRAM support | 16 Mbit and 64 Mbit 3.3V |
| SDRAM CAS latency | 2 or 3 |
| SDRAM banks/chip | 2 or 4 |
| PCI bus width | 64 bits |
| Gate count | 250,000 gates |
| Pin count | 474 total (379 signal pins) |
| Package | Ceramic BGA |
| Process | .35-μm CMOS |

**Table 3**
Main Sequencer States

| Transaction Type | Number of States |
|---|---|
| Idle | 1 |
| Wait for idle | 2 |
| Cache fill | 4 |
| Cache fill from flash ROM | 4 |
| Cache fill from dummy memory | 2 |
| L3 cache victim write back | 3 |
| L2 cache victim write back | 2 |
| DMA read | 9 |
| DMA write | 14 |
| SDRAM refresh | 2 |
| SDRAM mode set | 2 |
| PCI read | 2 |
| CSR read | 3 |
| Flash ROM byte read | 1 |
| PCI or CSR write | 1 |
| Flash ROM byte write | 1 |
| Errors | 2 |
| Total states | 55 |

ory (ROM), stall cycles are needed, because of the slow access time and narrow data bus of the flash ROM. Table 4 gives the cycle types supported by the data cycle sequencer.

Some control signals from the memory sequencer are driven from the main sequencer, and some are driven from the data cycle sequencer. In addition, a number of control signals may be driven by either state machine. This is necessary because of pipelining considerations, as transactions are handed off from the main sequencer to the data cycle sequencer. For example, during a DMA write transaction, the DMA write data is selected on the internal data path multiplexer one cycle before it appears at the data pins, so the select signal for the internal multiplexer is driven by the main sequencer for the first data cycle of the DMA write and is then driven by the data cycle sequencer for three additional data cycles. The shared control signals are implemented separately in each state machine, and the outputs are simply ORed together.

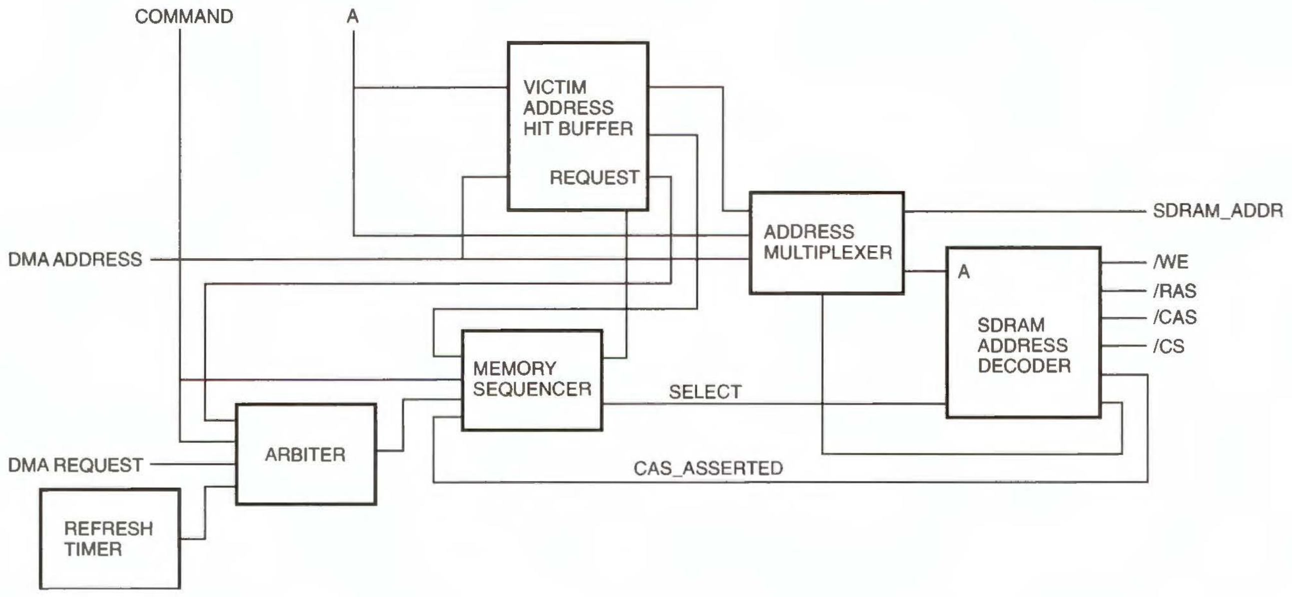


**Figure 9**
Block Diagram of the Memory Controller

**Table 4**
Data Cycle Sequencer States

| Data Cycle Type |
|---|
| Idle |
| Cache fill |
| Cache fill from victim buffer |
| Cache fill from flash ROM |
| Cache fill from dummy memory |
| L2 cache victim write back |
| L3 cache victim write back to 21174 ASIC |
| L3 cache victim write back from 21174 ASIC to memory |
| DMA read from memory |
| DMA read from L2 cache |
| DMA read from L3 cache |
| DMA read from victim buffer |
| DMA write to memory |
| Read merge data from memory for DMA write |
| Read merge data from L2 cache for DMA write |
| Read merge data from L3 cache for DMA write |
| Read merge data from victim buffer for DMA write |
| CSR read |
| PCI read |
| Flash ROM byte read |
| PCI or CSR write |
| Flash ROM byte write |

### *Address Multiplexer*

The address multiplexer accepts an address from the 21164 address bus, from a DMA transaction, or from the victim buffer and then selects a subset of the address bits to be driven to the multiplexed address pins of the SDRAMs. Figure 10 shows a block diagram of the address path. As with most other DRAMs, the SDRAM address pins are used first for an n-bit row address that selects one of $2^n$ rows, and later for an m-bit column address that selects one of the $2^m$ bits within that row. To minimize the delay through the address logic, the address bits are assigned to row and column addresses in a way that allows support of many different SDRAM chip types, always using the same row address selection and using only two possible selections of column addresses. Table 5 gives the address bit assignments.

To further accelerate the address decoding, the input to the address multiplexer is sent directly from the address receiver ahead of the receive register, and a significant portion of the address decoding and multiplexing is completed before the first clock.

### *SDRAM Address Decoder*

To achieve minimum latency, the address path of the memory controller is designed to send a row or column address to the SDRAMs as quickly as possible after the address is received from the CPU. This goal conflicts with the need to support a wide variety of DIMM sizes in a random mix, and the overall address path is extremely complex in fan-out and gate count.

The opening and closing of rows (row access strobe [RAS]) and the launch of transactions (column access strobe [CAS]) is controlled separately for each DIMM pair by one of eight DIMM pair controllers. Each DIMM pair controller, in turn, contains eight bank controllers to control the four possible banks within each of the two groups of memory chips on the DIMM pair, for a total of 64 bank controllers. Figure 11 shows the details.

The address path and the launch of memory transactions are controlled by four signals from the main sequencer: SELECT, SELECT_FOR_READ, SELECT_FOR_WRITE, and SELECT_FOR_RMW.

The SELECT signal directs the DIMM pair controllers to open the selected row. There are three basic cases. In the simplest case, the row is already open, and no further action is needed. In the second case, no row is open, and the DIMM pair controller must issue an activate command (or RAS) to the SDRAMs. In the most complex case, an incorrect row is open, and it

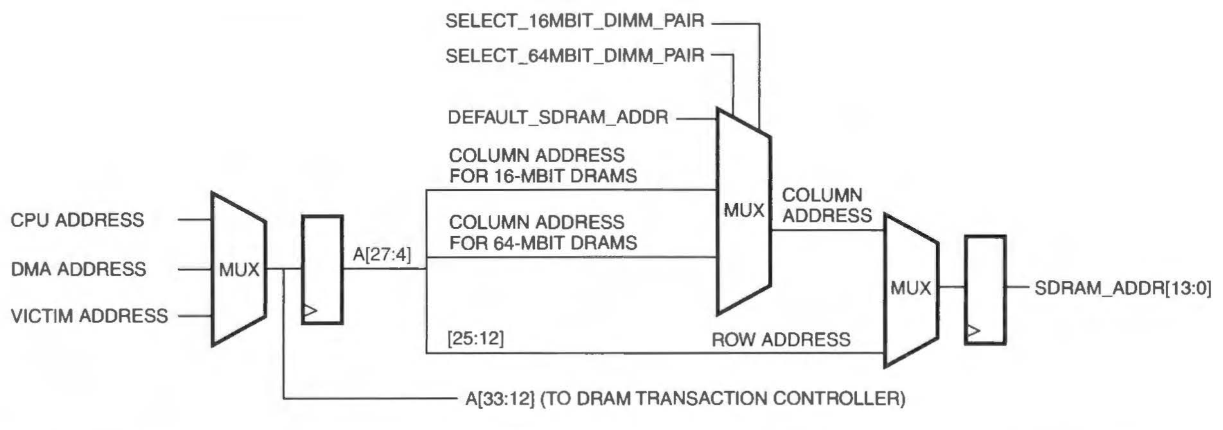


**Figure 10**
Address Multiplexer

**Table 5**
Row and Column Address Mapping

| SDRAM Chip Type | Banks | Number of Bits, Row/Column | Bank Select Bits | Row Bits | Column Bits[1] |
|---|---|---|---|---|---|
| 16M x 4 | 4 | 14/10 | 25,24 | 23:12 | 27:26,11:4 |
| 8M x 8 | 4 | 14/9 | 25,24 | 23:12 | 26,11:4 |
| 4M x 16 | 4 | 14/8 | 25,24 | 23:1 2 | 11:4 |
| 16M x 4 | 2 | 14/10 | 24 | 25,23:12 | 27:26,11:4 |
| 8M x 8 | 2 | 14/9 | 24 | 25,23:12 | 26,11:4 |
| 4M x 16 | 2 | 14/8 | 24 | 25,23:12 | 11:4 |
| 4M x 4 | 2 | 12/10 | 23 | 23:12 | 25:24,11:4 |
| 2M x 8 | 2 | 12/9 | 23 | 23:12 | 24,11:4 |
| 1M x 16 | 2 | 12/8 | 23 | 23:12 | 11:4 |

Note
1. All 10 or 12 column bits for each SDRAM size are sent to the SDRAMs. For x8 or x16 SDRAM chip configurations, one or two high-order column bits are ignored by the SDRAMs.

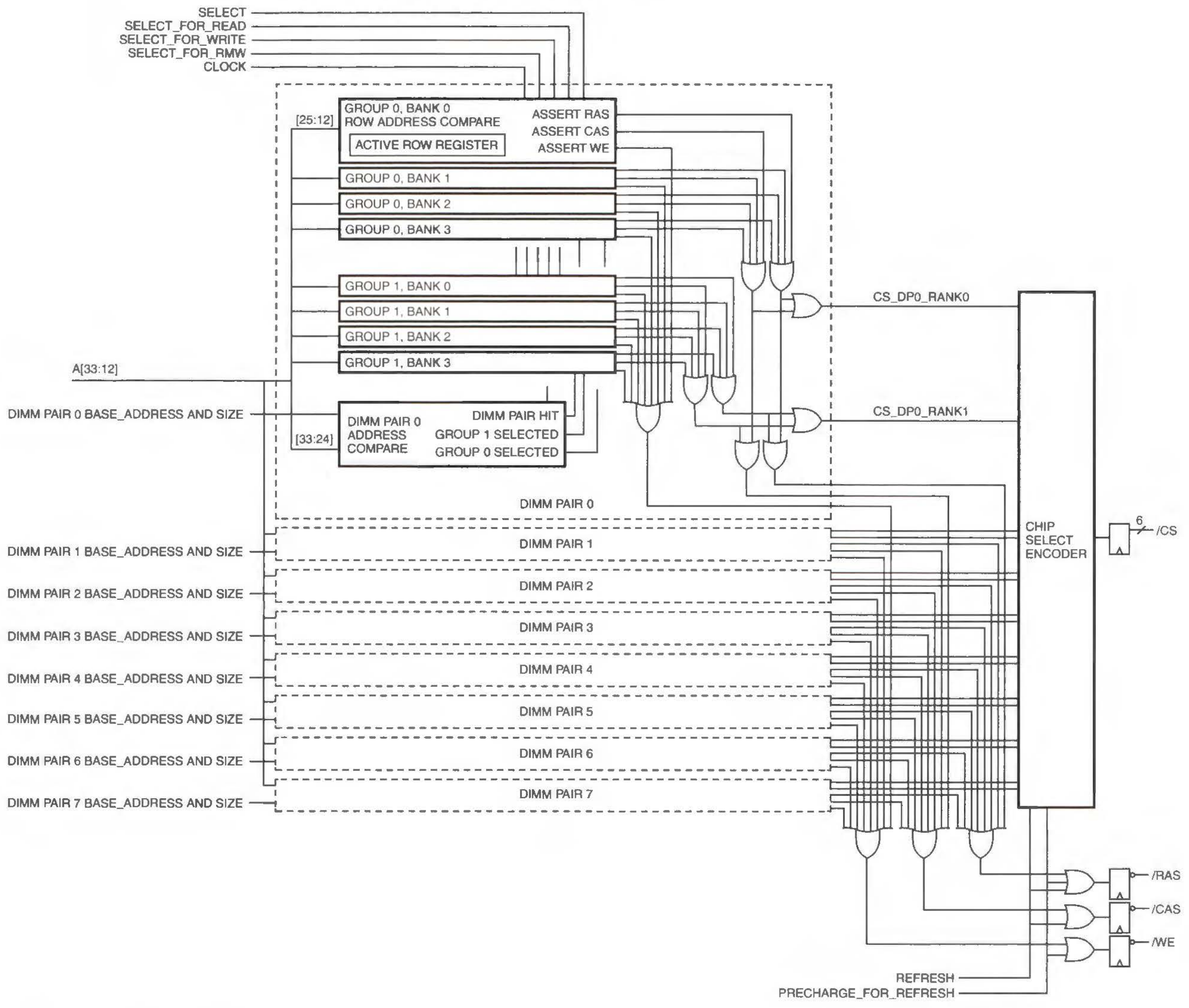


**Figure 11**
SDRAM Address Decoder

must be closed by a precharge command before the correct row can be opened. The main sequencer asserts SELECT continuously until a data transfer is started.

The DIMM pair controllers do not need to report the completion of the SELECT operation. They simply open the selected row as quickly as possible. However, if none of the DIMM pair controllers recognizes the address, a nonexistent memory status is signaled to the main sequencer.

The SELECT signal can be issued speculatively. There is no commitment to complete a data transfer to the selected group of SDRAMs. For example, on a DMA read, SELECT is issued at the same time that a cache probe is launched to the CPU. If the CPU subsequently provides the read data, no memory read is needed, and the main sequencer simply deasserts SELECT.

The main sequencer asserts a SELECT_FOR_READ, SELECT_FOR_WRITE, or SELECT_FOR_RMW signal when it is ready to start a data transfer. SELECT_FOR_RMW starts a normal read sequence, with the exception that the row hit predictor is ignored and the row is unconditionally left open at the end of the read, in anticipation that the next operation will be a write to the same address. (The subsequent write sequence would work properly even if the row were not left open, so this is strictly a performance optimization.) In response to a SELECT_FOR_x signal, the selected DIMM pair controller starts the specified transaction type as soon as possible. If necessary, the transaction start is delayed until the selected row has been opened. When the DIMM pair controller starts the data transaction (i.e., when it asserts CAS), it reports CAS_ASSERTED to the main sequencer.

The memory controller provides almost unlimited flexibility for the use of different DIMM types among the three DIMM pairs (eight DIMM pairs in the server configuration). Table 6 gives the fields in the DIMM pair control registers. The DIMM pair size, as well as some SDRAM parameters, can be set individually per DIMM pair through these register fields.

**Table 6**
DIMM Pair Control Register Fields

| Field | Use |
|---|---|
| ENABLE | Indicates that this bank is installed and usable |
| BASE_ADDRESS[33:24] | Defines starting address |
| SIZE[3:0] | Defines total size of DIMM pair, 16 Mbytes – 512 Mbytes |
| TWO_GROUPS | Indicates that DIMM pair has two groups of chips |
| 64MBIT | Selects 64-Mbit column mappings for this DIMM pair |
| 4BANK | Indicates that the SDRAM chips each contain four banks |

### *Hot Rows*

The SDRAMs have an interesting feature that makes it possible to improve the average latency of the main memory. They permit two (or four) hot rows in the two (or four) banks in each memory chip to remain active simultaneously. Because a workstation configuration has as many as three DIMM pairs, with either one or two separately accessed groups of chips per DIMM pair, as many as 24 rows can remain open within the memory system. (In the server configuration, the memory controller can support as many as 64 hot rows on eight DIMM pairs.)

The collection of hot rows can be regarded as a cache. This cache has unconventional characteristics, since the size and number of the hot rows vary with the type of SDRAM chip and with the number and type of memory DIMMs installed in the machine. In the maximum configuration, the cache consists of 24 cache lines (i.e., hot rows) with a line size of 16 kilobytes (KB), for a total cache size of 384 KB. Although the cache is not large, its bandwidth is 16 KB every 105 nanoseconds or 152 gigabytes per second. (The bus between this cache and main memory is 131,072 bits wide, not counting the 16,384 ECC bits!)

Table 7 gives the state bits associated with each bank within each group of chips on each DIMM pair in the memory system. To keep track of the hot rows, a row address register is provided for each bank. There are 64 copies of the state given in Table 5, although only 24 of them are usable in the workstation configuration. Prior to each memory access, the row address portion of the current memory address is compared against the selected row address register to determine whether the open row can be used or whether a new row must be opened. (Of course, if the ROW_ACTIVE bit is not set, a new row must always be opened.)

As with any cache, the hit rate is an important factor in determining the effectiveness of the hot row cache. If a memory reference hits on an open row, latency is reduced due to the faster access time of the open row. However, if the memory reference misses on an open row, the row must first be closed (i.e., a DRAM precharge cycle must be done) before another row can be accessed. In the past, some memory controllers

**Table 7**
State Bits for Each of 64 SDRAM Banks

| State | Use |
|---|---|
| ACTIVE_ROW[25:12] | The row currently in use, if any |
| ROW_ACTIVE | Indicates whether row is open |
| ROW_HIT_HISTORY[3:0] | Hit/miss state from last four accesses |
| TIMEOUT_CTR[3:0] | Tracks busy time after command is issued to bank |

were designed to keep the most recently used row open in one or more memory banks. This scheme provides some performance benefit on some programs. On most programs, however, the scheme hurts performance, because the hit rate is so low that the access time reduction on hits is overshadowed by the time needed to close the open rows on misses.

The memory controller uses predictors to guess whether the next access to a row will be a hit or a miss. If a hit is predicted on the next access, the row is left open at the end of the current access. If a miss is predicted, the row is closed (i.e., the memory bank is precharged), thus saving the subsequent time penalty for closing the row if the next access is, in fact, a miss. A separate predictor is used for each potential open row. There are 64 predictors in all, although only 24 can be used in the workstation configuration. Each predictor records the hit/miss result for the previous four accesses to the associated bank. If an access to the bank goes to the same row as the previous access to the same bank, it is recorded as a hit (whether or not the row was kept open); otherwise, it is recorded as a miss. The predictor then uses the four-bit history to predict whether the next access will be a hit or a miss. If the previous four accesses are all hits, it predicts a hit. If the previous four accesses are all misses, it predicts a miss. Since the optimum policy is not obvious for the other 14 cases, a software-controlled 16-bit precharge policy register is provided to define the policy for each of the 16 possible cases. Software can set this register to specify the desired policy or can disable the hot row scheme altogether by setting the register to zeros.

The precharge policy register is set to 1110 1000 1000 0000 upon initialization, which keeps the row open whenever three of the preceding four accesses are row hits. To date, we have tested only with this setting and with the all-zeros and all-ones settings, and we have not yet determined whether the default setting is optimal. The adaptive hot row feature provides a 23-percent improvement in measured (i.e., best-case) memory latency and a 7-percent improvement in measured bandwidth against the STREAM benchmark, as reflected in Figure 1. Testing of the adaptive hot row feature shows an average performance improvement of 2.0 percent on SPEC95fp base and an average of 0.9 percent on SPEC95int base.

The level of improvement varies considerably over the individual benchmarks in the SPEC suite, as shown in Figure 12, with improvements ranging from –0.6 percent to +5.5 percent.

For completeness, we also tested with the all-ones case, i.e., with the current row left open at the end of each access. This scheme resulted in a 5-percent performance loss on SPEC95fp base and a 2-percent performance loss on SPEC95int base.

The row hit predictor is not as useful for configurations with an L3 cache, because the 21164 interface enforces a substantial minimum memory latency for memory reads to allow for the worst-case completion time for an unrelated 21164 cache access. In most cases, this minimum latency erases the latency improvement obtained by hitting on an open row.

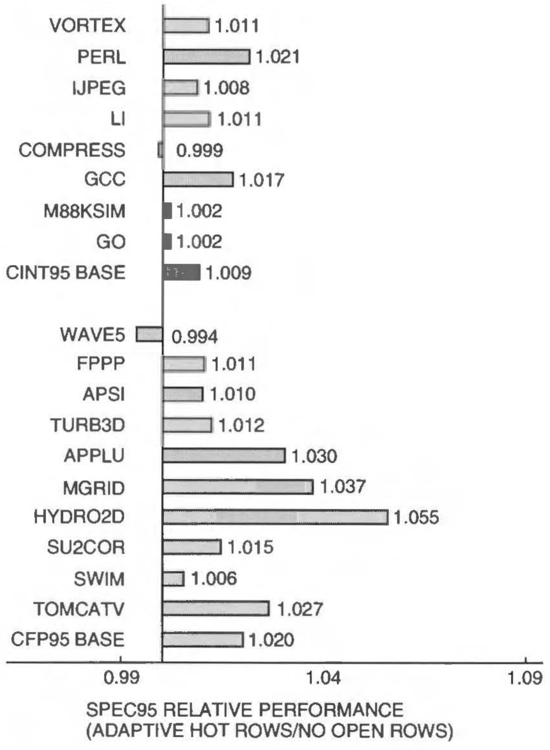


**Figure 12**
Performance Benefit of Adaptive Hot Rows

### *Victim Buffer*

The 21174 chip has a two-entry victim buffer to capture victims from the 21164 CPU and hold them for subsequent write back to memory. A two-entry buffer is provided so that write back can be deferred in favor of processing fill requests. Normally, fill requests have priority over victim write backs. When the second victim buffer entry is allocated, the victim write-back priority is elevated to ensure that at least one of the victims is immediately retired, thus avoiding a potential buffer overrun. The victim buffers have address comparators to check the victim address against all fills and DMA reads and writes. On a victim buffer address hit, read data or merge data is supplied directly from the victim buffer: the data is passed over the external data bus to avoid the use of an additional data path within the chip.

### *Arbiter*

The memory sequencer must process requests from several different sources. The arbiter selects which request to process next. The arbiter must serve two conflicting goals: (1) priority service for latency-sensitive requests, and (2) avoidance of lockouts and deadlocks. Table 8

**Table 8**
Arbitration Priorities

| Arbitration Priority | If Request Register is Latched | If Request Register is NOT Latched |
|---|---|---|
| Highest | Set SDRAM Mode | CPU request |
| . | Refresh (if overdue) | Set SDRAM mode |
| . | Victim write back (if both buffers are full) | Refresh (if overdue) |
| . | DMA read | Victim write back (if both buffers are full) |
| . | DMA address translation buffer fill | DMA read |
| . | DMA write (if both buffers are full) | DMA address translation buffer fill |
| . | PCI read completion | DMA write (if both buffers are full) |
| . | CPU request | PCI read completion |
| . | Victim write back | Victim write back |
| . | DMA write | DMA write |
| Lowest | Refresh | Refresh |

gives the arbitration priorities. Normally, CPU cache misses are treated as the highest priority requests. However, the arbiter avoids excessive latency on other requests by processing requests in batches. When a request is first accepted, the existing set of requests (usually just one) is captured in a latch. All requests within that group are serviced before any new request is considered. During the servicing of the latched request(s), several new service requests may arrive. When the last latched request is serviced, the latch is opened and a new batch of requests is captured. For a heavy request load, this process approximates a round-robin priority scheme but is much simpler to implement.

Normally, DMA write requests, L3 cache victims, and refresh requests are low-priority requests. However, a second request of any of these types can become pending before the first one has been serviced. In this case, the priority for the request is increased to avoid unnecessary blockage or overruns.

### Refresh Controller

A simple, free-running refresh timer provides a refresh request at a programmable interval, typically every 15 microseconds. The refresh request is treated as a low-priority request until half the refresh interval has expired. At that time, the refresh controller asserts a high-priority refresh request that is serviced before all other memory requests to ensure that refreshes are not blocked indefinitely by other traffic. When the arbiter grants the refresh request, the main sequencer performs a refresh operation on all banks simultaneously.

### Cache Coherence

The Alpha I/O architecture requires that all DMA operations be fully cache coherent. Consequently, every DMA access requires a cache lookup in the 21164 microprocessor's level 2 (L2) write-back cache and/or in the external L3 cache, which is controlled by the 21164. In general, the memory controller starts the row access portion of the memory access at the same time that it requests the cache lookup from the CPU, in anticipation that the lookup will miss in the cache. For DMA reads, the 21164 microprocessor may return read data, which is then used to satisfy the read request. For DMA writes, several flows are possible, as shown in Figure 13.

If the DMA write covers an entire cache line, the memory controller requests that the CPU invalidate the associated entry, and the DMA data is then written directly to memory. If the write covers a partial cache line, a cache lookup is performed; on a hit, the cache data is merged into the write buffer before the data is written to memory. If the cache misses on a partial line write, two outcomes are possible: If each 128-bit memory cycle is either completely written or completely unchanged, the data is written to memory directly, and the write to the unchanged portions is suppressed using the data mask (DQM) signal on the memory DIMMs. If there are partially written 128-bit portions, the full cache line is read from memory and the appropriate portions are merged with the DMA write data.

### Cache Flushing

The memory controller provides two novel features for assistance in flushing the write-back cache. These features are intended to be used in power-managed configurations where it may be desirable to shut down the processor and cache frequently, without losing memory contents and cache coherence. If the cache contains dirty lines at the time of shutdown, these lines must be forced back to memory before the cache can be disabled. The 21164 microprocessor provides no direct assistance for flushing its internal cache or the external L3 cache, so it is generally necessary to read a large block of memory to flush the cache. The mem-

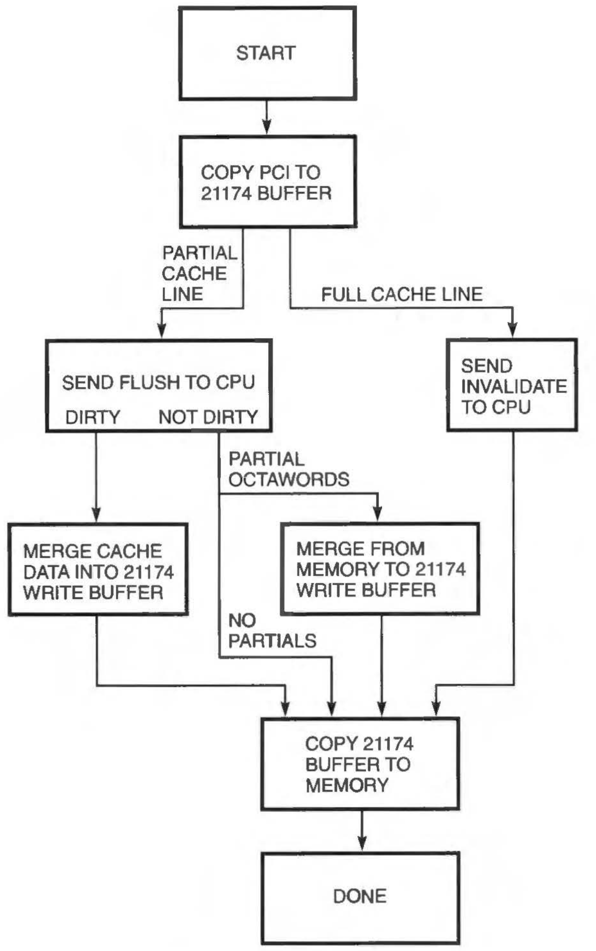


**Figure 13**
DMA Write Flows

ory controller provides a large block of ROM (which always reads as zeros) for this purpose. Of course, no memory array is needed to support this feature, so the implementation complexity is negligible.

In addition to the fake memory, the memory controller provides a cache valid map that can make cache flushing faster when the cache is lightly used. The cache valid map is a 32-bit register, in which each bit indicates whether a particular section of the cache has been (partially) filled. When a cache fill request is processed, the memory controller sets the corresponding bit in the cache map register. The bits of the register can be individually reset under program control. To use this feature, the cache must first be completely flushed and the register reset to zeros. Thereafter, the cache map register reflects which sections of the cache can potentially contain dirty data. During a subsequent cache flush, the flush algorithm can skip over those sections of the cache whose corresponding bits in the cache map register have not been set. This feature is useful primarily for cache shutdowns that occur between keystrokes, i.e., when the system is nearly idle but has frequent, short bursts of activity to service keyboard input, clock interrupts, modem receive data, or similar workloads.

## Conclusion

The 21174 chip provides a cost-effective core logic subsystem for a high-performance workstation. The use of SDRAMs, high-pin-count BGA packaging, and hot rows make possible a new level of memory performance, without the need for wide memory buses or a complex memory data path.

## Acknowledgments

I would like to express my appreciation to Jim Delmonico, Jeff Forsythe, Frank Fuentes, Paul Hill, Kenny House, Mark Kelley, Yong Oh, Rob Orlov, Rajen Ramchandani, Don Rice, Dean Sovie, and Ken Weiss, who gave a year of their lives to implement and simulate the 21174 chip and who made it work on the first pass! I would also like to thank Lance Berc, who provided most of the latency data, and Greg Devereau, Arlens Barosy, and John Henning, who provided other performance measurements.

## References

1. S. Perl and R. Sites, "Studies of Windows NT Performance using Dynamic Execution Traces," *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation* (1996).

2. J. Edmonson et al., "Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor," *Digital Technical Journal,* vol. 7 no. 1 (1995): 119–135.

3. *Digital Semiconductor 21164 (366-MHz Through 433 MHz) Alpha Microprocessor* (Maynard, Mass.: Digital Equipment Corporation, Order No. EC-QP99A-TE, 1996).

4. J. McCalpin, "STREAM: Measuring Sustainable Memory Bandwidth in High Performance Computers," http://www.cs.virginia.edu/stream/ (Charlottesville, Virg.: University of Virginia, 1997).

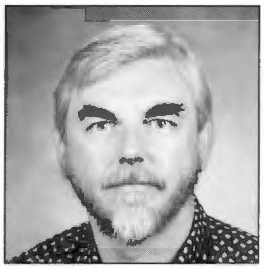
5. L. Berc, "Memory Latency," DIGITAL internal Web pages, September 1996.

6. J. Zurawski, J. Murray, and P. Lemmon, "The Design and Verification of the AlphaStation 600 5-series Workstation," *Digital Technical Journal,* vol. 7 no. 1 (1995): 89–99.

7. D. Conroy, "Cacheless Ev5," DIGITAL internal document.

8. *KM44S4020B/KMS4021B CMOS SDRAM Data Sheet* (Seoul, Korea: Samsung Electronics Company, Ltd., 1996).

## Biography



**Reinhard C. Schumann**
Until recently, Reinhard Schumann was the architect for value-line Alpha workstations. In that capacity, he provided technical leadership for the design of the 21174-based DIGITAL Personal Workstation a-series and au-series and led the design effort for the 21174 ASIC. During his 18-year tenure at DIGITAL, he contributed to the design of many other products and was awarded four patents. He holds B.S. and M.S. degrees in computer science from Cornell University and has published two papers on integrated circuit design. He is now employed at Avici Systems.

## Further Readings

The *Digital Technical Journal* is a refereed, quarterly publication of papers that explore the foundations of DIGITAL's products and technologies. *Journal* content is selected by the Journal Advisory Board, and papers are written by DIGITAL engineers and engineering partners. Engineers who would like to contribute a paper to the *Journal* should contact the managing editor, Jane Blake, at Jane.Blake@digital.com.

Topics covered in previous issues of the *Digital Technical Journal* are as follows:

**FX!32 Emulation and Translation/Visual Fortran/ MEMORY CHANNEL 2 Interconnect/ObjectBroker Security/StrongARM Microprocessor**
Vol. 9, No. 1, 1997, EC-N7963-18

**AlphaServer 4100 System/Oracle and Sybase Database Products for VLM/Instruction Execution on Alpha Processors**
Vol. 8, No. 4, 1997, EC-N7629-18

**Internet Protocol V.6/Preservation of Historical Computer Systems/Fortran for Parallel Computing/ Server Performance Evaluation and Optimization/ Internet Collaboration Software**
Vol. 8, No. 3, 1996, EC-N7285-18

**Spiralog Log-structured File System/OpenVMS for 64-bit Addressable Virtual Memory/High-performance Message Passing for Clusters/Speech Recognition Software**
Vol. 8, No. 2, 1996, EY-N6992-18

**DIGITAL UNIX Clusters/Object Modification Tools/eXcursion for Windows Operating Systems/ Network Directory Services**
Vol. 8, No. 1, 1996, EY-U025E-TJ

**Audio and Video Technologies/UNIX Available Servers/Real-time Debugging Tools**
Vol. 7, No. 4, 1995, EY-U002E-TJ

**High Performance Fortran in Parallel Environments/ Sequoia 2000 Research**
Vol. 7, No. 3, 1995, EY-T838E-TJ
(Available only on the Internet)

**Graphical Software Development/Systems Engineering**
Vol. 7, No. 2, 1995, EY-U001E-TJ

**Database Integration/Alpha Servers & Workstations/ Alpha 21164 CPU**
Vol. 7, No. 1, 1995, EY-T135E-TJ
(Available only on the Internet)

**RAID Array Controllers/Workflow Models/PC LAN and System Management Tools**
Vol. 6, No. 4, Fall 1994, EY-T118E-TJ

**AlphaServer Multiprocessing Systems/DEC OSF/1 Symmetric Multiprocessing/Scientific Computing Optimization for Alpha**
Vol. 6, No. 3, Summer 1994, EY-S799E-TJ

**Alpha AXP Partners—Cray, Raytheon, Kubota/ DECchip 21071/21072 PCI Chip Sets/DLT2000 Tape Drive**
Vol. 6, No. 2, Spring 1994, EY-F947E-TJ

**High-performance Networking/OpenVMS AXP System Software/Alpha AXP PC Hardware**
Vol. 6, No. 1, Winter 1994, EY-Q011E-TJ

**Software Process and Quality**
Vol. 5, No. 4, Fall 1993, EY-P920E-DP

**Product Internationalization**
Vol. 5, No. 3, Summer 1993, EY-P986E-DP

**Multimedia/Application Control**
Vol. 5, No. 2, Spring 1993, EY-P963E-DP

**DECnet Open Networking**
Vol. 5, No. 1, Winter 1993, EY-M770E-DP

**Alpha AXP Architecture and Systems**
Vol. 4, No. 4, Special Issue 1992, EY-J886E-DP

**NVAX-microprocessor VAX Systems**
Vol. 4, No. 3, Summer 1992, EY-J884E-DP

**Semiconductor Technologies**
Vol. 4, No. 2, Spring 1992, EY-L521E-DP

**PATHWORKS: PC Integration Software**
Vol. 4, No. 1, Winter 1992, EY-J825E-DP

**Image Processing, Video Terminals, and Printer Technologies**
Vol. 3, No. 4, Fall 1991, EY-H889E-DP

**Availability in VAXcluster Systems/Network Performance and Adapters**
Vol. 3, No. 3, Summer 1991, EY-H890E-DP

**Fiber Distributed Data Interface**
Vol. 3, No. 2, Spring 1991, EY-H876E-DP

**Transaction Processing, Databases, and Fault-tolerant Systems**
Vol. 3, No. 1, Winter 1991, EY-F588E-DP

**VAX 9000 Series**
Vol. 2, No. 4, Fall 1990, EY-E762E-DP

**DECwindows Program**
Vol. 2, No. 3, Summer 1990, EY-E756E-DP

**VAX 6000 Model 400 System**
Vol. 2, No. 2, Spring 1990, EY-C197E-DP

**Compound Document Architecture**
Vol. 2, No. 1, Winter 1990, EY-C196E-DP

## Technical Publications by DIGITAL Authors

N. Anderson and D. Manley, "A Matrix Extension of Winograd's Inner Product Algorithm," *Theoretical Computer Science 131* (1994).

A. Barbieri, "Benefits Applications and Data Analysis Techniques for Linewidth Multilevel Experimental Design," *Proceedings of the International Society of Photo-Optical Instrumentation Engineers (SPIE): Metrology, Inspection, and Process Control for Microlithography X* (March 1996).

G. Bouchard and P. Bannon, "Design Objective of the 0.35-micron Alpha 21164 Microprocessor," *1996 HOT Chips VIII Symposium* (August 1996).

C. Brench, "Experiences in EMI Modeling for Product Design," *IEEE International Symposium on EMC* (August 1996).

A. Charny, "Time Scale Analysis and Scalability Issues for Explicit Rate Allocation in ATM Networks," *IEEE/ACM Transactions on Networking* (August 1996).

K. Coar, "Converting an AdvFS Filesystem to UFS," *Digital Systems Report* (January/February 1996).

R. Collica, J. Ramirez, and W. Taam, "Process Monitoring in Integrated Circuit Fabrication Using Both Yield and Spatial Statistics," *Quality and Reliability Engineering International,* vol. 12 (1996).

S. DiPirro, "64-Bit Addressing in OpenVMS 7.0," *DIGITAL Age* (December 1995).

S. DiPirro, "Managing 64-Bit OpenVMS Systems," *Digital Systems Journal* (September/October 1995).

S. DiPirro, "The OpenVMS Alpha System-Code Debugger," *Digital Systems Report* (January/February 1996).

R. Dixson, J. Schneir, T. McWaid, N. Sullivan, V. Tsai, S. Zaidi, and S. Brueck, "Toward Accurate Linewidth Metrology Using Atomic Force Microscopy and Tip Characterization," *Proceedings of the International Society of Photo-Optical Instrumentation Engineers (SPIE): Metrology, Inspection, and Process Control for Microlithography X* (March 1996).

R. Dixson, N. Sullivan, J. Schneir, T. McWaid, V. Tsai, J. Prochazka, and M. Young, "Measurement of a CD and Sidewall Angle Artifact with Two Dimensional CD AFM Metrology," *Proceedings of the International Society of Photo-Optical Instrumentation Engineers (SPIE): Metrology, Inspection, and Process Control for Microlithography X* (March 1996).

A. Elsherbeni, O. Ramahi, and C. Smith, "FDTD Analysis of New Types of Microstrip Transmission Lines for High Frequency Applications," *Progress in Electromagnetics Research Symposium (PIERS) Proceedings* (July 1996).

R. Frankovic, G. Bernstein, and J. Clement, "Duty Cycle and Frequency Effects of Pulsed-DC Currents on Electromigration-Induced Stress in Al Interconnects," *Materials Research Society Symposium Proceedings Volume 428: Materials Reliability in Microelectronics VI* (April 1996).

C. Gianos and D. Hobson, "Design Considerations for Digital's PowerStorm Graphics Processor," *IBM Journal of Research and Development* (July 1996).

B. Gonzalez, S. Knecht, H. Handy, and J. Ramirez, "The Effect of Ultrasonic Frequency on Fine Pitch Aluminum Wedge Wirebond," *Proceedings of the 46th Electronic Components and Technology Conference* (May 1996).

R. Hellweg, Jr., H. Pei, and L. Wittig, "Precision of a New Method to Measure Fan Structure-Bourne Vibration," *Proceedings of Internoise 96* (July 30–August 2, 1996).

K. Hornbach, "Competing by Business Design—The Reshaping of the Computer Industry," *Long Range Planning,* vol. 29, no. 5 (1996).

S. Knecht, B. Gonzalez, and K. Sieber, "Fusing Current of Short Aluminum Bond Wire," *Fifth InterSociety Conference on Thermal Phenomena in Electronic Systems (I-THERM V)* (May 29–June 1, 1996).

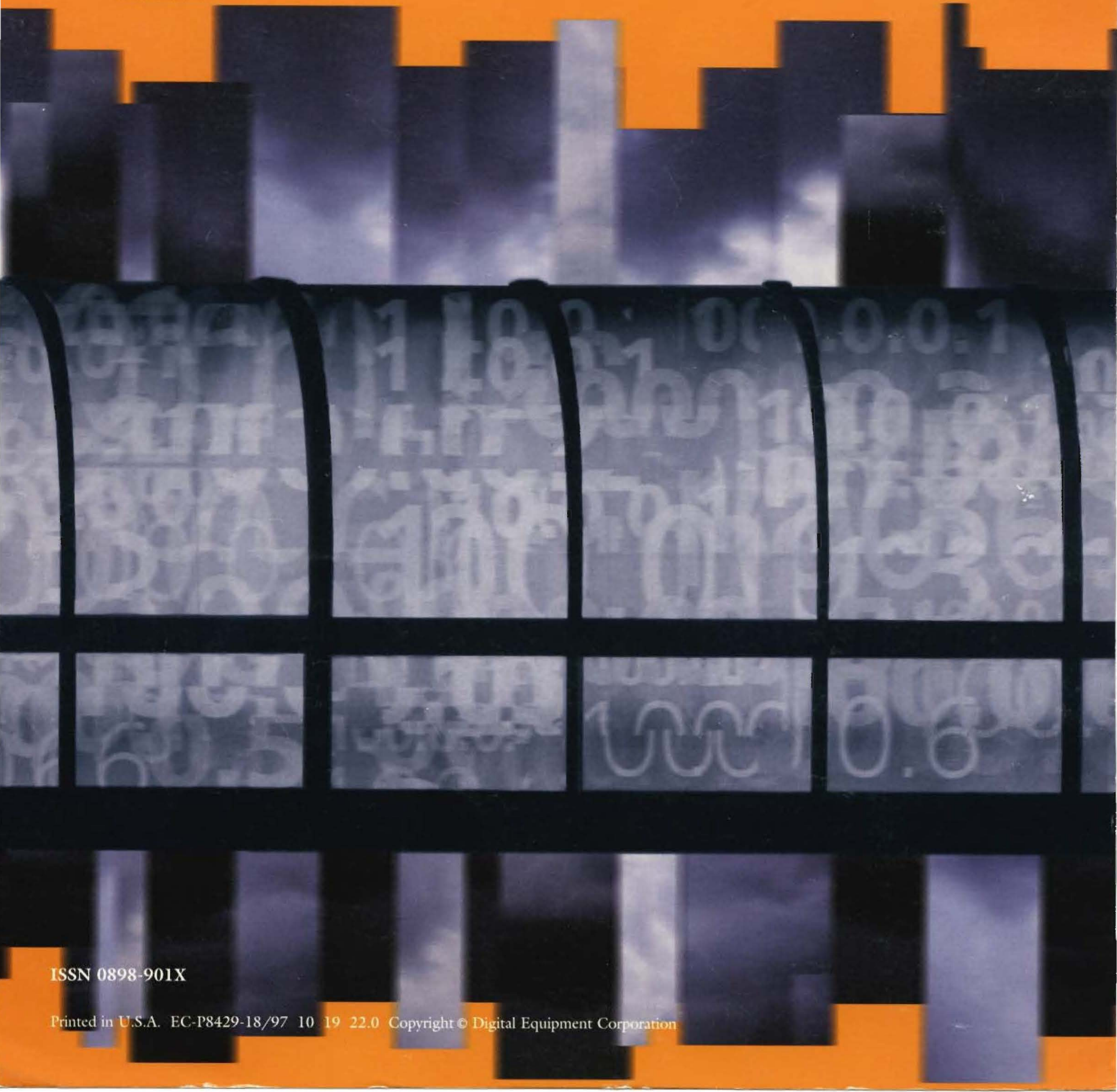
J. Kowaleski, G. Wolrich, T. Fischer, R. Dupcak, P. Kroesen, T. Pham, and A. Oelsin, "A Dual-Execution Pipelined Floating-Point CMOS Processor," *1996 IEEE International Solid-State Circuits Conference, Digest of Technical Papers* (February 1996).

K. Mistry, "New Hot Carrier Failure Criterion for *p*-Channel Transistors Based on Transistor Leakage Currents," *Solid-State Electronics,* vol. 39, no. 11 (1996).

K. Orvek, S. Dass, L. Gruber, S. Dumford, M. Piasecki, G. Pollard, and I. Fink, "Migrating Deep-UV Lithography to the 0.25-μm Regime: Issues and Outlook," *Proceedings of the International Society of Photo-Optical Instrumentation Engineers (SPIE): Optical Microlithography IX* (March 1996).

M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, and G. Zelesnik, "Abstractions for Software Architecture and Tools to Support Them," *IEEE Transactions on Software Engineering* (April 1995).

R. Woods, "How Emulators Do Their Work," *UNIX Review* (October 1995).



ISSN 0898-901X