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.

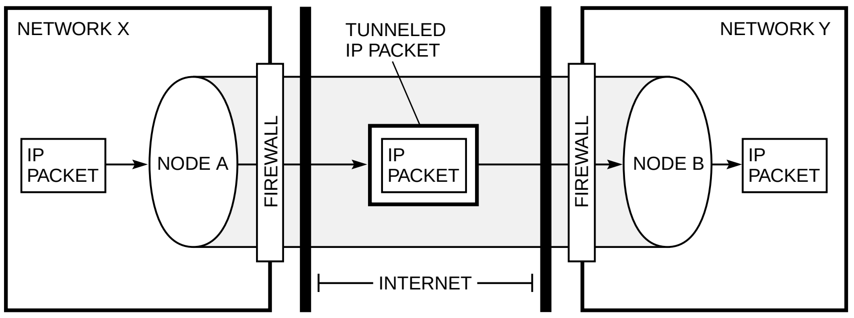


Figure 1
A Secure Tunnel in Operation

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.

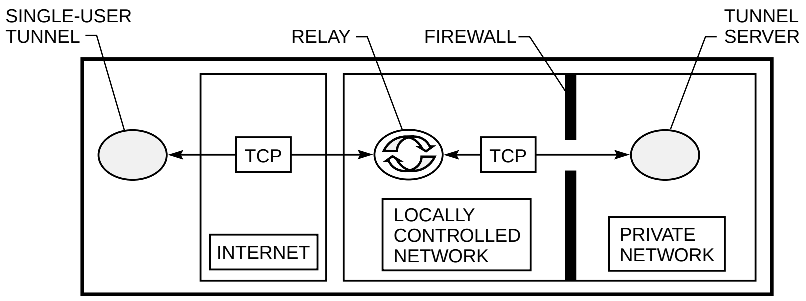


Figure 2
Tunnel with Intelligent Relay

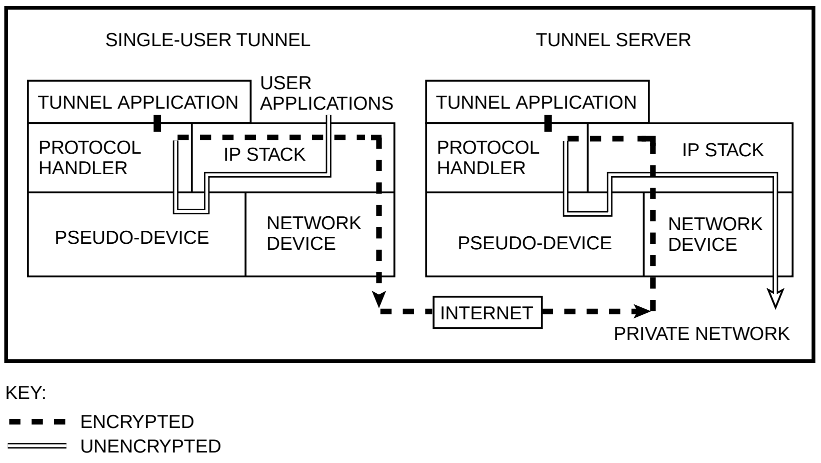


Figure 3
System Components and Data Flow

The Tunnel Application

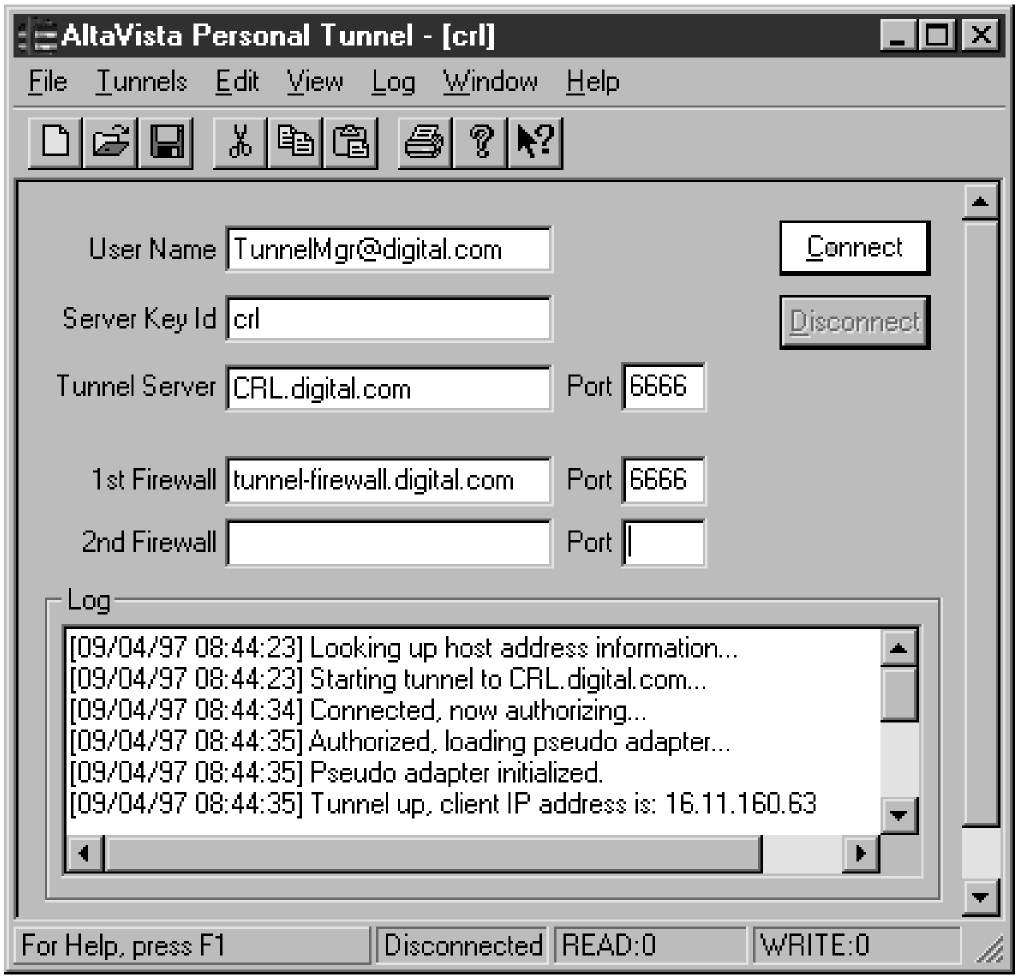
The primary function of the tunnel application is to present a user interface (UI). Although each instantiation of the user interface is slightly different, the function of the application remains the same. The AltaVista Personal Tunnel '97, a single-user configuration, offers a straightforward graphical 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
Log
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

```
[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
```

For Help, press F1 | 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

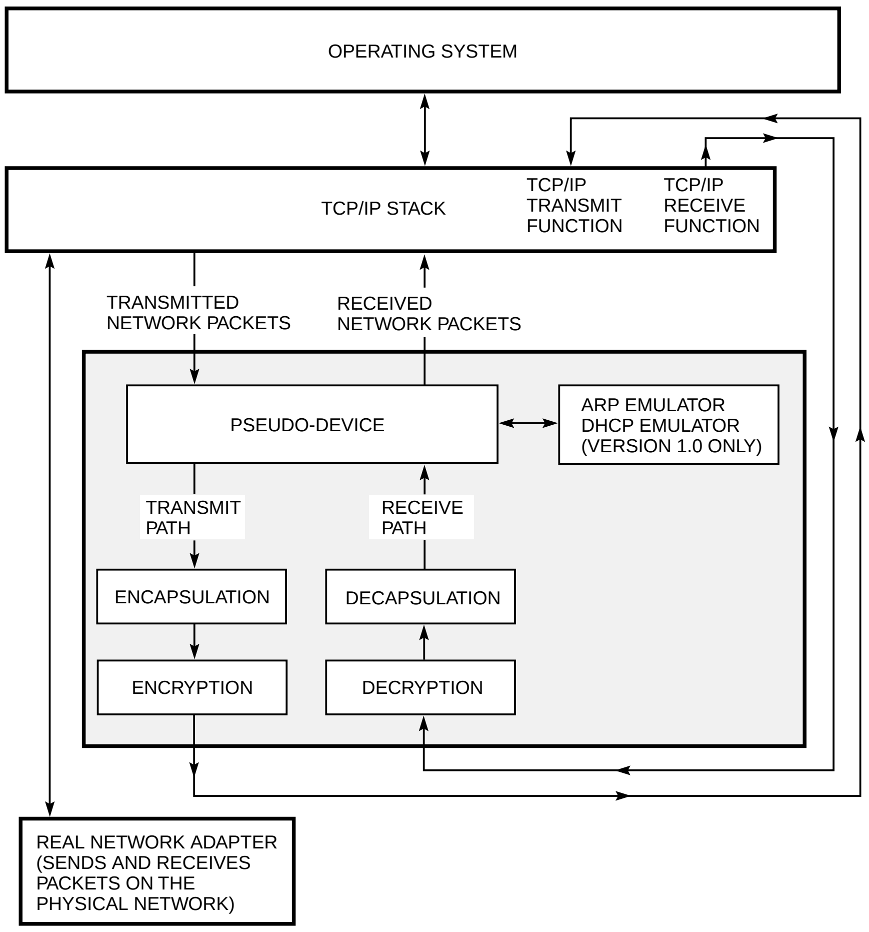


Figure 6
Control Flow in the Windows Pseudo-Device

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

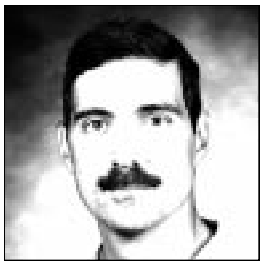
We owe thanks to many people for the success of the AltaVista Tunnel project. Virgil Champlin wrote the IP-in-TCP tunneling code on which we based our current UNIX pseudo-device. Mitch Lichtenberg provided help and guidance concerning all issues related to the Windows operating system. Wick Nichols contributed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