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.

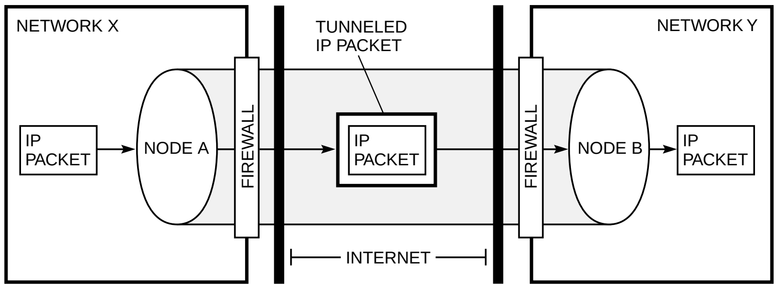


Figure 1
A Secure Tunnel in Operation

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.

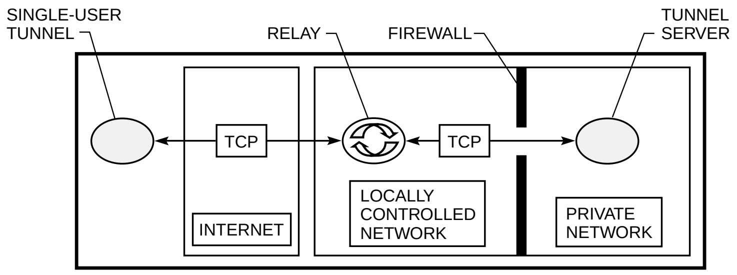


Figure 2
Tunnel with Intelligent Relay

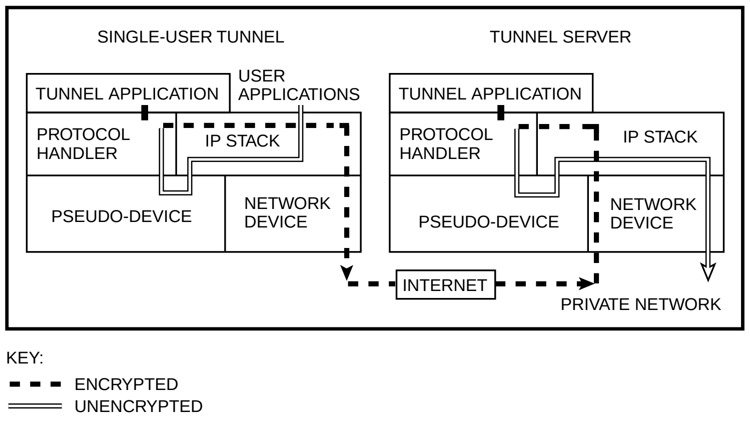


Figure 3
System Components and Data Flow

The Tunnel Application

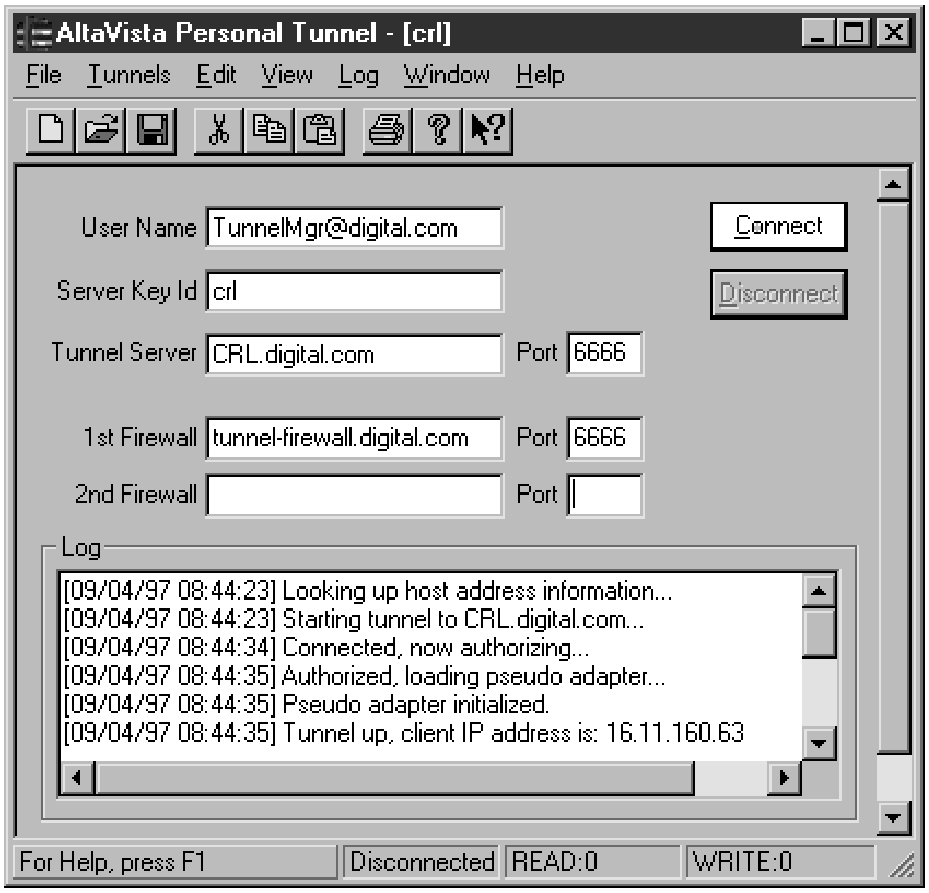
The primary function of the tunnel application is to present a user interface (UI). Although each instantiation of the user interface is slightly different, the function of the application remains the same. The AltaVista Personal Tunnel '97, a single-user configuration, offers a straightforward graphical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

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
```

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

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.