Moat: a Virtual Private Network Appliance and Services Platform

John S. Denker, Steven M. Bellovin – AT&T Laboratories
Hugh Daniel – FreeS/WAN Project
Nancy L. Mintz, Tom Killian, and Mark A. Plotnick – AT&T Laboratories

ABSTRACT

We have implemented a system for virtual private networking, with special attention to the needs of telecommuters. In particular, we used off-the-shelf hardware and open-source software to create a platform to provide IP security and other services for in-home networks.

Our experience has taught us a number of things about the scalability of the FreeS/WAN IPsec system, about the widespread mis-handling of path-MTU discovery on the internet, and about the implications of tunnels on the basic architecture of the network.

Additional Keywords: VPN, Linux, Residential Gateway, MSS, fragmentation.

Overview

Extending the Network

Suppose we have some customers who are affiliated with a corporation[1] that has a good local-area network, possibly even a wide-area network. Further suppose that there are secondary locations that have not heretofore been served by the corporate network; these could include the customers' homes, or a smallish branch office, or whatever.

In many cases, linking the secondary location to the corporate network is highly desirable. The customers may use the secondary location in order to save the time, money, and risk associated with commuting to

Physically versus Virtually Private Networks

Over the years, we have used several different methods for connecting the secondary location to the main network.

One approach was to use a *non-private network*. For example, the customers could use a local ISP to establish a link from their PCs to the Internet. From there, they could telnet to some corporate portal. This has several drawbacks. For one thing, this simple configuration exposes their PCs to attack from all the hackers on the Internet. To fend off such attacks, they would need some sort of firewall. Another drawback is that there are multiple points where their data (including sensitive information such as passwords) could be read by eavesdroppers, and even possibly altered in transit.

Another approach was to arrange it so that each corporation's data moved over physically separate wires. This is sometimes called a *Physically Private Network*. We found such systems to have many drawbacks. Either the secondary locations needed to make long-distance calls, or multiple strategically-placed modem banks were required. Each modem bank required an expensive dedicated "backhaul" link to the main location. Furthermore, it is getting harder and harder to physically protect such links against tampering.

Nowadays, the best approach in most cases is to transform a non-private network into a virtually private network (VPN) using software. The rest of this paper is devoted to explaining how this is done.

Physical View

A typical telecommuting situation is shown in Figure 1. (Many variations and extensions are possible, some of which will be discussed below.)

The networks drawn with double lines constitute the private network. The objective is to give all machines on the private network a reasonable degree of protection against attacks coming from anywhere outside.

In typical usage, a packet goes from one of the clients on the in-home network, through the moat, via the internet, through the security portal, to a host on the corporate network. The moat and the security portal are the primary subjects of this paper.

In Figure 1, if we dared to connect the client machines directly to the internet (instead of going through the moat), we would be exposed to all sorts of attacks, including the following:

- For starters, a typical machine in our address space is routinely attacked every couple of days by probes looking for various well-known security loopholes. Our cable provider filters out attacks against the ms-networking file-sharing ports; otherwise the number of attacks would be even larger.

[1]We will extend the term "corporation" to include universities, government agencies, and other such entities, even if they don't meet the precise legal definition of corporation.

- Secondly, there is the possibility that evildoers could compromise one of the routers or other machines along the way.
- A relatively remote possibility is that a neighbor could attack the signal on the neighborhood cable. This is harder than you might think, because of security features in the cable modems. Each modem (unlike, say, an ethernet tap) knows what IP addresses it is supposed to serve, and is programmed to not pass traffic intended for other modems. This is enforced with weak link-level encryption.

There are two ways to look at how our system protects the private networks. We begin by discussing the physical viewpoint, as diagrammed in Figure 1.

On the home network, the client machines (the PC, the Mac, etc.) are configured so that the moat is their default router. When an IP packet (which we will call a raw IP packet) from the client machine arrives at the moat, it is encrypted. This encrypted data is then encapsulated (''put into an envelope'') and sent via the internet to the security portal at corporate headquarters. The security portal removes the encrypted data from the envelope, decrypts it, and sends the raw IP data on its way via the corporate network.

Data flowing in the other direction is treated the same way.

Because of this system, a hacker cannot determine the raw contents of the packets flowing between the home LAN and the corporate network. Furthermore, the hacker cannot determine the identities of the client machines or the corporate hosts, or even how many of them there are. The only traffic that flows over the public internet consists of packets from the moat to the security portal and vice versa. All such packets use IP protocol 50 (ESP, i.e., Encapsulated

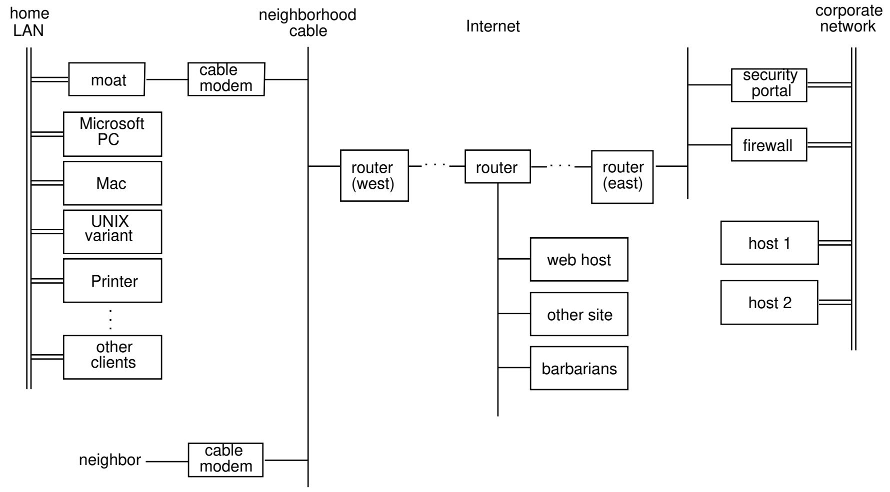

Figure 1: Telecommuting Setup – Physical View.

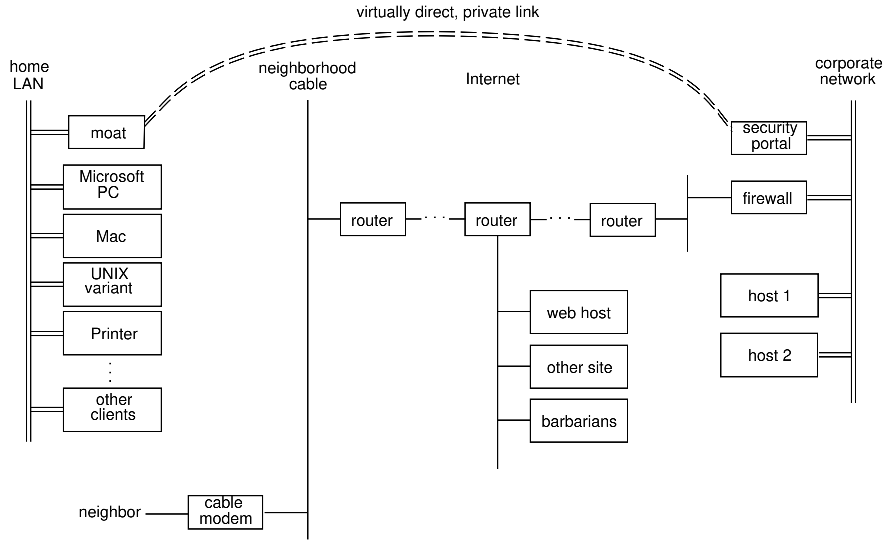

Figure 2: Telecommuting Setup – Virtual View.

Security Payload) so an attacker cannot even determine the IP protocols that the raw packets are using.

A typical machine on the public internet will have no route to the client PC except through the corporate firewall. Even if an attacker guesses that the moat is in a position to forward traffic to the client PC, the moat refuses to forward traffic that doesn't come from the corporate security portal.

Virtual View

Another way to look at this is the virtual viewpoint, as diagrammed in Figure 2. It appears to the users that there is a virtually direct, private connection from the moat to the security portal.

Indeed, the traceroute command conforms to the virtual view, reporting that there is a single hop between the moat the the security portal, no matter how many physical hops there are in Figure 1.

In the setup shown in Figure 2, the only way a client machine can contact a server on the public internet is to go out through the firewall at corporate headquarters.[2] In the other direction, a machine on the public internet cannot initiate a telnet session to any machine on the private network, because that is disallowed by the corporate firewall.

Goals

This project began as a research project, but quickly scaled up when a group within AT&T needed a large number of VPN systems. They found out three weeks before the deployment deadline that the commercially available hardware and software systems that they had expected to use were unreliable and/or unavailable in sufficient quantity. Our main goals, in approximate order of importance, were:

- available on time.
- reliable.
- suitable for a heterogeneous home network (not just a single machine running windows 98).
- easy to install, administer, and manage.
- scalable (as discussed in section "Scalability Issues").
- capable of matching the speed of the cable modem.
- usable as platform for further research in networking and services, e.g., green light for alerting for email).
- affordable.
- quiet (otherwise it will get turned off).
- physically small.

We met all these goals except that the moats in the first batch were not quite as small as we wished.

Ingredients

Hardware

For the first-generation moats, we chose the following configuration:

- Commercial off-the-shelf PC platform.
- No keyboard, no screen, and no video subsystem.
- No sound subsystem and no CDROM drive.
- Two ethernet network interface cards.
- Minimal RAM (16 MB, which is about 2x more than needed).
- Cheapest possible hard disk (4 GB, about 100x more than needed).
- Cheapest possible CPU (K6, 300 MHz).
- Power supply with ultra-quiet fan.
- Floppy drive.
- Serial port.
- BIOS that is happy to boot with no keyboard or video card.

We arranged with the PC manufacturer to have the whole batch built with the desired software (listed below) and our public keys pre-installed.

In the second-generation moats, we used an LS-120 superfloppy in place of the hard disk and floppy. Since the disk is used exceedingly rarely (essentially for boot-up only), wearing out the disk is not an issue.

Software

Running on the typical moat we have

- Linux – operating system.
- FreeS/WAN – IP security system.
- ssh – secure shell.
- configuration scripts.
- network monitoring scripts.
- dhcp client – to get the moat's wild-side IP address from the ISP.
- dhcpd* – to assign IP addresses to client machines.
- xntpd* – time-of-day setting system.
- named* – Domain Name System secondary.

where the three services marked with (*) are accessible from the private-side interface (facing the in-home network) and not from the wild-side interface (facing the cable modem).

Xntpd is more important than you might think; it allows the logfiles on the moat and the security portal to be compared.

The reason why the moat provides DHCP and DNS service on its own, rather than simply forwarding such requests through the IPsec tunnel, is that we want users to be able to use their in-home networks even if the wide-area network is temporarily down.

Software for the moat is cross-compiled; that is, the moat does not contain its own development environment. On the machine where we do the compilations, we rely on gcc and the gnu development tools, and cvs.

It is also worth noting that the services listed above are the only ones running. In particular the moat provides:

[2] There are other ways to do this, as will be discussed below, but this method is particularly easy to implement, and is easily shown to be compliant with corporate firewall policy.

- no web server,
- no telnetd,
- no ftpd, and
- none of the basic inetd services. The moat literally will not give you the time of day (port 37).

Scalability Issues

A stated goal of the FreeS/WAN project is to encourage very wide adoption of IP security. Similarly, AT&T is committed to providing secure IP service over shared media (e.g., cable modems) on a large scale. To us, if it's not scalable, it's not interesting.

Therefore it is not sufficient to have an IPsec implementation that supports just a few tunnels per portal, lovingly configured by an expert.

The system must be database driven, so that hundreds or thousands of customers can be transferred from one system to another, without requiring laborious manual re-entry of data.

The system must not assume that the configuration is known and unchanging. When you have hundreds or thousands of tunnels per portal, something is always changing.

At our behest, the FreeS/WAN team added many "scalability" features to the IPsec system. These include:

- The system can add, delete, or reconfigure tunnels on the fly. The configuration file, which in small-scale systems could be a single flat file, implements an `include` directive which allows us to implement a database with one record per moat. Note that we are using the Linux directory/file system as our database, using one file per record.
- If one of the tunnels cannot be brought up, the system does not hang waiting for it, but goes on to the next.

Configuration Procedure

Databases

We have a simple database that lists all the moat customers. The key fields include

- the wild-side IP address (in case the ISP statically assigns one to this customer) or an indication that the wild-side address will be assigned dynamically via DHCP.
- the range of private-side IP addresses (which we assign arbitrarily).
- the email address to which we send the "welcome" message and instructions for configuring the client machines.

We have a collection of shell scripts (about 1000 lines total) that extracts the required information from the database and configures the moat. Without these scripts the configuration would be quite laborious and error prone. There are more than a dozen configuration files that are affected.

Identification Without Static IP Addresses

The current versions of the FreeS/WAN IPsec package provide only one way for a moat to identify itself, namely, its wild-side IP address. This is a disaster in cases where the ISP assigns a non-constant address to the moat. To overcome this limitation, we devised a complex procedure:

- The moat gets its wild-side IP address *du jour* from the ISP via DHCP.
- The moat rewrites its IPsec configuration files accordingly.
- The moat contacts the security portal using ssh. Note that this does not require the IPsec tunnel to be operational.
- The security portal infers the moat identification based on what ssh key the moat presents, and can see what IP address the moat is using.
- The security portal rewrites its IPsec configuration files accordingly.
- Both sides bring up the tunnel using the new configuration.

The IPsec RFCs envision other forms of identification, such as using the Fully Qualified Domain Name (FQDN) but they have not yet been implemented in FreeS/WAN.

Comparison Against Alternatives

- We have seen hardware solutions that provide features comparable to the moat, but they were more than twice the price of the moat, and were hard to administer.
- Our cable provider charges extra for more than one IP address, and won't provide more than three IP addresses at any price. This affects all VPNs implemented in software.
- We have evaluated software solutions. The leading contender...
  - is not reliable for windows 95. Even on windows 98 it has been known to crash in such a way as to wipe out your C drive.
  - is not available for non-microsoft systems.
  - requires laborious and error-prone installation per PC.
  - requires laborious re-configuration every time the ISP changes the client's wild-side IP address. Note that some ISPs change the IP address multiple times per week.
- The microsoft implementation of PPTP has half a dozen known security holes.
- We probably could have achieved comparable results using other operating systems and/or IPsec packages. There are so many of them that we could not possibly evaluate them all.

Multiple In-Home Networks

In many homes, there are multiple computers, and having them all connected to the same virtual private network may not be the optimal configuration.

- For example, there may be kids in the house who want to surf the internet. The kids have their own computer(s) and should not be using the corporate work-at-home computers.
- As another example, there may be two or more adults with different employers, each of which needs a separate VPN.

Brute-Force Solutions

Right now the kid-net feature can be implemented as shown in Figure 3, where PC-a is connected directly to the wild internet. The user puts a hub between the moat and the cable modem, and arranges with the cable provider to get $N + 1$ IP addresses (one for the moat, plus one for each of the non-corporate client machines). Typically the cable provider imposes a modest charge for each additional IP address, and some providers impose a limit of three IP addresses per customer, or some other low limit.

A lame approximation of this feature can be achieved without paying for a second IP address. You take turns plugging the moat or the kids' computer directly to the cable modem. The cable modem must be reset each time (because it otherwise remembers the Media-Access address of the host to which it is connected, and refuses to talk to any other host). Each reset takes about three minutes. Sigh.

Moat-Based Solutions

A more-elegant solution to the same problem is shown in Figure 4. The wild-side interface of the moat is physically capable of accessing the entire internet. We can put another connector on the moat, implementing another subnet that the kids could use. An advantage of this over the previous solution is that the moat would offer this subnet some minimal firewall service.

A similar solution, again employing multiple interfaces on the moat, could provide for multiple VPNs.

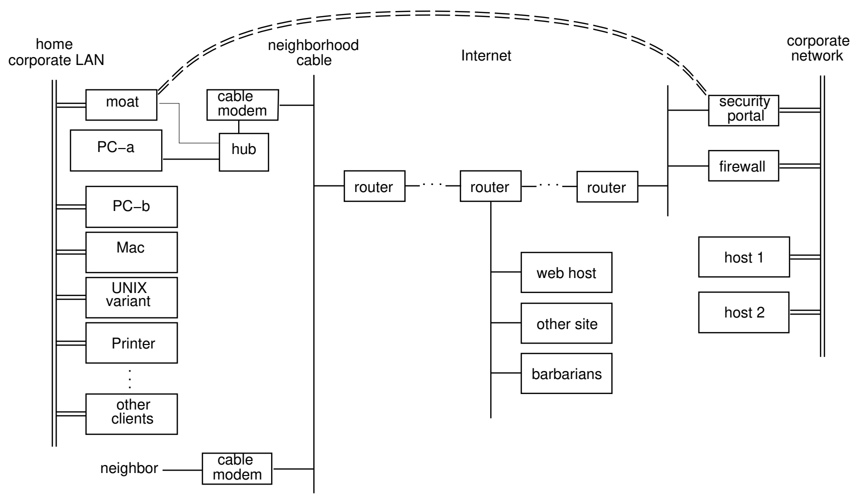

Figure 3: Multi-Net Hardware Solution.

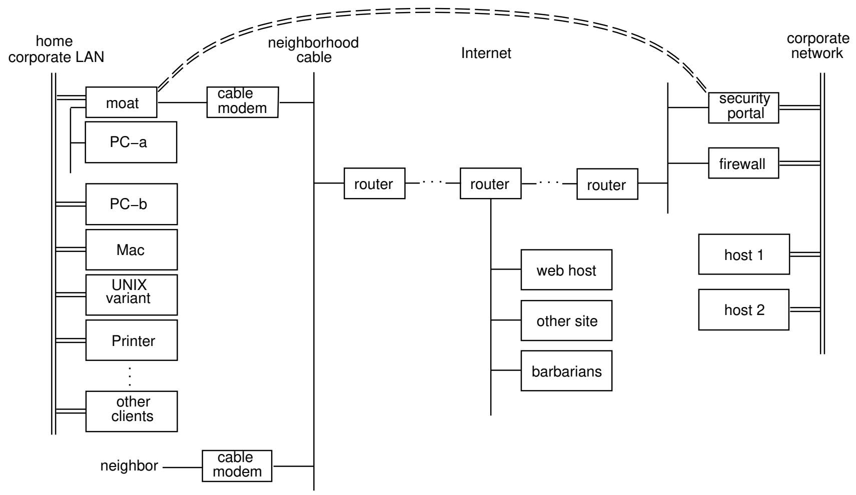

Figure 4: Multi-Net Software Solution.

This is 100% possible in principle. Like many things, doing it badly would be easy, but doing it well is not so easy. With N interfaces on the moat, there are N factorial paths that packets could take, most of which are disallowed. For $N > 2$, a solution that preserves security with high assurance would require more careful work than we have yet been able to give, so this is still in the experimental stage.

Distributed Firewalls

We now consider the internet connectivity of PC-b as shown in Figure 4. This machine (unlike PC-a) has access to the corporate VPN.

Given a sufficiently fast link to the main corporate network, it works just fine for traffic between PC-b and generic hosts on the public internet to go through the corporate firewall. However, it would be more elegant, and in some cases more efficient, if the moat itself could implement corporate firewall policy itself, and route such traffic to/from the internet directly. This is discussed in [3].

Tunneling & Fragmentation

It is quite possible for a tunnel that is fully RFC-compliant to be unable to interoperate with a very large percentage of the sites on the internet, because of fragmentation and MTU problems, as we now explain.

Keep in mind that the objective is reliable and efficient communication. That's all.

As a means to that end, it is better to send a smaller number of large packets, rather than a larger number of small packets. If a packet is fragmented in transit and reassembled before delivery, it magnifies the effect of packet loss in transit. That is because the higher levels of the protocol, which are responsible for retransmission, are forced to retransmit the whole packet, not just the fragment that got lost.

Therefore it is *sometimes* true that the most-efficient packet size is the largest size that can be transmitted without fragmentation, i.e., the *path MTU* (Maximum Transmission Unit). But this is not always true, as discussed below.

As a means to attempt path-MTU discovery, hosts often begin by sending large packets with the DF bit set, and seeing if they get through. But remember this trick is at least two assumptions removed from the actual goal of reliable and efficient communication. See http://www.ietf.org/rfc/rfc1191.html for details.

Many TCP clients (notably microsoft) make an optimistic guess and set their initial MSS (Maximum Segment Size) to the largest plausible value. This is perfectly proper, and should typically result in efficiency if other players do their part. This initial guess is necessarily made with no knowledge of the actual path-MTU.

The large initial MSS makes it likely that early in the session, packets will be sent that exceed the MTU of some router along the path – especially when there is encapsulation going on at some point, such as an IPsec tunnel. Remember that the MSS concept is applicable at the TCP layer, while the MTU concept is applicable at a much lower layer. The assumption that an MSS of size X will correspond to an MTU of size $X + 40$ is invalidated by the overhead bytes introduced by the encapsulation.

Suppose an oversized packet (with the DF bit set) arrives at a router. The RFC says "In this case the gateway must discard the datagram and may return a destination unreachable message." Specifically this message is ICMP type 3 code 4 and it explains that fragmentation is needed and suggests a new packet size. See http://www.ietf.org/rfc/rfc792.html for details.

Note that the frag-needed messages are *optional* according to the RFC.

We note that path-MTU discovery without them is at best rather inefficient. But in real life, the situation is much worse than that. We observe that the vast majority of the world's web servers improperly assume that the routers *must* return a frag-needed message. The microsoft web site is one of the few that is both efficient and robust… efficient in that it starts out by sending large packets, and robust that it will (even in the absence of frag-needed messages) reduce its MSS if large packets don't get through. See observations section below.

There are some firewalls (the Firewall-One brand in particular, and quite likely others) that in their usual configuration do not pass the ICMP frag-needed datagrams. We consider this a weakness in the firewalls. This is a pain in the neck to fix, but overcomes

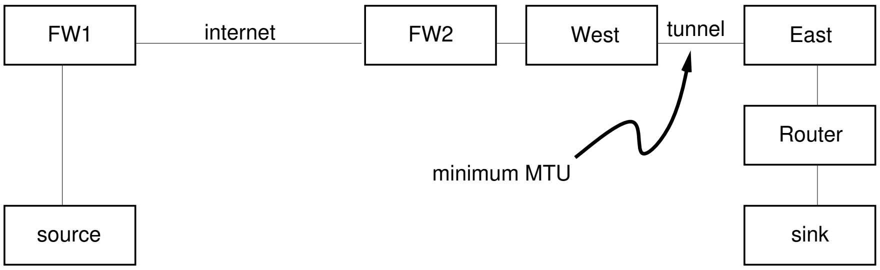

Figure 5: Interior Minimum MTU.

the naughty behavior of about half the world's web sites; see details below.

What's much much worse is that the persons administering typical IPsec tunnels cannot in general fix the frag-needed problem by fixing their own firewalls. Consider the situation shown in Figure 5, where there is a tunnel between West and East. Suppose you control FW2 and everything to the right thereof, whereas FW1 and everything to the left thereof belong to somebody else like aol.com.

It appears that something like 15% of the server-sites in the world assume that there will be "no interior minima" in the MTUs along a path. They create their own black hole, and then improperly fail to perform black-hole detection. Your clients will hang when trying to contact such servers, and there 's almost nothing you can do about it.

Similarly it appears that more than half of the world's sites improperly require the frag-needed messages when there are interior minima.

Note that a verrrry large fraction of the world's clients blissfully (and properly!) negotiate for the biggest MSS they can get.

Finally note that we want the tunnel to interoperate with the world as it actually exists. This is a stricter requirement than merely complying with the RFCs.

Therefore our policy is to have the tunnel implement a virtually-large MTU. This will in some cases require that packets (with DF set) that arrive at one portal (West) be encapsulated in multiple envelopes. When these envelopes arrive at the other portal (East) they will be re-assembled so that an unfragmented packet can be sent on its way toward the final destination (sink).

This policy comes as a shock to some people, because it eliminates any possibility of the source/sink pair being able to discover the effective MTU of the path inside the tunnel. However this is the way it has to be. Suppose for instance that the path inside the tunnel had (at the real-IP level) an MTU equal to the smallest permissible internet packet. If we tried to force the incoming raw packet to be sent in a single envelope, there would be no room for the encapsulation header overhead, and therefore no connectivity at all. An extreme way to make the same point is this: suppose we are sending IP over ATM, which uses 48-byte packets. Setting the DF bit cannot possibly prevent fragmentation of the IP packet into multiple ATM cells.

The basic, essential function of the DF bit is to ensure that packets that leave the source with the DF bit arrive at the destination in one piece. This is important, because not all hosts are capable of reassembling fragments. Note that any packets that are fragmented on entrance to the tunnel are reassembled upon exit, thereby upholding this essential meaning of the DF bit.

Note that this virtually-large MTU is consistent with the fact that (at the abstract level) there is only one hop between the two ends of the tunnel (as reported by, e.g., traceroute) no matter how many hops there are (at the real-IP level) inside the tunnel.

This inability of the endpoints to discover the MTU of the tunnel will in some cases lead to reduced efficiency. In our application (and we believe in most applications) this inefficiency is vastly preferable to

Site	MTU Observation
www.microsoft.com	efficient and robust; large initial MSS but notches down
www.ibm.com	inefficient: always requests a small MSS
www.snap.com	inefficient: always requests a small MSS
www.toad.com	inefficient: never sets DF, always sends small packets
www.sandelman.ottawa.on.ca	inefficient: never sets DF, always sends small packets
www.hotbot.com	semi-naughty: eventually reduces MSS, but is horribly slow about it
www.netscape.com	naughty
www.altavista.com	naughty
www.yahoo.com	naughty
www.clinet.fi	naughty
www.sgi.com	naughty
www.psi.net	naughty
www.cygnus.com	naughty
www.quintillion.com	naughty
akpublic.research.att.com	naughty
www.aol.com	incorrigible
www.intel.com	incorrigible
www.compaq.com	incorrigible

Table 1: MTU bug observations.

the alternative, which is complete inability to reach incorrigible sites such as aol.com and intel.com.

Observations

Terminology: "naughty" means the server site (meaning the source/FW1 combination) does not recover properly when we start out with an MSS that cannot be supported by the actual path-MTU, unless an ICMP frag-needed message is emitted by FW2. Similarly "incorrigible" means that it does not recover even if the frag-needed messages are being generated and passed by FW2; presumably they are being blocked by FW1 or some such.

Table 1 gives a partial list of servers we've checked.

Summary: Tunnel MTU

- There are lots of firewalls out there that eat ICMP.
- There are lots of naughty sites out there that act as if gateways *must* generate frag-needed messages, even though the RFC doesn't require it.
- There are more than a few incorrigible sites out there which (perhaps because of their own firewalls) don't do the right thing even when frag-needed messages are being sent toward them.
- There are clients (notably microsoft) that negotiate for the largest possible MSS. The source and sink make an initial guess based on information about the MTU of the first link at each end of the path, but they have no initial knowledge of the real path-MTU. This behavior appears to be fully compliant with the RFCs.
- The foregoing can be summed up as follows: many sites make the implicit, improper assumption that the path will have no interior minima.
- Tunneling software is quite likely to break that assumption.
- In special cases, it may be possible to get enough control over the clients so that the tunnel can advertise its real MTU, thereby gaining efficiency . . . but this cannot be the default.
- More generally, to achieve a good level of interoperation with the world as it exists, warts and all, a tunnel needs to have a virtually-large MTU.

Routing

The main points of this section are:

- Every portal is a router, and needs to act like one.
- The existence of tunnels requires us to re-think basic internet architecture.

Before explaining these points, we must distinguish a couple of concepts: The kernel (Linux in this case) implements a "kernel routing table". This allows packets within a given host to be routed to the correct interface (eth0, eth1, et cetera). Given a properly set up routing table, the host can function as a gateway, so packets that come in on one interface can be forwarded to another. We will call this level of functionality a *micro* router.

A fully-featured internet router requires other functions. It needs to run a program such as routed(8). That is, it needs to send and receive routing messages, so that it and its peers can keep their routing tables up to date in the face of ever-changing network connectivity. A host with this level of functionality we will call a *macro* router.

Layers of Routing

Next, we must distinguish three or four levels of routing that must occur:

First, there is routing that occurs well below the IPsec layer. Consider the routers shown in Figure 1. They must maintain their routing tables to reflect the routes that connect them. This occurs at the raw-IP layer, and the portals at the ends of the IPsec tunnels generally do not and cannot know anything about this.

Next, there is routing that should occur below the IPsec layer but which is actually done at the IPsec level. The current FreeS/WAN implementation does not trust the kernel routing tables to send the IPsec traffic over the right route. Therefore, as part of the IPsec configuration, FreeS/WAN must be told about the "next-hop" router. That is, in Figure 1, the moat must be told the IP address of the "west" router, and the corporate security portal must be told the IP address of the "east" router.

Next, there is routing that must occur at the IPsec layer. The linux 2.0xx kernel routing mechanism can only route traffic based on the destination IP address. However, the IPsec specifications require that decisions about how to encrypt, and how to route traffic, must be made on an assortment of "selectors" including destination address, source address, protocol type, et cetera. The current version of FreeS/WAN only implements the source and destination selectors.

Another type of routing occurs at or above the IPsec layer. This occurs when a given host has more than one tunnel that serves the same subnet. In Figure 1, imagine that there were redundant security portals on the corporate wide-area network, located at a number of different sites. The IPsec implementation on the moat would need to choose which of these tunnels to use. This decision would be based on information from lower protocol layers, i.e., how efficiently each tunnel can carry traffic, whether it is up at the moment, et cetera.

Finally, there is routing that must occur above the IPsec layer. A portal creates a route to the machines on the far end of the tunnel. In general, the portal should act like a macro-router; i.e., it should advertise this route. For instance, in the case of multiple portals just described, hosts and routers on the corporate network would have a choice of which portal to use when sending traffic to the moat. This would depend not only on the efficiency of the tunnel (once the traffic reached the portal) but on how efficiently a given host could reach a given portal.

The current version of FreeS/WAN cannot handle redundant tunnels. It does not perform high-level macro-routing (i.e., advertising its routes). It does not gracefully handle low-level macro-routing (e.g., changes in the address of the "west" router). These are hard problems, and are very high on our wish-list.

Grand Implications

Consider the following chain of implications:

- There are many good reasons to widely deploy IPsec.
- IPsec requires tunneling. This is the first good reason for really wide application of tunneling.
- The existence of tunnels requires a re-think of the basic design of the internet.

The existing procedures for macro-routing do not appear to scale very well. There are too many routes, and too many updates to the routing tables.

The deployment of large numbers of tunnels could greatly increase the number of routes. This can be expected to put additional stress on the routers.

Furthermore, we must examine the notion that the tunnel is layered on top of the raw-IP link. Link failures break this notion, cutting across the layer boundaries. If (in Figure 1) the route from "west" to "east" changes, that's no big deal, but if "west" totally loses its connection to "east", then the moats (possibly a very great number of moats) must be notified.

The technique of sending routine enquiries ("keep-alive" messages) to check on the health of a path scales exceedingly poorly. If every endpoint of every conversation tried to do this, very soon the entire bandwidth of the net would be used for keep-alive messages. A better strategy is for each node to keep track of the health of its immediate neighbors. Then, if there is an outage, it can send appropriate notifications using higher-level protocols.

The present situation has at some interesting upsides. For example, the IPsec header contains a sequence number. If packets arrive out of sequence, it is a sensitive indicator of loss – more sensitive than the sequencers built into the higher-level protocols. Having a good loss indicator is very important, since loss is practically the only usable indicator of congestion.

It may be possible to exploit this and other properties of the IPsec protocol to solve some problems. Since the IPsec protocol is not 100% set in stone at this point, we have what may be a once-in-lifetime opportunity to design in additional features to solve important problems that heretofore seemed only tangentially (or less) related to the basic objectives of IPsec.

Other Lessons Learned

- Until recently, the ethernet drivers for Linux were quite intolerant of small variations in the ethernet cards. We found many cases where a driver worked OK with a name-brand card, but failed miserably with clones, even clones that work perfectly well under MS windows.

  Furthermore, we found that even with name-brand cards, many of the drivers failed under high-load conditions. This was particularly serious because the FreeS/WAN Linux-IPsec system appears to stress cards and drivers in ways that ordinary network traffic does not.

  Recent drivers seem much better behaved.
- Linux is remarkably vulnerable to having its power turned off when the filesystems have not been cleanly unmounted.

  Since the moat has no screen or keyboard, a filesystem inconsistency that cannot be fixed by "fsck -a" or other automatic procedure incapacitates the system totally and permanently. We fixed this by mounting a ramdisk on /, making /usr read-only, and arranging that if /var fails fsck, it is restored from a read-only backup.
- In the home environment, people are quite intolerant of machines with noisy fans.

Performance

The round-trip "ping" time between a typical client machine and a typical host on the corporate network is routinely less than 25 milliseconds for short packets and less than 85 milliseconds for 1400 byte packets. Almost all of this time is attributable to the link between the moat and the security portal at the real-IP level; the encryption and encapsulation takes at most 1 ms for the short packets and at most 15 ms for the long packets.

On a laboratory link with negligible delay at the real-IP level, the system is observed to sustain a throughput of 6.9 megabits per second. This is at or near the limit imposed by the CPU cycles required to perform the triple-DES encryption of the data stream.

Hardware reliability has been good. During roughly 2000 moat-weeks of operation, we have not had any failures in the field except for one case where it appears the cable modem and moat were destroyed by a lightning strike, one hard-disk crash, and one transient outage attributed to overheating in an un-air-conditioned location in midsummer.

Software reliability has been good. We discovered one race condition that would cause a tunnel to fail with a probability of about 0.1% per day per moat. The FreeS/WAN authors quickly fixed this. The only large-scale outage resulted from a failure of syslogd of all things, due in turn to the Linux kernel's highly sub-optimal implementation of fsync(). It uses an N^2

algorithm, which cannot keep up when the log file gets too big.

Far and away the biggest performance problems are at the raw-IP layer. Remember, IPsec is layered on top of raw IP. If the lower layer is broken, there's not much the higher layers can do about it. We have learned that it is important to give the users (and support staff) enough access to the various layers that, when there is a problem, they can tell whether it is the raw-IP layer or the IPsec layer that needs attention.

Conclusions

- Our customers have been delighted with their moats. The combination of moat and cable modem gives the in-home LAN such good performance that customers usually find it indistinguishable from being at work.
- Our system administrators have been very happy with the moats, because they are more reliable and easier to administer than the alternatives.
- Using off-the-shelf hardware allowed us to meet a very tight time schedule at a very reasonable price.
- Using open software was a real joy. When things go wrong, you can fix them. When you want to add features, you can just do it.
- Our experience with the moats has given us a much greater understanding of which features are needed in an in-home service platform.

Acknowledgements

We are profoundly grateful to the authors of FreeS/WAN and the software mentioned in the "Software" subsection. Other folks who have made important contributions to the moat project include Norm Schryer, Sharon Gray, Sam Alexander, and Steven Gao.

References

[1] Technical information on IPsec can be found at http://www.ietf.org/rfc/rfc2401.txt and references therein.

[2] The FreeS/WAN home page is at http://www.xs4all.nl/˜freeswan and includes links to detailed documentation.

[3] *Distributed Firewalls* by Steven M. Bellovin, http://www.research.att.com/˜smb/papers/distfw.ps .