Preserving TCP Connections Across Host Address Changes

Vassilis Prevelakis[1], Sotiris Ioannidis[2]

[1] Computer Science Department, Drexel University,
Philadelphia, PA 19104, USA
[2] Computer Science Department, Stevens Institute of Technology,
Hoboken, NJ 07030, USA

Abstract. The predominance of short-lived connections in today's Internet has created the perception that it is perfectly acceptable to change a host's IP address with little regard about established connections. Indeed, the increased mobility offered by laptops with wireless network interfaces, and the aggressive use of short DHCP leases are leading the way towards an environment where IP addresses are transient and last for short time periods. However, there is still a place for long-lived connections (typically lasting hours or even days) for remote login sessions, over the network backups, *etc.* There is, therefore, a real need for a system that allows such connections to survive changes in the IP addresses of the hosts at either end of the connection.

In this paper we present a kernel-based mechanism that recognizes address changes and recovers from them. Furthermore, we discuss the security implications of such a scheme, and show that our system provides an effective defense against both eavesdropping and man-in-the-middle attacks.

1 Introduction

Applications based on the Internet Protocols generally assume that the address of a given node remains the same over long periods of time. Long term connections, especially those that use connection-oriented protocols such as TCP, rely on this assumption to allow connections that may "last months" and can even survive temporary disruptions to the network. The assumption of address immutability is, however, increasingly difficult to sustain. Mobile nodes (*e.g.,* laptops, phones, PDAs, *etc.*) can change addresses as they move from one network to another, but even fixed nodes connected to the Internet via dial-up or DSL link have addresses that change every time their connection is reset. In some cases ISP's initiate such address changes to force users that need a permanent (static) IP address to pay for one. Unfortunately, established connections (*e.g.,* ssh sessions) do not survive the address change, because they rely on fixed source and destination IP addresses. In order to protect these connections when one of the endpoints gets a new IP address, some kind of mechanism is required to allow both ends of the connection to update the addresses associated with that connection, or to continue using their initial addresses through address translation or tunnels.

In this paper we examine various techniques that have been proposed to address this problem and describe a new technique based on "redirects." We also discuss the security implications of the use of such mechanisms and propose a novel technique that prevents third parties from hijacking connections.

2 Connection Redirection

To better understand the redirect protocol, consider the following scenario shown in Figure 1 where the communication between two hosts is disrupted because the IP address of one of the hosts changes.

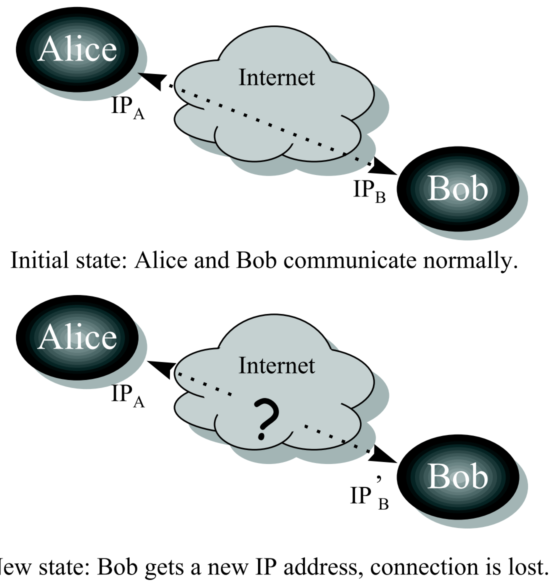
Alice
Internet
IP_A
IP_B
Bob
Initial state: Alice and Bob communicate normally.
Alice
Internet
IP_A
?
IP'_B
Bob
New state: Bob gets a new IP address, connection is lost.

Fig. 1. When Bob acquires a new IP address, all established connections with Alice will be lost.

Assume two hosts, Alice and Bob. with IP addresses IP_A and IP_B respectively. Initially, Alice and Bob establish a TCP connection and communicate normally. At some stage in the communication, Bob is forced to acquire a new IP address (IP'_B). At that point if Alice sends a packet to Bob's original address, IP_B, she will either get an ICMP error (*e.g.,* ADDRESS UNREACHABLE), or the packet will be lost (silently). Moreover, if another host has grabbed IP_B, Alice will get a TCP RST. In the first and last cases Alice will immediately know that the connection has been lost and she will tear down her side of the connection, but in the second case Alice will have to wait until the connection times out.

In every case, the connection will be lost and will have to be reestablished. To retain the connection, both sides must update the state of the IP tables in the network stack, changing all instances of IP_B to IP'_B. Since Bob knows that his address has changed, he can effect the local changes, but he also has to inform Alice via an Address Change Message (ACM, shown in Figure 2), so that she can update her network state.

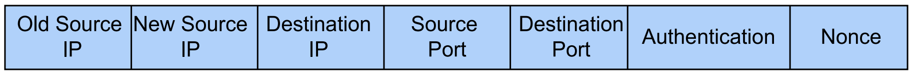

Fig. 2. Format of the Address Change Message.

The ACM may affect all established TCP connections between Bob and Alice causing all applicable network stack entries to be updated. Alternatively, it may apply only to a specific TCP connection. In the latter case Bob will need to send individual ACMs for every TCP connection between himself and Alice.

While the above protocol can be used to redirect TCP connections one has to be sure that this mechanism cannot be used to hijack existing connections. Specifically, the protection mechanism must address both packet injection attacks and data modification attacks.

Packet Injection A packet injection attack is one where an attacker sends a specially crafted ACM causing an existing connection to be redirected to a host controlled by the attacker. This attack is facilitated by the ability of the attacker to eavesdrop on the communication channel and, hence, is particularly likely in wireless networks where it is trivial to monitor network traffic.

To protect against such attacks, we can either use some pre-arranged secret to guard the ACM, or use sequence numbers that the attacker is unlikely to guess (unless, of course, the attacker is able to monitor the traffic). The extent to which we want to protect hosts from redirect attacks will define which defence mechanism one should use. Understanding the specific types of packet injection attacks will help us form protection guidelines.

The ability to eavesdrop on the communication opens additional avenues of attack such as replay attacks whereby the attacker records an earlier exchange involving an ACM and reuses it to acquire a connection at some point in the future. In order to understand this attack consider the following scenario.

In a LAN where address assignment is handled via DHCP, the attacker can trigger an address change on a host causing it to emit an ACM message. A further address change will cause the victim host to move to a different IP address allowing the attacker to use the original ACM to redirect traffic back to the freshly released address.

Data Modification In a man-in-the-middle scenario, the attacker is able to inspect and modify packets exchanged between the two communicating parties. While this attack is harder to carry out in general, if the attacker is in the same LAN as the victim, ARP spoofing can be used to allow traffic to flow through the attacker.

To appreciate the difficulty of countering data modification attacks, consider the use of a shared secret to protect the ACM. If the secret is sent during the current session, the attacker will be able to intercept it and modify it. Thus, the two parties must exchange the secret ahead of time, or use a trusted third party to introduce them to each other.

2.1 Global versus local redirects

When redirecting connections we can opt to send a single redirect request which applies to all active sessions between the two hosts, or send ACMs for each connection separately.

By using individual ACMs for each connection we leverage TCP's sequence numbers in order to ensure that the connection is unlikely to be hijacked because the attacker must guess not only the source and destination ports, but the sequence number as well. If the attacker cannot eavesdrop on the connection, they are unlikely to be able to guess the correct combination or even to mount a brute-force attack by trying all possible combinations due to the size of the search space.

For global redirects we can use a token established at the beginning of the session to authenticate the redirect request. This token can be a 32-bit or 64-bit quantity sent in the original connection request packet. Either side can use that token if it needs to send a redirect packet.

Neither of the two solutions above address the problem of an eavesdropping attacker. Of course, if an attacker is able to monitor the traffic on a connection, he or she will be able to mount a great variety of attacks against that session (including message injection). However, it may not be acceptable to allow the attacker to redirect the entire session.

A good way to address this problem is not to send the token in the clear but to use Diffie Hellman exchange to establish a token, known by both sides but unavailable to a potential eavesdropper. [17] The DH exchange, however introduces extra overhead, is vulnerable to a man in the middle attack and may allow an attacker to mount a denial of service attack on either side by forcing them to perform repeated DH exchanges.

We have addressed these issues with a two level technique that (a) limits the expensive negotiation to long-lived sessions and (b) allows information from previous sessions to be used to enhance the level of security and resist man-in-the-middle attacks as well. The latter technique is similar to the authentication mechanism employed by ssh, that is to transfer the host key when the first connection between the two hosts is made. Moreover, since the authentication procedure involves the two hosts and not the individual connections we can further reduce overheads by allowing one redirect message to affect all the connections between the two hosts.

3 Design Considerations

We have developed a prototype to test our methods and provide feedback on the efficacy of the various techniques described the previous section. It is important to observe here that we are primarily concerned with authentication and, as we shell see later, data integrity; privacy is not our concern and hence we avoid encryption and its associated overhead. Existing systems such as ssh or TLS may be used as needed to satisfy privacy requirements.

3.1 Initial Key Exchange

As we have seen above, in order to construct a system which is resistant to both data monitoring (eavesdropping) and man-in-the-middle attacks, we need to set up a session key. This is done as part of the initial TCP/IP handshake (Figure 3).

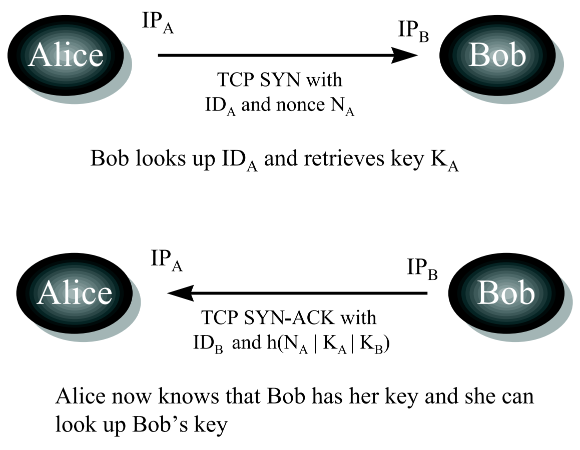
Alice
IP_A
IP_B
Bob
TCP SYN with
ID_A and nonce N_A
Bob looks up ID_A and retrieves key K_A
Alice
IP_A
IP_B
Bob
TCP SYN-ACK with
ID_B and $h(N_A \mid K_A \mid K_B)$
Alice now knows that Bob has her key and she can look up Bob's key

Fig. 3. Session key agreement between parties that know each other. Alice sends her ID along with a nonce to Bob. Bob uses the ID to find Alice's key and then responds with his ID along with the hashed value of the nonce along with their two keys.

Assuming that Alice and Bob have met before, they already know each other's secret key. This is used in the negotiation for the session key. Note that Bob does not have a single key that he gives to all his friends, but rather maintains a list in the form:

$$ID_A\ K_A\ K_B\ \text{timestamp}$$

The reason is that if hosts always use the same key for all their transactions, a malicious host M could contact Alice to get her key and then contact Bob to get his key and thus be in a position to impersonate either one. We prefer this method of authentication as opposed to using public key cryptography because our method is vastly cheaper in terms of CPU requirements.

The timestamp field is used to indicate the last time the host was contacted to allow for pruning of the list to avoid maintaining old and potentially useless information.

3.2 First contact

If Alice and Bob have never met before, or if either host has purged the key information from its database, they will need to exchange keys. This operation is not performed by the kernel, as it involves a lot of operations that are better done in user-land.

We have modified the network code in the OpenBSD kernel to call an application (keyserv) during the initial stages of the session key exchange (Figure 4). The keyserv program maintains a table with known hosts so that the storage and management of these keys may be managed by the user.

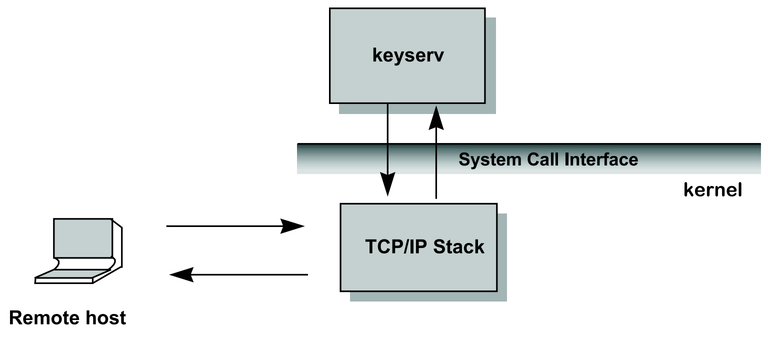

Fig. 4. When an ACM-aware connection is initiated, the kernel asks a user-level application to carry out the session key negotiation.

3.3 Short versus Long lived connections

Based on typical TCP/IP connection lifetimes we observe that a great many connections are short lived, and do not require the use of our mechanism. For such connections we should not attempt to use our mechanism to avoid burdening the end-systems with the associated overhead. But how can we determine whether a connection is likely to require the use of the connection redirection system? Initially we based our decision on the service (*i.e.,*we activated our protocol based on the port used for the connection). This allowed us to use the mechanism for, say, ssh connections, but not http. This approach assumes that we have a good understanding of the type of services in use, the use various ports used and that each service can be nicely categorized as short- or long-lived.

So we looked for a more flexible mechanism and we ended up using the following heuristic: *if a connection lasts longer than 10 seconds, it is likely to be a long-lived connection.* We, therefore, arrived at the state model shown in Figure 5.

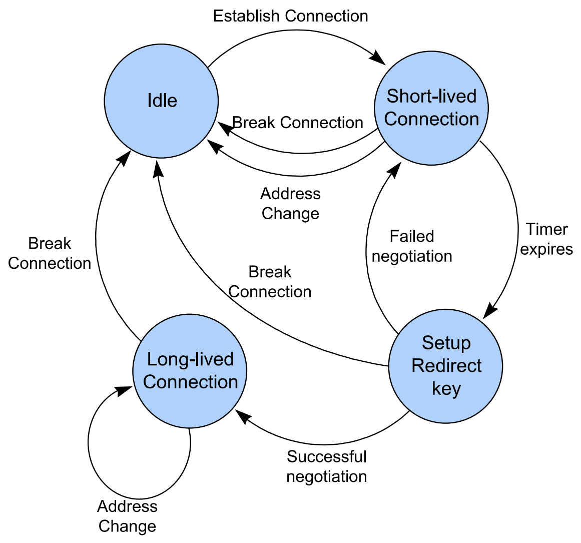

Fig. 5. State diagram showing the activation of the address redirection system.

Initially the host is Idle (no connections). When a remote host connects, we start a 10 second timer. Before the timer expires, we treat the connection as a short-lived one, so that if an address change occurs, the connection is dropped. Assuming that the connection is still up when the timer expires, we establish the redirect key and enter the "long-lived" state where our mechanism can be used to recover from address changes. This technique allowed us to reduce overheads and yet has proven to be extremely accurate in predicting the long-lived connections.

3.4 Data Integrity

In RFC-2385 [9] the authors suggest that BGP sessions can be protected through the use of MD5 hashes. The proposed technique involves calculating the hash of the packet to which we have appended a "password."

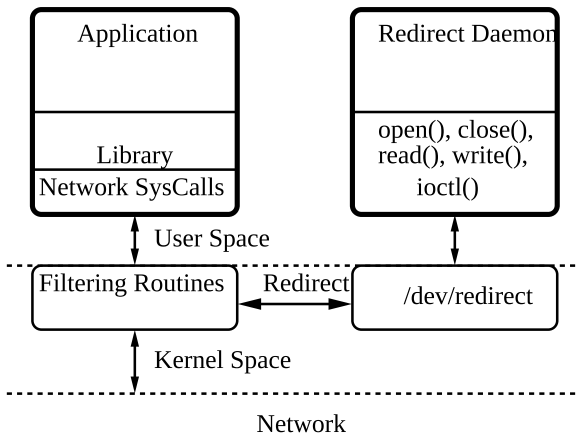

Fig. 6. Block diagram of the redirect implementation. IP addresses are modified on the fly using filtering routines to maintain the network connections.

In our system we can do something similar using the session key as password and utilizing a more secure algorithm instead of MD5 (see discussion by Dobbertin [7] on the problems with the MD5 hash). This scheme will enhance the typical TCP session with integrity checks (but will not provide privacy) at a very small overhead, since we have already carried out the key establishment negotiation. Since this integrity support is available to long-lived sessions, it is activated after the timer expires (see Figure 5 above).

4 Implementation

We implemented the redirect architecture under OpenBSD 3.6 [1] as a proof of concept. Our implementation consists of three components: *(1)* a set of kernel extensions, that are responsible for locating and enabling the modification of IP addresses dynamically in the kernel; *(2)* a user-level daemon process, which implements the redirect monitoring system; and *(3)* a device driver, which is used to modify the IP data structures according to the requirements of the redirect monitor. Our prototype is very lightweight, consisting of a few hundred lines of `C` code.

Figure 6 shows a graphical representation of the system, with all its components. The core of the redirect mechanism lives in kernel space and is comprised of the filtering routines and the device driver. The redirect monitoring engine lives in user space, inside the redirect daemon process. Any incoming or outgoing IP packets go through the filter and are subject to possible redirect.

Kernel Extensions We implemented a set of kernel extensions to permit to modify the Protocol Control Block of existing network connections. This functionality is supported by two operations: `search_pcb` and `modify_pcb`. More specifically, we search all the Protocol Control Blocks for connections of interest and then we modify them accordingly. In the case of a local IP address change, we need to modify the source address of every existing connection. Whereas in the case of a remote IP address change, we only need to modify existing connections to that remote host.

Redirect Device To maximize the flexibility of our system and allow for easy experimentation, we decided to make the redirect daemon a user level process. To support this architecture, we had to implement a *pseudo device driver*, `/dev/redirect`, that serves as a communication path between the user–space redirect daemon, and the IP packet redirect engine in the kernel. Our device driver, implemented as a loadable module, supports the usual operations (`open(2)`, `close(2)`, `read(2)`, `write(2)`, and `ioctl(2)`).

```
struct redirect_request {
        int             local_or_remote;
        in_addr_t       oldIPaddress, newIPaddress;
};

for (ever) {
        if (IP local address change) {
                update Protocol Control Block with new source IP address
                for (every existing connection)
                        notify remote redirect daemons
        } else if (IP remote address change) {
                update Protocol Control Block with new destination
                IP address
        }
}
```

Fig. 7. Pseudocode of redirect daemon.

Redirect Daemon The last component of our system is the redirect daemon. It is a user-level process responsible for making decisions on whether to redirect IP packets or not. These decisions are based on the changes of the host IP address.

The redirect daemon continuously monitors the network interface for IP address changes. When it detects a change it starts executing the protocol described in Section 3. The redirect daemon of the host that experienced the IP address change, communicates with the redirect daemon of all the network hosts it has established connections to notify them of the IP address change. It then issues a call to the redirect device driver to update the in-kernel Protocol Control Block tables of the existing connections with the new IP address. Similar update actions are taken by the remote redirect daemons upon the receipt of the notification (see Figure 7).

5 Related Work

The problem of maintaining existing connections when the IP address changes is not unique to home networks. As with cell phone networks, Mobile IP (MIP) systems have to address the problem of "handoff" *i.e.,* what happens if the cell phone or mobile PC moves from one area to another [16].

However, MIP systems must also satisfy additional requirements related to the roaming nature of mobile users. In particular, what happens whenever a mobile PC remains disconnected from the network for a significant amount of time (*e.g.,* during a long flight, or over the weekend) [18]. Another issue is how to ensure that other computers can establish connections to the mobile PC while it moves from network to network.

5.1 Forwarding Node

These problems necessitate the use of a forwarding node that has a fixed IP address [3, 5, 6]. The mobile PC contacts the forwarding node in order to send and receive packets. A popular way of handling this transparently is via an overlay network: the mobile PC establishes a tunnel with the forwarding node and sends all packets over the tunnel (*i.e.,* the tunnel is designated the default route), while the forwarding node performs NAT on the packets using the tunnel. Other hosts think that packets from the mobile PC originate from the forwarding host due to the use of NAT, while incoming packets are sent over the tunnel to the mobile PC. Packets flow though the tunnel oblivious to the changes of the IP address of the mobile PC.

We have a similar setup in operation for almost 8 years. The forwarding station is an OpenBSD machine connected to the network, while the "mobile PCs" are located in home networks connected via DSL or cable modems. The home networks use an embedded system (running a special version of OpenBSD that boots off a compact flash memory device) acts as an integrated firewall and VPN gateway [15, 14].

We use IPsec in tunnel mode to implement the overlay network (Figure 8). Our recent implementation based on OpenBSD 3.0 has been in continuous operation for almost three years (1022 days uptime) demonstrating that sessions can survive even migrations between ISPs (Verizon DSL to Comcast cable).

However, the overlay network technique has two major limitations: one is that it requires a forwarding node with a fixed, globally unique IP address, and the other is that packets always have to take a detour via the forwarding station in order to reach hosts in the Internet. The latter both increases latency and leaves the system vulnerable to failures in the forwarding node, the network hosting the forwarding node, or the transit networks linking the mobile PC to the forwarding node [2].

While these issues may be acceptable in the case of a truly mobile PC, we do not believe that they are acceptable in a home setting, where the end-user is unlikely to be willing to shoulder the cost of the forwarding station or the latency imposed by the detour to the forwarding node.

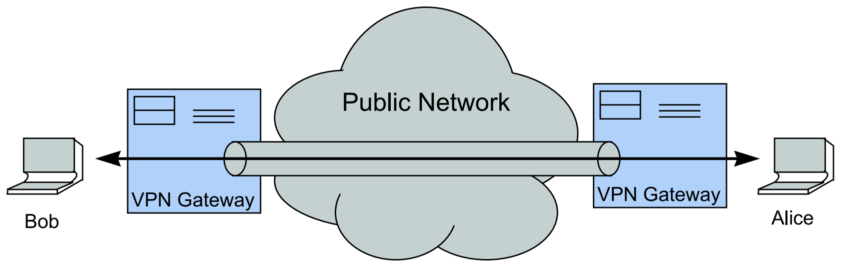

Fig. 8. The overlay network hides changes in the external addresses of the gateway hosts. Internal hosts (Bob and Alice) can use their own addresses all the time.

5.2 Proxy-based systems

Another way to handle address changes is to use a common rendezvous host [3, 19, 8]. Both sides connect to the third part that has a fixed address. In this way even if one or even both hosts change addresses, they can locate each other via the common host. This technique is used mainly by voice telephony (VoIP) applications. Such applications also have the added benefit of using connectionless protocols allowing much flexibility in dealing with address changes.

5.3 Mobile IPv6

The design of IPv6 has created a special address class (link-local addresses) within the huge available address space. Link-local addresses are not routable, but are unique and serve to identify a host within a LAN. Though the use of tunneling, hosts can communicate using their link-local addresses and hence be immune to address changes in the intervening network. A special mechanism based on two new destination option fields within packets (binding update and binding acknowledgement) are used to facilitate the updates to the tunnels [13, 12, 4, 10, 11].

6 Conclusions

We have presented a system that allows TCP connections to survive addresses changes in the communicating hosts. Our system is designed to allow existing connections to be migrated to new IP addresses without the knowledge or cooperation of the application. Under our system, when an address change occurs, all instances of the original IP address in the kernel IP tables, are dynamically replaced with the new address. Remote systems that have established connections are also notified so that they can update their own data structures.

Recognizing that without adequate safeguards this procedure would create serious security problems, we have implemented a comprehensive security mechanism that protects connections from hijacking even against man-in-the-middle attacks.

Care has been taken to minimize the costs associated with this mechanism, both by reducing the computational overheads and by deferring the expensive cryptographic operations until we are reasonably sure that the connection is in fact long-term and can, therefore, benefit from our services. Another benefit of our approach is that having carried out the necessary mutual authentication between the two hosts, we can use this information to provide integrity checking of the connection with almost negligible overhead. We believe that our system combines efficiency and utility and we would like to see it become a standard feature of all IP-based systems.

Acknowledgements This work was supported in part by the National Science Foundation under grants ANI-0133537, DUE-0417085 and CCR-0331584.

References

1. The OpenBSD Operating System. http://www.openbsd.org/.
2. N. Aghdaie and Y. Tamir. Client-Transparent Fault-Tolerant Web Service. In *Proceedings of the 20th IEEE International Performance, Computing, and Communications Conference*, April 2001.
3. I. F. Akyidiz. Mobility Management in Current and Future Communications Networks. *IEEE Network*, 12(6):39–49, July/August 1998.
4. P. Bhagwat and C. Perkins. A Mobile Networking System based on Internet Protocol (IP). In *Proceedings of USENIX Symposium on Mobile and Location Independent Computing*, pages 69–82, August 1993.
5. A. T. Campbell, J. Gomez, S. Kim, Z. Turanyi, and C. Y. Wan. Comparison of IP Micromobility Protocols. *IEEE Wireless Communications*, pages 72–82, February 2002.
6. A. T. Campbell, J. Gomez, S. Kim, Z. Turanyi, C. Y. Wan, and A. G. Valko. Design, Implementation and Evaluation of Cellular IP. In *IEEE Personal Communications, Special Issue on IP-based Mobile Telecommunications Networks*, June/July 2000.
7. H. Dobbertin. The Status of MD5 After a Recent Attack. *RSA Labs' CryptoBytes*, 2(2), Summer 1996.
8. D. Funato, K. Yasuda, and H. Tokuda. TCP-R: TCP mobility support for continuous operation. In *IEEE International Conference on Network Protocols*, pages 229–236, October 1997.
9. A. Heffernan. RFC 2385: Protection of BGP Sessions via the TCP MD5 Signature Option. Request for Comments, Internet Engineering Task Force, August 1998.
10. John Ioannidis, Dan Duchamp, and Gerald Q. Maguire Jr. IP-Based Protocols for Mobile Internetworking. In *Proceedings of SIGCOMM*, pages 235–245. ACM, September 1991.
11. John Ioannidis. *Protocols for Mobile Internetworking.* PhD thesis, Columbia University in the City of New York, 1993.
12. D. Jonhson and C. Perkins. Mobility Support in IPv6. Internet Draft, Internet Engineering Task Force, July 2001. Work in progress.
13. C. Perkins. RFC 2002: IP Mobility Support. Request for Comments, Internet Engineering Task Force, October 1996.
14. Vassilis Prevelakis and Angelos Keromytis. Designing an Embedded Firewall/VPN Gateway. In *Proceedings of the International Network Conference*, 2002.

15. Vassilis Prevelakis and Angelos Keromytis. Drop-in Security for Distributed and Portable Computing Elements. *Journal of Internet Research*, 13(2), 2003.
16. P. Stuckman. *The GSM Evolution*. Wiley, 2003.
17. Gong Su. *MOVE: Mobility with Persistent Network Connections*. PhD thesis, Columbia University, New York, New York, 2004.
18. R. Zhang, T. F. Abdelzaher, and J. A. Stankovic. Efficient TCP Connection Failover in Web Server Clusters. In *Proceedings of IEEE InfoCom*, March 2004.
19. S. Zhuang, K. Lai, I. Stoica, R. Katz, and S. Shenker. Host Mobility using an Internet Indirection Infrastructure. In *First International Conference on Mobile Systems, Applications, and Services (ACM/USENIX Mobisys)*, May 2003.