Transparent Mobility with Minimal Infrastructure

Praveen Yalagandula, Amit Garg, Mike Dahlin, Lorenzo Alvisi,Harrick Vin
ypraveen, amitji, dahlin, lorenzo,vin@cs.utexas.edu
University of Texas at Austin

Draft July 2001
Please send comments to the authors.
Please see http://www.cs.utexas.edu/users/lasr for updates.

Abstract

In this paper, we introduce VIP—a virtual IP layer—that applies the principle of virtual addressing to Internet naming. VIP's goal is to support mobility in a way that is incrementally deployable and that requires little installation or configuration effort. VIP achieves this goal by following two design principles (1) transparent mobility*: the system virtualizes the IP level of the protocol stack—the "neck of the protocol hourglass"—to avoid modifying higher-level network protocols and applications, and (2)* minimal infrastructure: *the system takes advantage of and minimizes changes to existing network infrastructure. In particular, VIP relies on widely-deployed infrastructure—DHCP for dynamic IP assignment, Dynamic Secure DNS for updating name-to-IP mappings, and IPSec for secure communication—rather than requiring deployment of new translation infrastructure. Overall, we find that VIP efficiently supports transparent mobility in a way that an individual user can easily deploy and use.*

1 Introduction

The current Internet naming abstraction requires applications to explicitly translate machine names to IP addresses. Applications then use these IP addresses to refer to and communicate with remote machines. Although exposing physical IP addresses in Internet naming has worked tolerably well in the past, users' network environments are becoming more complex and dynamic. Soon, a individual user may have dozens of machines sharing data and services, and each machine's IP addresses may often change due to mobility, due to switching between different network connections (e.g., Ethernet, 802.11, and infrared), and due to dynamic IP assignment (e.g., DHCP). Because higher level protocols and applications often implicitly assume that IP addresses correspond to specific, unchanging hosts, changes in the physical IP address used to reach a host can cause failures in these protocols and applications.

In this paper, we introduce VIP, a virtual IP layer, that applies the principle of virtual addressing to Internet naming[1]. VIP presents an abstraction in which machines refer to one another by name and in which physical IP addresses are hidden from higher-level protocols.

Two key goals of VIP's design are incremental deployability and minimal configuration: individual users should be able to easily deploy and make use of the system to enable seamless communication across the user's collection of machines, and adding a new VIP-enabled device to a system should be nearly as simple as adding a DHCP-enabled device to a system today. VIP achieves these goals by following two design principles (1) **transparent mobility**: the system virtualizes the IP level of the protocol stack—the "neck of the protocol hourglass"—to avoid modifying higher-level network protocols and applications, and (2) **minimal infrastructure**: the system takes advantage of and minimizes changes to existing network infrastructure.

MobileIP [13, 15] also attempts to support transparent mobility, but despite its development in 1994 and standardization in 1996, MobileIP is not widely used. We believe this is due to its infrastructure requirements:

[1] A previous system, also named VIP, shares our philosophy of virtualizing IP addresses to hide physical IP address changes [19], but our implementation and system properties differ considerably. See Section 2 for details.

to use MobileIP a user must acquire *static, globally unique IP addresses* for each device and a user must deploy a *home agent* machine that forwards packets routed to these addresses to the user's mobile devices.

Although these design trade-offs made sense in the early 1990's, recent trends in network deployment increase the significance of these requirements as barriers to deployments. First, the 32-bit IPv4 address space makes static IP addresses expensive. For example, an ISP account that includes multiple static IP addresses may be much more expensive than a dynamic-IP account, making it costly for an individual user to deploy MobileIP for personal use. In addition, the widespread use of firewalls to protect intranets and even home networks makes it hard to deploy a globally-reachable MobileIP home agent; often a dedicated machine must be deployed to the public side of the firewall, where machine deployment may be tightly controlled.

VIP is a simple system that addresses these problems. VIP uniquely identifies machines by their fully-qualified domain names (FQDNs) (e.g., example.acm.org) rather than their IP addresses (e.g., 199.222.69.43), and uses Secure Dynamic DNS (DDNS) [5] to update and distribute name-to-address mappings as machines move. To maintain backward compatibility, each pair of communicating nodes negotiates virtual IP addresses, which are opaque 32-bit tokens that correspond to their FQDNs. Applications and other layers above VIP use these 32-bit virtual IP addresses rather than physical IP addresses. When machines communicate, the VIP layer translates between virtual and physical IP addresses.

This approach supports transparent mobility without requiring deployment of new infrastructure. First, applications and other network layers above the VIP layer never see physical IP addresses, so the VIP layer is free to change the mapping from the the virtual IP address token used to identify a machine to the physical address used to route packets. Second, routers and other infrastructure below the VIP layer never see virtual IP addresses, so each pair of communicating nodes is free to negotiate lightweight VIP-to-FQDN mappings rather than relying on globally unique and unchanging static IP addresses. Third, by using DDNS for mapping FQDNs to IP addresses, the system takes advantage of existing translation infrastructure rather than requiring deployment of new translation infrastructure. Furthermore, free third-party DNS servers such as no-ip.com and dyndns.org allow individual users to use the system without even having to manage their own name servers.

We have implemented the VIP system in Linux. Our experiments show that the system provides efficient remapping both when one node moves and when both parties in a connection simultaneously change their addresses. A key optimization in the system is peer-to-peer hints, which can greatly improve the latency of this remapping. We use IPSec [10] to provide end-to-end security as VIP $\leftrightarrow$ IP address mappings change; to keep infrastructure requirements low, VIP implements the option of a simple peer-to-peer anonymous key exchange protocol for IPSec that is similar to the one used by SSH [11]. Overall, we find that new infrastructure and practices that have become widespread since the standardization of MobileIP (e.g., DHCP for dynamic IP assignment [4], Dynamic Secure DNS for updating name-to-IP mappings [5], and IPSec for secure communication [10]) make it relatively simple to support transparent mobility without requiring other infrastructure changes.

The main limitation of this approach is our decision to modify end-stations rather than relying on external infrastructure. There are two issues. First, although our system is backwards compatible in that it allows communications with unmodified machines, our system does not support migration of connections unless both participating machines implement our extension. Second, although this approach simplifies many aspects of deployment, modifying end-stations does involve barriers of its own. However, as others have argued [18], although the MobileIP design assumed that it was easier to modify routers than end stations, in practice the reverse seems to be true. Furthermore, modifying end stations solves the "chicken and egg" problem faced by infrastructure-based approaches: in an end-station based approach, an individual can take advantage of the optimization by upgrading her machines; whereas in an infrastructure-based approach, there is little incentive to deploy new infrastructure until a large user base emerges and little incentive to become part of that user base until infrastructure is deployed.

The rest of this paper proceeds as follows. In Section 2, we describe how our work differs with previous achievements in this field. In Section 3, we describe the protocol and in Section 4 we discuss the implications of IP virtualization on the system design and behavior. Section 5 evaluates the system. And, Section 6 summarizes the contribution and presents avenues for further research.

2 Related Work

The problem of supporting host mobility on the Internet has been extensively studied. MobileIP, like VIP, is designed to achieve transparent mobility by virtualizing the IP level of the protocol stack. Other approaches have been proposed at the IP, transport, and application levels of the protocol stack or have used names rather than physical addresses for routing.

2.1 MobileIP

MobileIP [13, 15] has many similarities with VIP. Both systems share the goal of providing transparent mobility. In both systems, this is achieved by decoupling the routing and naming roles that coexist in conventional IP addresses; in both systems a host receives two distinct identifiers - a permanent name, that does not change when a host moves, and a variable address, that changes to reflect the host's current point of attachment to the Internet.

There are two fundamental differences between MobileIP and VIP.

The first is the nature of a mobile host's permanent name: in Mobile IP the home address is a valid IP address to which packets can be routed, but in VIP the virtual address is a node's fully-qualified domain name (FQDN). For backwards compatibility with layers above VIP, VIP introduces 32-bit tokens that act as synonyms with FQDNs, but these VIP addresses have no semantic meaning within IP. This clean distinction between virtual and physical IP addresses makes deployment easier and cheaper for a user. Whereas VIP nodes automatically generate lightweight VIP address tokens, a user wishing to deploy a device using MobileIP must acquire a static IP address from an ISP.

The second fundamental difference is the mechanism used to map between home (or virtual) addresses and correspondent (or physical) addresses. MobileIP uses a home agent machine that receives packets addressed to a mobile node's home address and that tunnels packets to the mobile node's correspondent address. Instead, VIP uses DNS to map machine names to physical IP addresses. By using DNS for translation, VIP takes advantage of existing infrastructure rather than requiring deployment of new translation infrastructure. Furthermore, free third-party DNS servers such as no-ip.com and dyndns.org allow individual users to use VIP without even having to manage their own name servers.

Note that although route optimizations to some implementations of MobileIP modify both communicating end-stations to eliminate need for separate foreign agents [13] and to reduce triangle routing [14], these implementations still require home agents to establish and update the mappings cached at end stations. But also note that MobileIP's approach of using routable addresses and home agents provides one advantage over VIP: if a mobile node that has been modified to support MobileIP communicates with a fixed node that does not support MobileIP, MobileIP's home agent can forward packets from the fixed node as the mobile node moves. Conversely, although VIP retains backwards compatibility in that VIP and non-VIP nodes can communicate, VIP does not support mobility in such a scenario. Unfortunately, it appears that this advantage of MobileIP fundamentally requires that virtual addresses be routable, which we believe raises too high a barrier to deployment.

IPv6 [3] supports mobility using MobileIP techniques. However, it requires deployment of large amounts of new infrastructure.

2.2 Other approaches

Below, we discuss other approaches to mobility, organized by the level of the protocol stack at which virtualization is introduced: the IP layer, transport layer, or application layer. Finally, we discuss several other "name-centric" routing architectures.

Our VIP system shares the idea of virtualizing the IP layer to support mobility with an earlier system, also called VIP and introduced by Teraoka et. al [19]. Both systems strive to separate the logical name of a host from its changing physical IP address, but the systems differ significantly in their approach to achieving this goal. First, Teraoka's VIP depends on the permanent name of a host being equal to the host's initial physical IP address in its home network. In our VIP, a virtual IP address has no IP semantics and can be basically chosen arbitrarily. Second, Teraoka's VIP requires routers—including at least one on the home network of each host—to store the correct mapping between the host's logical name and its physical name. The home network router plays a role similar to that of the home agent in MobileIP.

Gupta and Reddy [9] propose a IP-level redirection mechanism that is similar to MobileIP with route optimization. The focus of this work is on anycast, but the technique can be applied to mobility as well.

Snoeren and Balakrishnan [18] propose an architecture for supporting mobility in TCP that, like VIP, is designed to minimize dependence on new infrastructure. This architecture relies on a peer-to-peer protocol to update address information when a node moves. Our approach differs in two ways. First, we implement mobility at the IP level rather than the transport level. Conceptually, we believe abstracting IP addresses directly above the IP layer is a simpler approach, and this approach has the practical advantage of allowing one implementation of virtualization to support a wide range of higher-level protocols, including TCP, RTP, ICMP, and UDP, and it supports straightforward integration with IPSec. Second, because we focus on supporting users with dozens of devices, we regard simultaneous movement of both ends of a connection as an important case—encountered, for instance, when a user carrying several devices exits a building—and we engineer our protocol to support it.

Several systems rely on applications to detect loss of connectivity and to switch to alternative or updated IP addresses associated with a target machine's name. This approach is most useful where application use is characterized by short transactions that may be retried if a network address changes. Smart clients [22] extend this approach by allowing servers to specify application-specific session fail-over code. Zhang and Dao [23] provide a user-level session abstraction to support automatic fail-over.

The idea of exposing names to applications and invisibly translating from name to physical address or route is a core idea in the Intentional Naming system [1] and TRIAD [2]. These systems, however, are more ambitious efforts to re-engineer the protocol stack. Intentional naming foregoes backwards compatibility, and both introduce translation to the routing infrastructure.

3 VIP architecture

The basic idea of VIP is simple; the VIP framework identifies a machine by its unique fully-qualified domain name (FQDN, e.g., example.acm.org), and the VIP layer on each machine maintains a mapping from FQDN to the physical IP addresses of peer machines so that it can direct messages addressed to an FQDN to that machine's current location – its current physical IP address. As IP addresses change due to migration, VIP updates this FQDN↔IP mapping using secure dynamic DNS. But, because FQDNs do not change, communication transparently continues across physical IP address changes.

Unfortunately, current applications use IP address as the basis for communication and the FQDN merely as a means of obtaining it. We maintain backwards compatibility by virtualizing IP through a layer of indirection. Thus each FQDN is mapped to a 32-bit token, which we call a virtual IP address, that in turn maps to the physical IP address. We refer to the former as the VIP address or virtual IP address and the latter as the IP address or physical IP address.

The VIP address is integrated into the system by a VIP layer that resides immediately above the IP layer. Layers above VIP see and work with virtual IP addresses, which are merely backwards-compatible synonyms for FQDNs, and layers below VIP see the physical IP addresses required to route packets to their intended destination. A separate user-level daemon maintains these FQDN↔VIP address and VIP address↔IP address mappings and updates the latter using dynamic DNS.

To simplify reasoning about security, the system uses IPSec to encrypt and authenticate all VIP communication. Following our goal of minimal infrastructure, this security scheme uses a simple peer-to-peer "anonymous" key exchange protocol similar to the one used in SSH [11].

Overall, this architecture provides a simple means of supporting mobility. Furthermore, it imposes little or no additional infrastructure requirements. Bind version 9 includes secure dynamic DNS updates [21], and many current operating systems, for example Linux and Windows 2000, ship with dynamic DNS support. Furthermore, a number of web-based, free or low-cost dynamic DNS services allow individual users to use dynamic DNS without running their own name servers. IPSec is similarly widely available, as is DHCP to allow mobile clients to get temporary IP addresses.

The key design issues for the VIP architecture lie in the creation and maintenance of the two mappings, namely FQDN↔VIP address and VIP address↔IP address. The former deals with how we choose and associate virtual tokens with unique names and avoid collisions in these mappings. The latter resolves connection migration. In what follows we describe the design of these two mappings and then complete the description of the protocol by describing how VIP-addressed packets are encapsulated in IP-addressed packets.

3.1 FQDN↔VIP address negotiation

The assignment of VIP tokens to FQDNs should satisfy the following four properties.

- *Unchanging and unique*. A VIP is a synonym for an FQDN that has these properties. Furthermore, the mapping should be collision-free with respect to real IP addresses because legacy machines will not include a protocol for resolving collisions.
- *Symmetric.* Communicating machines must agree on what they call one another. Thus if machine A maps $FQDN_A \leftrightarrow VIP_A$, machine B must also map $FQDN_A \leftrightarrow VIP_A$ when A and B communicate. Several higher layer protocols, including TCP, assume this property.
- *Scalable.* A mobile device will communicate with dozens or hundreds of servers and other mobile devices. And servers may communicate with millions of mobile devices.
- *Lightweight.* It should be easy and inexpensive to assign mappings to devices to make it easy for a user to deploy the system.

Unfortunately, it is difficult to satisfy these properties simultaneously with 32-bit tokens. For example, hashing the FQDN into the 28-bit class-E reserved IP address space meets all of these criteria except uniqueness – different FQDNs may map to the same VIP addresses (and these collisions will be too frequent to ignore due to the birthday paradox). Conversely, uniqueness could be assured with a more systematic assignment of VIP tokens to FQDNs (e.g., by extending the current, hierarchical IP assignment rules to include VIP assignment). Unfortunately, such a heavy-weight methodology has the same limitations as the current IP address assignment process.

We resolve this dilemma by relaxing the first requirement. In particular, VIP includes a negotiation phase that provides limited-duration VIP addresses that are pair-wise unique rather than permanent VIP addresses that are globally unique. Thus, VIP's naming semantics are weaker than current naming semantics in two ways: (1) an FQDN↔VIP mapping is not guaranteed to hold across reboots and (2) an FQDN↔VIP mapping may not be shared across machines. Few applications are affected by these weaker semantics, but some are. For example, web server log analysis might tacitly assume that IP addresses are permanent and global machine identifiers.

The negotiation protocol works as follows. If machine X wishes to communicate with machine Y, it makes a library call to gethostbyname($FQDN_Y$), and gethostbyname() queries the local VIP daemon, which is a DNS name daemon that we have modified to support VIP. If the VIP daemon already stores a VIP mapping for $FQDN_Y$, it returns VIP_Y. Otherwise, it issues two network DNS queries using the standard DNS protocol. The first query is for Y's DNS A (address) record. The second query is also for a DNS A record, but it asks for the address of the imaginary host *VIP_MAGIC_NUMBER*-$FQDN_Y$ (e.g., VIP105067072021-example.acm.org.)[2] The first query returns IP_Y, Y's current physical IP address. The second query returns either an error (if Y does not support VIP) or $VIP_{proposeY}$, a proposed VIP address with which to identify Y (if Y does support VIP). In the former case, backwards compatibility is retained because the VIP daemon simply returns Y's physical IP address. In the latter case, the VIP daemon running on X now contacts the VIP daemon running on Y to negotiate mutually acceptable VIP synonyms for $FQDN_X$ and $FQDN_Y$ using X's and Y's current physical IP addresses, IP_X and IP_Y, for communication. The negotiation protocol has the following steps.

1. X selects $VIP_{proposedX}$, decides if it will accept or reject the mapping $VIP_{proposedY}$, and sends the following message to IP_Y: [($Request$, $FQDN_X$, $VIP_{proposedX}$, $FQDN_Y$, $VIP_{proposedY}$, accept or reject $VIP_{proposedY}$)]
2. Upon receipt of this message Y first verifies that $FQDN_X$ maps to IP_X by querying DNS using $FQDN_X$. If the resulting IP address differs from IP_X, Y ignores this request. Otherwise, Y accepts or rejects the $FQDN_X \leftrightarrow VIP_{proposedX}$ mapping and

[2] A cleaner alternative would be to add a new DNS record type. We choose the magic number approach to make it easier to use third-party web-based DNS servers.

stores the mapping if it is accepted. Then, if X rejected Y's previous proposal, Y selects a new $VIP_{proposedY}$. In any event, Y then sends the following message to IP_X: [($Reply$, $FQDN_X$, $VIP_{proposedX}$, $FQDN_Y$, $VIP_{proposedY}$, accept or reject $VIP_{proposedX}$)]

3. Upon X's receipt of this message, if either X or Y rejected a proposed VIP address during the previous round, X initiates another round from step 1. Once both machines accept the proposals during the previous round, X's VIP daemon stores Y's mapping $FQDN_Y \leftrightarrow VIP_Y$ and returns VIP_Y to the gethostbyname() call that initiated the mapping.

At any point, either party may terminate the negotiation, causing X to fall back on physical IP addresses for communication. X does this by simply returning the physical IP address. Y does this by proposing address 0.0.0.0. To deal with lost messages and host mobility during this negotiation, if X does not receive a reply from Y within a timeout, it retransmits its last message. If X does not receive a reply to that retransmission, it restarts the protocol from the DNS network query.

A machine accepts an FQDN↔VIP mapping proposed by another machine if (a) the mapping is from the standard range of VIP addresses (we currently use the 28-bit reserved class-E range) and (b) the mapping is not currently in use for a different FQDN. Alternatively, if a proposed mapping is not from the standard VIP address range, the system does a reverse DNS lookup using the proposed VIP address and accepts the mapping if the resulting FQDN matches the proposal. This feature allows a machine to use a dedicated VIP address that no other machine can try to claim. As described in Section 4.2, this approach is one technique available to installations willing to pay for static IP addresses for resisting some denial of service attacks.

This negotiation typically adds one round trip time to connection set-up with VIP-enabled destinations. It also adds one additional DNS query to each host lookup. The primary purpose of this query is to allow systems to efficiently detect legacy, non-VIP clients; it thus eliminates the need for the VIP daemon to send a probe to the remote host to verify its VIP capability. The secondary purpose is to simplify the protocol by initializing $VIP_{proposedY}$. Note that, the additional DNS query is pipelined with the standard DNS query to minimize latency.

VIP nodes must carefully control garbage collection of FQDN $\leftrightarrow$ VIP address mappings to maintain the abstraction that VIP addresses are synonyms for FQDNs. We discuss garbage collection policies in Section 4.1.2. Regardless of the policy, garbage collection is a local decision, so if a machine A discards a mapping ($FQDN_B$, VIP_B), A may later receive a message from B. If a machine receives a VIP packet for which it has no FQDN↔VIP address mapping, the machine discards the packet and sends a negative acknowledgment message to the sender's VIP daemon. That daemon then executes the negotiation protocol described above—including the DNS query—to obtain a new mapping, using the previous mapping as the initial values for $VIP_{proposedA}$ and $VIP_{proposedB}$. If the protocol is unable to re-establish the same mapping for VIP_A, B's daemon marks the original VIP_A value as *invalid* and discards future packets sent to that VIP address. To maintain correct semantics, this address may not be re-used until it is garbage collected according to the system's standard VIP reuse rules. Note that A and B may continue to communicate using the new mapping, although applications on B using the old mapping will have their packets dropped.

Two sets of issues remain for understanding this protocol. First, end-hosts running this protocol have freedom to decide which VIP addresses to propose and when to garbage collect FQDN↔VIP address mappings. Second, the protocol must guard against malicious attacks. We discuss mapping policies and security in Section 4.

3.2 VIP↔IP mapping

The set of VIP address↔IP address mappings a machine stores can be thought of as a cache of mappings for the targets with which the machine communicates. This cache must be kept consistent with the true IP addresses of those targets. We use an invalidation protocol with leases [8] to accomplish this.

When a machine X's IP address changes, it sends invalidation hints (VIP_X, $newIP_X$) to the VIP daemons on its *Active Partner List*, the set of machines with which X has communicated during the previous T seconds. Conceptually, the invalidation hints signal the receiver to query DNS for the new mapping the next time it sends a message to X, but since X sends the hints via IPSec and includes the updated values, the receiver can trust them and update its mappings immediately.

There are two cases when a machine Y will not receive an invalidation when a machine it had been talking to, X, moves: (a) the lease has expired or (b) Y's IP address also changed. Therefore, any time a node Y sends a message to a node X from which Y has not received a packet from during the last T seconds, Y queries DNS to renew the mapping. In addition, if a machine does not receive an acknowledgment to an invalidation, it queries DNS to re-validate the mapping and then resends the invalidation hint.

3.3 Packet encapsulation

The VIP system encapsulates VIP-addressed packets in IP-addressed packets using IPIP encapsulation [12]. In our Linux prototype, outgoing IP packets pass through the IP routing table. If the destination address is a VIP address, it will fall into a range of addresses corresponding to a VIP interface and the packet will be handed to the VIP system's IP Encapsulation module. This module looks up the physical IP address corresponding to the VIP address and encapsulates the packet using this physical address. Now, when the packet passes through the routing table for the second time, its address corresponds to a physical IP interface, and it is sent on the corresponding physical network.

Incoming VIP packets arrive as IP packets whose next protocol flag indicates that they should be passed to the VIP de-encapsulation layer after IP processing. If these incoming packets' VIP address and IP address do not match the corresponding VIP address $\leftrightarrow$ IP address mapping stored by the VIP daemon, they are dropped. An implementation may send a negative acknowledgment in this case, but doing so is not necessary for correct migration. Assuming that the VIP $\leftrightarrow$ IP mapping matches, the packet is de-encapsulated and passed to the next higher level of the protocol stack.

3.4 IPSec integration

Our VIP implementation transmits all packets encrypted via IPSec, using the unchanging VIP address to identify the key needed to encrypt/decrypt packets.

In the common case, key management is similar to that used in the SSH-1 secure shell protocol [11]: nodes exchange public keys in the clear during the negotiation protocol described in Section 3.1. As with SSH, the approach is designed to provide a practical trade-off between good security and deployment with minimal infrastructure. Similar to SSH, this approach is vulnerable to man in the middle attacks during the initial key exchange but provides end-to-end security after that key exchange. We discuss the security properties of the system in more detail in Section 4.2.

Although we expect this "anonymous" peer-to-peer key exchange to be the most common mode of operation, the use of VIPs as an unchanging synonym for a machine name simplifies more systematic use of IPSec. In particular, our prototype supports key exchange via DNSSEC [20]. In this mode of operation, DNS provides a certificate chain that binds a public key to a FQDN. If such a certificate is provided, the VIP layer uses it rather than using anonymous peer-to-peer key exchange.

4 System Issues for IP virtualisation

This section discusses three ways in which system design is affected by the IP virtualization outlined in the above architecture. First, the system must manage the FQDN$\leftrightarrow$VIP address mappings carefully to maintain the abstraction that a VIP address is a synonym for a FQDN. In particular, the protocol leaves two policy decisions to implementations—local VIP selection and VIP garbage collection. We discuss these policies in Section 4.1. Second because the system allows transparent remapping of IP addresses, security must be carefully considered throughout the design. We discuss the system's end-to-end security and resistance to denial of service in Section 4.2. Third, VIP transparently remaps VIP addresses $\leftrightarrow$ IP addresses. For most applications this is useful, but for some applications and protocols, transparent remapping can cause problems. We discuss how we allow higher-level topology-aware protocols and applications to break this transparency in Section 4.3.

4.1 FQDN $\leftrightarrow$ VIP management

4.1.1 Local VIP selection policy

The protocol allows a machine to represent itself with any VIP address from a reserved range of values. Different implementations are free to choose these addresses in different ways. In our initial implementation, machines attempt to reuse a small number of VIP addresses. This is because for simplicity our Linux prototype uses an IP *interface* to represent each VIP alias for the local machine, and some modules above the IP layer—notably the FreeS/WAN 1.8 implementation of IPSec that we use—assume a small number of interfaces.

In Linux and most Unix systems, the interface data structure represents a physical network connection with a particular IP address (e.g., an Ethernet card), and we reuse these facilities for our implementation of VIP. Each machine randomly selects a fixed number of local VIP addresses and creates corresponding interfaces. After negotiating VIP mappings to communicate with a destination machine, the destination VIP address is added to the routing table for the corresponding local VIP address' interface so that outgoing packets sent to that destination VIP address are marked with the agreed upon source VIP address and are sent via the VIP protocol.

Although this approach is simple, restricting the number of VIP addresses a machine can use to refer to itself renders our prototype vulnerable to denial of service attacks in which a machine A attempts to communicate with another machine B where machine B has already communicated with other machines that have "claimed" the VIP addresses that A wishes to use. We believe this vulnerability is a good trade-off for the simplicity of our implementation. Furthermore, it appears relatively straightforward to dynamically allocate interfaces as needed as long as higher protocol layers make no a priori assumptions about the maximum number of interfaces a machine can have. We discuss this denial of service issue in more detail in Section 4.2.1.

4.1.2 VIP garbage collection policy

Because the VIP address space is smaller than the FQDN space, machines must garbage collect their FQDN$\leftrightarrow$VIP address mappings to limit the rate of unresolvable collisions. Unfortunately, the current mapping of FQDN$\leftrightarrow$physical IP address does not have a well-defined consistency model, so it is not clear when the VIP layer can safely re-use a mapping for a different FQDN. For example, it is possible that an application may send a packet to a VIP address that was resolved from an FQDN many days ago or after the connection has been idle for hours. Without a well-defined and widely-used consistency model for applications' FQDN$\leftrightarrow$IP mappings, *any* re-use of VIP addresses for different FQDNs has the potential to break the abstraction that a VIP address is a synonym for an FQDN. A VIP implementation must balance the risk of deleting a needed mapping against the reduced risk of unresolvable collision that comes from keeping the number of stored mappings small.

Note that this problem is not new to VIP. Current applications that use IP addresses long after resolving them risk sending packets to the wrong machine. In practice, this is rarely a problem, and we do not expect this to be a significant issue for VIP. Our goal is therefore to build a system that is simple, that works well with legacy applications, and that has well-defined semantics on which future applications can build.

Our prototype implements a *reclaim-on-reboot* policy: FQDN$\leftrightarrow$VIP address mappings are guaranteed to remain valid until a machine reboots. This approach should work with all applications except those that write IP addresses to disk and use them later. A disadvantage of this conservative garbage collection policy is that it may result in larger lookup tables and more collisions than needed.

A more aggressive policy that could be considered by some implementations is *lease*: each machine maintains an LRU list of VIP address mappings and discards elements that have not been used for T seconds, where T is chosen to be long enough that legacy applications are unlikely to use discarded mappings and to be short enough to limit the size of the FQDN$\leftrightarrow$VIP address table. Applications that wish to re-use a mapping longer than T seconds after its last use should re-validate the mapping with a DNS lookup.

4.1.3 Future directions: IPv6 and beyond

The concept of connecting to a name rather than an address is a simple way to support mobility. The complexity in maintaining a FQDN$\leftrightarrow$VIP address mapping comes from our desire remain compatible with 32-bit IPv4 applications. The short IPv4 address space appears to make the problems that arise fundamental, but it also appears that reasonable engineering compromises for negotiating, choosing, and garbage collecting these mappings can work well.

The VIP principles are even more attractive for applications and higher-level protocols that do not assume 32-bit IPv4 addresses. For example, 128-bit IPv6 addresses [3] might be constructed directly using a 128-bit MD5 hash [17] rather than requiring negotiation and table lookup. Or, systems could run using "native VIP" addresses: the “address” returned by gethostbyname() would simply be the target machine’s FQDN.

4.2 Security

The use of IPSec simplifies reasoning about VIP’s end-to-end security because IPSec provides end-to-end protection against attacks on the VIP system. After the initial key exchange, almost all traffic and protocol messages between nodes travel via IPSec. The one exception is the negative acknowledgments sent upon receipt of packets addressed to unmapped VIP addresses (see Section 3.1). So at worst, errors in packet encapsulation or the FQDN $\leftrightarrow$ VIP $\leftrightarrow$ IP mappings after the key exchange can result in denial of service; they can not result in delivery of data to/from an incorrect node.

Following the organization of Section 3, we analyze system security by examining the FQDN $\leftrightarrow$ VIP mapping, the VIP $\leftrightarrow$ IP mapping, and packet encapsulation.

4.2.1 FQDN$\leftrightarrow$VIP mapping

There are three attacks on the FQDN$\leftrightarrow$VIP mapping: (1) *negotiation:* an adversary can attempt to modify the initial negotiation, (2) *consistency:* an adversary can attempt to reuse a garbage collected VIP address, and (3) *VIP consumption:* an adversary can attempt to deny service by consuming available VIP addresses.

Negotiation. To minimize the demands on infrastructure, the system supports peer-to-peer “anonymous” IPSec key exchange. As with SSH, the approach is designed to provide a practical trade-off between good security and deployment with minimal infrastructure. Similar to SSH, this approach is vulnerable to man in the middle attacks during the initial key exchange – either by spoofing traffic to/from the DNS server or by spoofing traffic between the communicating parties, but the approach provides end-to-end security after that key exchange. As noted in Section 3.4, users can configure their DNS servers to supply certificate chains binding keys to FQDNs to address this limitation.

Consistency. The lack of a clear consistency model for legacy DNS $\leftrightarrow$ IP address mappings introduces the possibility of errors when an application uses a FQDN $\leftrightarrow$ VIP address mapping that is no longer valid. If an application attempts to send a packet using a VIP address after it has been locally garbage collected and reused, that packet will be sent to an unintended machine. Note that IPSec does not protect against this failure because the VIP address identifies the IPSec key to be used for a connection. Our solution is to constrain garbage collection of mappings according to a consistency model that (a) precisely defines semantics to allow careful application writers to ensure correct behavior and (b) provides conservative default behavior that results in correct behavior for most legacy or less carefully written applications. We have chosen to implement a reclaim-on-reboot policy in our prototype to meet these requirements.

Note that even without VIP, any application that communicates with nodes that use DHCP for dynamic IP address assignment is similarly vulnerable. In principle, all such applications should contact the node with which they are communicating to determine the length of the node’s DHCP lease. In practice few, if any, applications currently do this. DHCP appears to be a successful application of the two principles defined above – precisely defined behavior and conservative default behavior.

VIP consumption. In order for A and B to communicate, A must claim an unused VIP address in B’s FQDN $\leftrightarrow$ VIP table. If B’s table is full or if A is restricted in which addresses it can claim and those entries are full, then it is possible to deny VIP service between A and B.

Two implementation decisions in our prototype increase its vulnerability to such unresolvable collisions. First, as noted in Section 4.1.1, our implementation restricts each node to using a few VIP addresses to refer to itself. Note that future VIP implementations may address this issue by dynamically allocating local VIP addresses as needed. Second, as noted in Section 4.1.2, our implementation only garbage collects table entries at reboot. Its table may therefore be more full than absolutely necessary. Note that more aggressive garbage collection could be implemented; however as described earlier in this section, more aggressive garbage collection would increase the risk of applications using stale mappings.

VIP is designed to minimize the risk of random failures due to these collisions, to minimize the impact of such collisions when they (deliberately or accidently) occur, and to provide an option to eliminate the risk of collisions for machines willing to invest in additional infrastructure:

- Random collisions are rare. If a node randomly chooses N identifiers from the available 2^{28} identifiers and talks to a machine that has previously stored M mappings for other machines, the odds of all N addresses already being claimed are about $(\frac{M}{2^{28}})^N$. In the common case of peer-to-peer communication between mobile devices with two local identifiers and fewer than 1000 stored entries, the risk of unresolvable collision is less than 2^{-35} (that is, less than 2^{-36} for each of two machines). Communication with a popular server is more likely to encounter a collision. If a machine with two local identifiers tries to contact a server that stores 2^{20} entries, there is a 2^{-16} chance that both of the machine's addresses are already in use by the server to refer to other machines.
- The consequences of collision are limited. In our current prototype, an unresolvable collision causes machines to fall back on communicating with physical IP addresses. Support for mobility is lost in this case, but communication is still possible. In the future, we plan to enhance the protocol to deal with unresolvable collisions by falling back on "1-sided" VIP where one machine A uses its physical IP address as a VIP address. That physical IP address would be entered into the remote machine B's FQDN $\leftrightarrow$ VIP address table, but would be marked "temporary." B could move, changing A's $VIP_B \leftrightarrow IP_B$ mappings, but if A moves, the temporary mapping VIP_A would be discarded by B. This optimization may be useful for well-known, widely-accessed services which are at greater risk of denial-of-service and which are not likely to move.
- Machines may reserve static IP addresses to use as their VIP addresses to eliminate the risk of collision. As described in Section 3.1, the protocol uses reverse DNS lookup to ensure that only one node may claim a given static IP address as its VIP address.[3] Reserving and configuring static IP addresses, of course, requires additional installation effort and expense, so we do not believe it is appropriate for most VIP nodes. But, this configuration seems useful for well-known, widely-accessed services, which are at greater risk of accidental or deliberate denial of service and for which this additional effort is not likely to be burdensome.

[3] More precisely, this protocol ensures that the DNS domain in question controls re-use of a static IP address used as a VIP address.

4.2.2 VIP $\leftrightarrow$ IP mapping

VIP allows transparent remapping of the IP address to which a name (FQDN or VIP address) refers. There are two types of attack on this mapping: (1) modifying or introducing spurious updates and (2) preventing needed updates from occurring.

VIP is vulnerable to an adversary that can modify communication with DNS or that can modify or insert IP packets during the negotiation phase. Furthermore, if an adversary can prevent delivery of mapping invalidation messages, a node may continue to send packets to a stale IP address. In these cases, after the initial key exchange the use of IPSec prevents an adversary from reading (or forging) messages to (from) the old address, limiting damage to denial of service attacks.

These vulnerabilities are similar those of standard IP. Both systems rely on DNS to provide a correct IP address for a name and rely on the initial set of routers to transmit traffic without modification. VIP's mobility adds additional DNS lookups and additional routers to the mix, but VIP's use of IPSec limits the damage that can be caused by these additional dependencies.

4.2.3 Encapsulation

Because the address exposed to applications – the VIP address – is encapsulated in IP packets, IP router ingress filtering [7] does not prevent address spoofing: it is easy for a sender to insert any "from" VIP address in any packet. The system guards against this at two levels. First, if an incoming packet's IP address does not match the stored IP address for the VIP address from which the packet purports to come, the VIP layer drops the packet. Second, the IPSec layer discards incoming packets that were not encrypted by the encryption key corresponding to the VIP address claimed by the packet.

4.2.4 IPSec costs and benefits

We run VIP over IPSec rather than devising our own authentication protocols. One might argue that encrypting all traffic is more expensive than necessary, and we considered alternatives such as encrypting or authenticating only VIP-control traffic. But, we chose to encrypt VIP's data traffic as well because (a) the approach is simple, (b) processors are fast – even most palm-top computers can encrypt at rates approaching or exceeding their network bandwidths, and (c) the approach can detect and limit the damage of DNS spoofing (after the initial lookup), stale VIP $\leftrightarrow$ IP mappings (see Section 4.2.1 and 4.2.2), and attacks that spoof both the IP and VIP addresses in a packet (see Section 4.2.3).

We also believe that strong default security is prudent for systems striving to provide transparent mobility with minimal infrastructure. Mobile devices may face more risks than fixed devices; for example, they may use radio broadcast to communicate and they may encounter routers, DHCP servers, and other infrastructure provided by unknown or untrusted parties. Encryption of all VIP traffic supports transparency by providing additional protection for legacy applications and system configurations that are "transparently" brought into this more hostile environment. And, encryption of all VIP traffic supports minimal infrastructure by reducing the level of trust mobile nodes put in the infrastructure in which they function.

4.3 Topology-aware applications

In general, we believe that applications and protocols above IP generally treat IP addresses as machine identifiers (even though they technically "shouldn't") and applications below IP treat IP as identifying a network connection identifier for routing. VIP splits IP into two layers to separate these roles and thereby support transparent mobility. A few applications and protocols, however, explicitly or implicitly rely on the routing information conveyed by IP addresses. These topology-aware applications and protocols must be able to break VIP's transparent VIP $\leftrightarrow$ IP remapping.

For example, an application-level anycast algorithm [6] might wish to compare IP addresses to routing table entries to identify topologically nearby machines. As another example, TCP's congestion control algorithm assumes that the path between nodes is static; if the route changes because a node moves to a new network connection, TCP should adjust its congestion control state (e.g., by setting the congestion control window to 1 and entering slow-start).

To support topology-aware applications, VIP implementations should provide two new interfaces: a

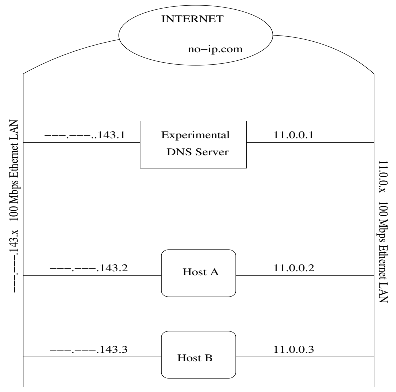


Figure 1: The evaluation testbed.

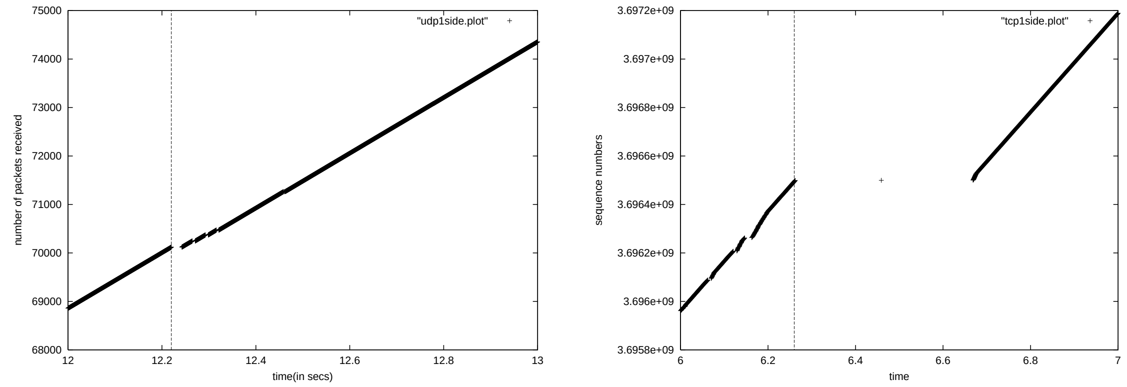
`p_gethostbyname()` interface that returns the physical IP address of a node given that node's name and a `callOnChange()` interface to register a callback for when a VIP address $\leftrightarrow$ IP address mapping changes. In addition, kernels implementing VIP should modify their TCP and other transports to register for and react to such callbacks. Note that our prototype does not yet include these interfaces or the congestion control callbacks.

Snoeren et. al [18] argue that the need of congestion control algorithms to react to changing routes means that IP mobility remapping should be done at the transport layer rather than the IP layer. We believe that callbacks provide a simpler and more general solution. Both approaches require modification of all transport protocols' congestion control code to add logic to handle an address change. But our approach allows us to instantiate the name-to-route translation code once—above the IP routing layer—and use it for many different transport layers. As described in Section 5, below, we have successfully run a wide range of protocols including TCP, UDP, ICMP, and RTP above our VIP layer.

5 Evaluation

Our prototype uses the Linux 2.2.17 kernel and modifies the IPIP Encapsulation layer to implement VIP. We use FreeS/WAN 1.8 for IPSec. The modified name daemon, VIPD daemon, is currently implemented in Perl and runs at user level.

Our test bed is shown in Figure 1. Hosts A and B are 933 MHz Pentium III machines with 256MB of RAM. Each one of these machines has two 100Mbps Ethernet cards, which are connected to different LANs. The RTT measured by PING requests has an average 250μs between the hosts when they are communicating on same



(a) (b)

Figure 2: One-sided switch for (a) UDP and (b) TCP connection.

LAN and 450μs when communicating through interfaces on different LANs. The mobility of the hosts is emulated by deactivating of the interfaces and activating the other interface through the *ifconfig* command.

A third machine of similar hardware configuration is used for running *named*, the domain name server, a part of bind 9.1.1 from `www.isc.org`. This version supports signed dynamic updates [5]. We create a domain `vipip.net` and assign each host a name from this domain. Each host also shares a secret key with the DNS server for dynamic update requests. This node also acts as a router between the two testbed LANs.

We have succesfully tested our system for numerous applications running on both hosts that communicate using various protocols. These include a telnet session over TCP, RealPlayer video streaming over RTP and UDP, and ping over ICMP. Communication between application on two hosts continues successfully for both the cases of switching interfaces on one host (one-sided switch) and simultaneously switching interfaces on both hosts (two-sided switch). We have also successfully tested our system using *no-ip.com* as the name server by assigning our hosts names from the *no-ip.com* domain and configuring the hosts to send the dynamic DNS update requests to *no-ip.com*'s DNS server.

For measuring switching times, we implement a simple application with A continuously sending dummy messages to B using either UDP or TCP. We collect the communication traces using *tcpdump*. For TCP, we plot the sequence number of each packet against the time it was received. For UDP, we plot the number of UDP packets received versus time.

In Figure 2, we plot the traces for the one-sided switch case for TCP and UDP. For UDP, the observed switching time is about 23ms, including the time to send an update to the peer ($\sim$20ms) and acknowledgment time ($\sim$3ms). For TCP, switching time is about 400ms. This is longer than UDP because of a TCP retransmission timeout (200ms) and a *delayed ack* timeout (200ms) [16]. Figure 3 plots the behavior of UDP transmission for the two-sided mobility case. The switching time is about 644ms which includes a 500ms timeout by the VIP daemon waiting for the ack of the hint it sends to the peer's VIP daemon. After the timeout, the VIP daemon does a DNS lookup and then sends the update hint to the other host at the correct physical address.

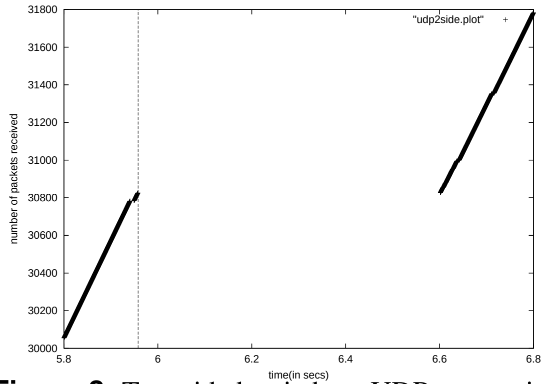


Figure 3: Two-sided switch on UDP connection.

In Figure 4, we look at the dependence of switching time on DNS lookup latency for two-sided switching in both TCP and UDP modes of communication. The delays were simulated by inserting an artificial *usleep* into the local VIP daemon. As expected, UDP switching time increases almost linearly with latency. However, TCP exhibits a stepwise exponential curve correspond-

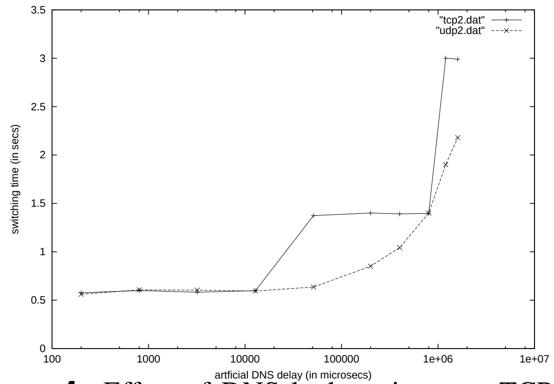


Figure 4: Effect of DNS lookup times on TCP and UDP two-sided switch case

ing to an exponential back off in retransmission.

Overall, we find that performance is good. For the two-sided switch case, fail-over time is dominated by the hint timeout, the round trip time to the DNS server, and the round trip time to the peer. Peer-to-peer updates appear to be a useful optimization. For the one-sided case, time is dominated by the round trip time between the peers and, for nearby peers, the overhead of our Perl-based name daemon.

6 Conclusion

VIP applies the principle of virtualization to IP addresses because exposing physical IP addresses to applications thwarts mobility and dynamic IP assignment, factors which will continue to grow in importance as mobility becomes more common, as users own increasing numbers of devices, and as the limited IPv4 address space is consumed. From the point of view of applications on VIP-enabled hosts, DNS names are used to address network traffic, and physical IP addresses are hidden.

A key benefit of VIP's implementation of virtualization is that its design emphasizes incremental deployability. Powerful building blocks for virtualization now exist so that virtualization can be done almost entirely by relying on existing infrastructure. As a result, VIP is simple enough to deploy that, for example, a Linux hobbyist could easily deploy and benefit from the system.

Acknowledgments

Sumit Garg and Anupam Rastogi were involved in the design of an earlier version of this system. We thank them for their efforts.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Symposium on Operating Systems Principles*, pages 186–201, 1999.

[2] D. Cheriton and M. Gritter. TRIAD: A new next generation Internet architecture, 2000.

[3] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). Request for Comments 1883, Network Working Group, December 1995.

[4] R. Droms. *Dynamic Host Configuration Protocol*. IETF, October 1983. RFC 1531.

[5] D. Eastlake. Secure Domain Name System Dynamic Update. Technical Report RFC-2137, Internet Engineering Task Force, Apr 1997.

[6] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In *Proceedings of IEEE Infocom*, March 1998.

[7] P. Ferguson and D. Senie. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. IETF, January 1998. RFC 2267.

[8] C. Gray and D. Cheriton. Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.

[9] S. Gupta and N. Reddy. A Client Oriented IP Level Redirection Mechanism. In *Proceedings of INFOCOM 99*. IEEE, March 1999.

[10] S. Kent and R. Atkinson. "Security Architecture for the Internet Protocol". Technical Report RFC-2401, Internet Engineering Task Force, Nov 1998.

[11] OpenSSH. http://www.openssh.com.

[12] C. Perkins. *IP Encapsulation within IP*. IETF, October 1996. RFC 2003.

[13] C. Perkins. IP Mobility Support. RFC 2002, IETF, October 1996.

[14] C. Perkins, A. Myles, and D. Johnson. The Internet Mobile Host Protocol (IMHP). In *Proceedings of INET*, June 1994.

[15] C. E. Perkins and A. Myles. Mobile IP. *Proceedings of International Telecommunications Symposium*, pages 415–419, 1994.

[16] Ed. R. Braden. *Requirements for Internet Hosts – Communication Layers*. IETF, October 1989. RFC 1122.

[17] R. Rivest. The MD5 Message-Digest Algorithm. Request for Comments 1321, Network Working Group, ISI, April 1992.

[18] A. C. Snoeren and H. Balakrishnan. An End-to-End Approach to Host Mobility. In *Proc. 6th International Conference on Mobile Computing and Networking (MobiCom)*, 2000.

[19] F. Teraoka, K. Uehara, H. Sunahara, and J. Murai. VIP: A Protocol Providing Host Mobility. *Communications of the ACM*, 37(8):67–75, August 1994.

[20] B. Wellington. *Domain Name System Security (DNSSEC) Signing Authority*. IETF, November 2000. RFC 3008.

[21] B. Wellington. *Secure Domain Name System (DNS) Dynamic Update*. IETF, November 2000. RFC 3007.

[22] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using Smart Clients to Build Scalable Services. In *Proceedings of the 1997 USENIX Technical Conference*, January 1997.

[23] Y. Zhang and S. Dao. A "Persistent Connection" Model for Mobile and Distributed Systems. In *The 4th International Conference on Computer Communications and Networks (IC-CCN)*, Las Vegas, NV, September 1995.