An End-to-End Approach to Host Mobility

Alex C. Snoeren and Hari Balakrishnan
MIT Laboratory for Computer Science
Cambridge, MA 02139
{snoeren, hari}@lcs.mit.edu

Abstract

We present the design and implementation of an end-to-end architecture for Internet host mobility using dynamic updates to the Domain Name System (DNS) to track host location. Existing TCP connections are retained using secure and efficient connection migration, enabling established connections to seamlessly negotiate a change in endpoint IP addresses without the need for a third party. Our architecture is secure—name updates are effected via the secure DNS update protocol, while TCP connection migration uses a novel set of *Migrate* options—and provides a pure end-system alternative to routing-based approaches such as Mobile IP.

Mobile IP was designed under the principle that fixed Internet hosts and applications were to remain unmodified and only the underlying IP substrate should change. Our architecture requires no changes to the unicast IP substrate, instead modifying transport protocols and applications at the end hosts. We argue that this is not a hindrance to deployment; rather, in a significant number of cases, it allows for an easier deployment path than Mobile IP, while simultaneously giving better performance. We compare and contrast the strengths of end-to-end and network-layer mobility schemes, and argue that end-to-end schemes are better suited to many common mobile applications. Our performance experiments show that handoff times are governed by TCP migrate latencies, and are on the order of a round-trip time of the communicating peers.

1 Introduction

The proliferation of mobile computing devices and wireless networking products over the past decade has made host and service mobility on the Internet an important problem. Delivering data to a mobile host across a network address change without disrupting existing connections can be tackled by introducing a level of indirection in the routing system. This is the approach taken by Mobile IP [27, 29], which deploys a home agent that intercepts packets destined for a host currently away from its home network, and delivers it to the mobile host via a foreign agent in the foreign network. This approach does not require any changes to the fixed (correspondent) hosts in the Internet, but does require changing the underlying IP substrate to effect this *triangle* routing, and an authentication protocol to ensure that connections are not hijacked by a malicious party. Mobile IP is a "pure" routing solution, a network-layer scheme that requires no changes to any higher layer of the Internet protocol stack.

There are many classes of mobile applications [16]: those where other hosts originate connections to a mobile host and can benefit from both host location and handoff support (e.g., a mobile Web server, mobile telephony); those where the mobile host originates all connections, which do not require host location services but can benefit from handoff support (e.g., mail readers, Web browsers); and those where an application-level retry suffices if the network address changes unexpectedly during a short transaction, which need neither to work well (e.g., DNS resolution). We believe that a good end-to-end architecture for host mobility will support all these modes, and empower applications to make the choice best suited to their needs. Our architecture is motivated by, and meets, this goal. It is an end-to-end approach; no changes to the IP substrate are required.

In our mobility architecture, the decision of whether to support transparent connectivity across network address changes (especially useful for mobile servers) or not (not needed for short client-server transactions) is left to the application. While Mobile IP-style, fully-transparent mobility support is general and sufficient for mobile applications, this generality comes at significant cost, complexity, and performance degradation.

To locate mobile hosts as they change their network attachment point, we take advantage of the widely-deployed Domain Name System (DNS) [20] and its ability to support secure dynamic updates [8, 35]. Because most Internet applications resolve hostnames to an IP address at the beginning of a transaction or connection, this approach is viable for initiating new sessions with mobile hosts. When a host changes its network attachment point (IP address), it sends a secure DNS update to one of the name servers in its home domain updating its current location. The name-to-address mappings for these hosts are uncacheable by other domains, so stale bindings are eliminated.

The ability to support continuous communication during periods of mobility without modifying the IP substrate is a more challenging problem. Because TCP is a connection-oriented reliable protocol,

This research was supported in part by DARPA (Grant No. MDA972-99-1-0014), NTT Corporation, and IBM. Alex C. Snoeren is supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

MobiCom 2000 08/2000 Boston, MA

many TCP applications reasonably expect this service model in the face of losses and transient link failures, route changes, or mobility. The two communicating peers must securely negotiate a change in the underlying network-layer IP address and then seamlessly continue communication. Furthermore, an approach that *requires* either communicating peer to learn about the new network-layer address before a move occurs is untenable because network-layer moves may be quite sudden and unpredictable.

We design a new end-to-end TCP option to support the secure migration of an established TCP connection across an IP address change. Using this option, a TCP peer can suspend an open connection and reactivate it from another IP address, transparent to an application that expects uninterrupted reliable communication with the peer. In this protocol, security is achieved through the use of a connection identifier, or *token*, which may be secured by a shared secret key negotiated through an Elliptic Curve Diffie-Hellman (ECDH) key exchange [36] during initial connection establishment. It requires no third party to authenticate migration requests, thereby allowing the end points to use whatever authentication mechanism they choose to establish a trust relationship. Although we only describe details for TCP migration, we find that this idea is general and can be implemented in a like manner for specific UDP-based protocols such as the Real-time Transport Protocol (RTP) to achieve seamless mobility for those protocols as well.

One way of thinking of our work is in the context of the end-to-end argument [32], which observes that functionality is often best implemented in a higher layer at an end system, where it can be done according to the application's specific requirements. We show that it is possible to implement mobility as an end-to-end service without network-layer support, while providing multiple mobility modes. In this sense, this is akin to applications deciding between UDP and TCP as a transport protocol; many opt for UDP's simplicity and timeliness over TCP's reliability. In the same fashion, applications should be able to select the mobility mode of their choice.

The other significant advantage of handling mobility on an end-to-end basis is that it enables higher layers like TCP and HTTP to learn about mobility and adapt to it. For example, it is a good idea after a network route change to restart TCP transmissions from slow start or a window-halving [13] since the bottleneck might have changed, or adapt the transmitted content to reflect new network conditions. These optimizations can be made naturally if mobility is handled end-to-end, since no extra signalling is needed. Indeed, the large body of work in mobile-aware applications [15, 22, 25] can benefit from our architecture.

Experience with previous end-to-end enhancements such as various TCP options (e.g., SACK [19]), path MTU discovery, HTTP/1.1, etc., has shown that such techniques often meet with less resistance to widespread deployment than changes to the IP substrate. This supports our belief that, in addition to the flexibility it offers, an end-to-end approach may be successfully deployed.

We have implemented this mobility architecture in Linux 2.2 and have conducted several experiments with it. We are encouraged by the ease with which seamless mobility can be achieved, the flexibility it provides, and the lack of performance degradation. Since our scheme does not impose any triangle routing anomalies, end-to-end latency for active connections is better than standard Mobile IP, and similar to Mobile IP with route optimization.

The rest of this paper describes the technical details of our approach. In Section 2, we survey related work in the area of mobility support. We describe our architecture in Section 3, and detail our new Migrate TCP option in Section 4. We discuss the security ramifications of our approach in Section 5 and our implementation and performance results in Section 6. We address some deployment issues in Section 7 and conclude in Section 8.

2 Related work

The problem of Internet host mobility has been approached from many angles in the literature, but they can be classified into two categories. Some techniques attempt to handle host relocation in a completely transparent fashion, hiding any changes in network structure from the end hosts. We term these techniques *network-layer* mobility. By contrast, many other approaches attempt to handle relocation at a higher level in the end host.

2.1 Network-layer mobility

Mobile IP (RFC 2002) [29] is the current IETF standard for supporting mobility on the Internet. It provides transparent support for host mobility by inserting a level of indirection into the routing architecture. By elevating the mobile host's *home address* from its function as an interface identifier to an *end-point identifier* (EID), Mobile IP ensures the delivery of packets destined to a mobile host's home address, independent of the host's physical point of attachment to the Internet, as reflected in its *care-of address*. Mobile IP does this by creating a routing tunnel between a mobile host's home network and its care-of address.

Such routing tunnels need to be implemented with care because advertising explicit host routes into the wide-area routing tables destroys routing scalability. Mobile IP uses a *home agent* physically attached to the mobile host's home network to intercept and tunnel packets to the mobile host. Hence, packets undergo *triangle* routing, which is often longer than the optimal unicast path.

Further compounding the problem is the widespread deployment of ingress filters [9], ratified in February 2000 by the IETF as a "Best Current Practice" to combat denial-of-service attacks. With this mechanism, a router does not forward packets with a source address foreign to the local network, which implies that a packet sent by a mobile host in a foreign network with its source address set to its home address will not be forwarded. The solution to this is to use *reverse tunneling*, which tunnels packets originating at the mobile host first to the host's home agent (using the host's care-of address as a source address), and then from there on to the destination using the home address as the source address. Thus, routing anomalies occur in both directions.

Perkins and Johnson present a route optimization option for Mobile IP to avoid triangle routing [28]. Here, correspondent hosts cache the care-of address of mobile hosts, allowing communication to proceed directly. It requires an authenticated message exchange from the home agent to the correspondent host [26]. The resulting Mobile IP scheme achieves performance almost equivalent to ours, but requires modifications to the end hosts' IP layer[1] as well as the

[1]In fact, the draft allows on-path routers to cache the care-of addresses instead of the end host, but this requires modifying yet another level of infrastructure.

infrastructure. In contrast, our approach achieves secure, seamless connection migration without a third-party home agent. It also provides a mobile host the ability to pick a mobility mode based on the needs of its applications.

IPv6 provides native support for multiple simultaneous host addresses, and Mobile IPv6 provides mobility support for IPv6 in much the same fashion as Mobile IP for IPv4. IPv6 extensions allow for the specification of a care-of address, which explicitly separates the role of the EID (the host's canonical IP address) and routing location (the care-of address). Gupta and Reddy propose a similar redirection mechanism for IPv4 through the use of ICMP-like control messages which establish care-of bindings at the end hosts [10].

Mysore and Bharghavan propose an interesting approach to network-layer mobility [23], where each mobile host is issued a permanent Class D IP multicast address that can serve as a unique EID. If multicast were widely deployed, this is a promising approach; because a Class D EID has the benefit of being directly routable by the routing infrastructure, it removes the need for an explicit care-of address. However, this scheme requires a robust, scalable, and efficient multicast infrastructure for a large number of sparse groups.

2.2 Higher-layer methods

The home-agent-based approach has also been applied at the transport layer, as in MSOCKS [18], where connection redirection was achieved using a split-connection proxy.

The general idea of using names as a level-of-indirection to handle object and node mobility is part of computer systems folklore. For some years now, people have talked about using the DNS to effect the level-of-indirection needed to support host mobility, but to our knowledge ours is the first specific and complete architecture that uses the DNS to support Internet host mobility. Recently, Adjie-Winoto *et al.* proposed the integration of name resolution and message routing in an Intentional Naming System to implement a "late binding" option that tracks highly mobile services and nodes [1], and it seems possible to improve the performance of that scheme using our connection migration approach.

Our approach differs fundamentally from EID/locator techniques since it requires no additional level of global addressing or indirection, but only a (normally pre-existing) DNS entry and a shared connection key between the two end hosts. Furthermore, unlike previous connection-ID draft proposals such as Huitema's ETCP [11] for TCP connection re-addressing, it requires no modification to the TCP header, packet format, or semantics.[2] Instead, it uses an additional TCP option and the inserts an additional field into the Transmission Control Block (TCB).

There is a large body of work relating to improving TCP performance in wireless and mobile environments [5, 6]. While not the focus of our work, our adherence to standard TCP semantics allows these schemes to continue to work well in our architecture. Furthermore, since end hosts are explicitly notified of mobility, significant performance enhancements can be achieved at the application level [25].

[2] Special RST handling is required on some networks that may rapidly reassign IP addresses; Section 4.5 discusses this issue.

3 An end-to-end architecture

In this section, we describe our end-system mobility architecture. There are three important components in this system: addressing, mobile host location, and connection migration. By giving the mobile host explicit control over its mobility mode, we remove the need for an additional (third-party) home-agent to broker packet routing. The DNS already provides a host location service, and any further control is managed by the communicating peers themselves, triggered by the mobile host when it changes network location.

We assume, like most mobility schemes, that mobile hosts do not change IP addresses more than a few times a minute. We believe this is a reasonable assumption for most common cases of mobility. We emphasize that this does not preclude physical mobility at rapid velocities across a homogeneous link technology, since that can be handled at the physical and link layers, e.g., via link-layer bridging [12].

The rest of this section discusses addressing in a foreign network and host location using the DNS. Section 4 is devoted to a detailed description of TCP connection migration.

3.1 Addressing

The key to the scalability of the Internet architecture is that the IP address serves as a routing locator, reflecting the addressee's point of attachment in the network topology. This enables aggregation based on address prefixes and allows routing to scale well. Our mobility architecture explicitly preserves this crucial property of Internet addressing.

Like Mobile IP, we separate the issues of obtaining an IP address in a foreign domain from locating and seamlessly communicating with mobile hosts. Any suitable mechanism for address allocation may be employed, such as manual assignment, the Dynamic Host Configuration Protocol (DHCP) [7], or an autoconfiguration protocol [34].

While IP addresses fundamentally denote a point of attachment in the Internet topology and say nothing about the identity of the host that may be connected to that attachment point, they have implicitly become associated with other properties as well. For example, they are often used to specify security and access policies as in the case of ingress filtering to alleviate denial-of-service attacks. Our architecture works without violating this trust model and does not require any form of forward or reverse tunneling to maintain seamless connectivity. In a foreign network, a mobile host uses a locally obtained interface address valid in the foreign domain as its source address while communicating with other Internet hosts.

3.2 Locating a mobile host

Once a mobile host obtains an IP address, there are two ways in which it can communicate with correspondent hosts. First, as a client, when it actively opens connections to the correspondent host. In this case, there is no special host location task to be performed in our architecture; using the DNS as before works. However, if the mobile host were to move to another network attachment point during a connection, a new address would be obtained as described in the previous section, and the current connection would continue seamlessly via a secure negotiation with the communicating peer as described in Section 4. If a mobile host were always a client (not an uncommon case today), then no updates need to be made to any third party such as a home agent or the DNS.

To support mobile servers and other applications where Internet hosts actively originate communication with a mobile host, we use the DNS to provide a level of indirection between a host's current location and an invariant end-point identifier. In Mobile IP, a host's home address is the invariant, and all routing (in the absence of route optimization) occurs via the home agent that intercepts packets destined to this invariant. Ours is not a network-layer solution and we can therefore avoid the indirection for every packet transmission. We take advantage of the fact that a hostname lookup is ubiquitously done by most applications that originate communication with a network host, and use the DNS name as the invariant. We believe that this is a good architectural model: a DNS name identifies a host and does not assume anything about the network attachment point to which it may currently be attached, and the indirection occurs only when the initial lookup is done via a control message (a DNS lookup).

This implies that when the mobile host changes its attachment point, it must detect this and change the hostname-to-address ("A-record") mapping in the DNS. Fortunately, both tasks are easy to accomplish, the former by using a user-level daemon as in Mobile IP, and the latter by using the well-understood and widely available secure DNS update protocol [8, 35]. We note that some DHCP servers today issue a DNS update at client boot time when handing out a new address to a known client based on a static MAC-to-DNS table. This augurs well for the incremental deployability of our architecture, since DNS update support is widely available.

The DNS provides a mechanism by which name resolvers can cache name mappings for some period of time, specified in the time-to-live (TTL) field of the A-record. To avoid a stale mapping from being used from the name cache, we set the time-to-live (TTL) field for the A-record of the name of the mobile host to *zero*, which prevents this from being cached.[3] Contrary to what some might expect, this does not cause a significant scaling problem; name lookups for an uncached A-record do not have to start from a root name server, because in general the "NS-record" (name server record) of the mobile host's DNS name is cacheable for a long period of time (many hours by default). This causes the name lookup to start at the name server of the mobile host's domain, which scales well because of administrative delegation of the namespace and DNS server replication in any domain. We note that some content distribution networks for Web server replication of popular sites use the same approach of small-to-zero TTL values to redirect client requests to appropriate servers (e.g., Akamai [2]). There is no central hot spot because the name server records for a domain are themselves cacheable for relatively long periods of time.

Even with uncacheable DNS entries there still exists a possible race condition where a mobile host moves between when a correspondent host receives the result of its DNS query and when it initiates a TCP connection. Assuming a mobile host updates its DNS entry immediately upon reconnection, the chances of such an occurrence are quite small, but non-zero, especially for a mobile host that makes frequent moves. In this case, the correspondent host will attempt to open a TCP connection to the mobile host's old address, and has no automatic fail-over mechanism.

In this case, the application must perform another DNS lookup to find the new location of the mobile host. We note that the trend towards dynamic DNS records has caused such application-level retries to find their way into applications already—for instance, current FreeBSD `telnet` and `rsh` applications try alternate addresses if an initial connection fails to a host that has multiple DNS A-records. It seems to be only a minor addition to refresh DNS bindings if connection establishment fails.

4 TCP connection migration

A TCP connection [31] is uniquely identified by a 4-tuple: $\langle$*source address*, *source port*, *dest address*, *dest port*$\rangle$. Packets addressed to a different address, even if successfully delivered to the TCP stack on the mobile host, must not be demultiplexed to a connection established from a different address. Similarly, packets from a new address are also not associated with connections established from a previous address. This is crucial to the proper operation of servers on well-known ports.

We propose a new *Migrate* TCP option, included in SYN segments, that identifies a SYN packet as part of a previously established connection, rather than a request for a new connection. This Migrate option contains a *token* that identifies a previously established connection on the same destination $\langle$*address*, *port*$\rangle$ pair. The token is negotiated during initial connection establishment through the use of a *Migrate-Permitted* option. After a successful token negotiation, TCP connections may be uniquely identified by either their traditional $\langle$*source address*, *source port*, *dest address*, *dest port*$\rangle$ 4-tuple, or a new $\langle$*source address*, *source port*, *token*$\rangle$ triple on each host.

A mobile host may restart a previously-established TCP connection from a new address by sending a special Migrate SYN packet that contains the token identifying the previous connection. The fixed host will than re-synchronize the connection with the mobile host at the new end point. A migrated connection maintains the same control block and state (with a different end point, of course), including the sequence number space, so any necessary retransmissions can be requested in the standard fashion. This also ensures that SACK and any similar options continue to operate properly. Furthermore, any options negotiated on the initial SYN exchange remain in effect after connection migration, and need not be resent in a Migrate SYN.[4]

Since SYN segments consume a byte in the TCP sequence number space, Migrate SYNs are issued with the same sequence number as the last transmitted byte of data. This results in two bytes of data in a migrated TCP connection with the same sequence number (the new SYN and the previously-transmitted actual data), but this is not a problem since the Migrate SYN segment need never be explicitly acknowledged. Any packet received from the fixed host by a migrating host at the mobile host's new address that has a sequence number in the appropriate window for the current connection implicitly acknowledges the Migrate SYN. Similarly, any further seg-

[3] Modern versions of BIND honor this correctly.

[4] They can be, if needed. For example, it might be useful to renegotiate a new maximum segment size (MSS) reflecting the properties of the new path. We have not yet explored this in detail.

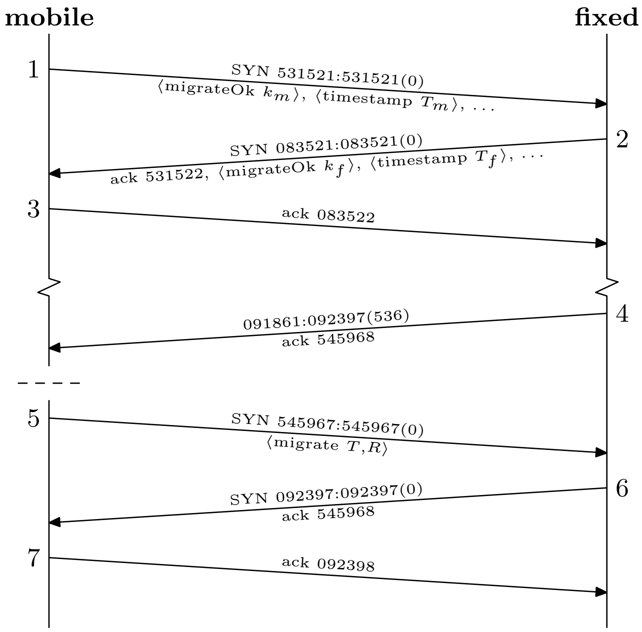

Figure 1: TCP Connection Migration

ments from the mobile host provide the fixed host an implicit acknowledgement of its SYN/ACK. Thus, there is exactly one byte in the sequence space that needs explicit acknowledgement even when the Migrate SYN is used.

4.1 An example

Figure 1 shows a sample connection where a mobile client connects to a fixed host and later moves to a new address. The mobile client initiates the TCP connection in standard fashion in message 1, including a Migrate-Permitted option in the SYN packet. The values k_m and T_m are parameters used in the token negotiation, described in Section 4.3. The fixed server, with a migrate-compliant TCP stack, indicates its acceptance of the Migrate-Permitted option by including the Migrate-Permitted option in its response (message 2). The client completes the three-way handshake with message 3, an ACK. The connection then proceeds as any other TCP connection would, until message 4, the last packet from the fixed host to the mobile host at its current address.

At some time later the mobile host moves to a new address, and notifies the fixed server by sending a SYN packet from its new address in message 5. This SYN includes the Migrate option, which contains the previously computed connection token as part of a migration request. Note that the sequence number of this Migrate SYN segment is the same as the last byte of transmitted data. The server responds in kind in message 6, also using the sequence number of its last transmitted byte of data. The ACK, however, is from the same sequence space as the previous connection. While in this example it acknowledges the same sequence number as the SYN that generated it, it could be the case that segments were lost during a period of disconnect while the mobile host moves, and that the ACK will be a duplicate ACK for the last successfully received in-sequence byte. Since it is addressed to the mobile host's new location, however, it serves as an implicit ACK of the SYN as well.

Upon receipt of this SYN/ACK, the mobile host similarly ACKs in the previous sequence space, and the connection resumes as before. All of the options negotiated on the initial SYN except the Migrate-Permitted option are still in effect, and need not be replicated in this or any subsequent migrations.

4.2 Securing the migration

It is possible to partially hijack TCP connections if an attacker can guess the sequence space being used by the connection [21]. With the Migrate options, an attacker who can guess both the sequence space and the connection token can hijack the connection completely. Furthermore, the ability to generate a Migrate SYN from anywhere greatly increases the connection's exposure. While ingress filtering can be used to prevent connection hijacking by attackers not on the path between the end hosts, such methods are ineffective in our case. We must therefore take care to secure the connection token.

The problem is relatively easy to solve if IP security (IPsec) [4] were deployed. While the spectrum of approaches that could be used is outside the scope of this paper, we note that IPsec provides sufficient mechanisms to secure migrateable connections. Currently, however, IPsec has not found wide-spread deployment. Hence, we provide a mechanism to self-secure the Migrate options. End hosts may elect to secretly negotiate an unguessable connection token, which then reduces the security of a migrateable TCP connection to that of a standard TCP connection, since no additional attacks are possible against a migrateable connection without guessing the token, and any attack against a standard TCP connection clearly remains feasible against a migrateable TCP connection.

An unguessable connection token is secured with a secret *connection key*. Since any host that obtains the connection key could fabricate the token and issue a Migrate request, we select the key with an Elliptic Curve Diffie-Hellman key exchange [36], as described below. Hosts using IPsec, or unconcerned with connection security, may choose to disable key negotiation to avoid excess computation.

4.3 Migrate-Permitted option

Hosts wishing to initiate a migrateable TCP connection send a Migrate-Permitted option in the initial SYN segment. Similar to the SACK-Permitted option [19], it should only be sent on SYN segments, and not during an established connection. Additionally, hosts wishing to cryptographically secure the connection token may conduct an Elliptic Curve Diffie-Hellman (ECDH) key exchange through the option negotiation. (Elliptic Curve Diffie-Hellman is preferred to other methods of key establishment due to its high security-to-bit-length ratio. Readers unfamiliar with Elliptic Curve cryptography can find the necessary background material in [3].)

As seen in figure 2, the Migrate-Permitted option comes in two variants—the insecure version, of length 3, and the secure version, with length 20. The secure version is used to negotiate a secret connection key, and contains an 8-bit *Curve Name* and a 136-bit *ECDH Public Key* fragment. The curve name field selects a particular set of domain parameters (the curve, underlying finite field, F, and its representation, the generating point, P, and the order of P, n), as specified in [3]. Use of the insecure version, which contains only a Curve Name field (which must be set to zero) allows the end host

Kind: 15	Length = 3/20	Curve Name	ECDH PK
ECDH Public Key (cont.)			
ECDH Public Key (cont.)			
ECDH Public Key (cont.)			
ECDH Public Key (cont.)			

Figure 2: TCP Migrate-Permitted option

	Kind: 16	Length = 19	ReqNo
Token			
Token (cont.)			
Request			
Request (cont.)			

Figure 3: TCP Migrate option

to skip the key negotiation process. In that case, the connection key is set to all zeros.

The secure variant of the Migrate-Permitted option also requires the use of the Timestamp [14] option in order to store up to 200 bits of ECDH keying material. The EDCH Public Key is encoded using the compressed conversion routine described in [3, Section 4.3.6]. The 136 least-significant bits are stored in the EDCH Public Key field of the Migrate-Permitted option, while the remaining 64 bits of the key are encoded in the Timestamp option. The timestamp option, while often included, is not used on SYN segments. The Protection Against Wrapped Sequence Numbers (PAWS) [14] check is only performed on synchronized connections, which by definition [31] includes only segments after the three-way handshake. Similarly, the Round-Trip Time Measurement (RTTM) [14] procedure only functions when a timestamp has been echoed—clearly this is never the case on an initial SYN segment. Hence the value of the Timestamp option on SYN segments is entirely irrelevant to current TCP stacks. Legacy TCP stacks will never receive a Migrate-Permitted option on a SYN/ACK, hence the Timestamp option will be processed normally. Special handling is only required for the SYN/ACK and following ACK segment on connections that have negotiated the Migrate-Permitted option, as Timestamp fields on these segments will not contain timestamps. Hence the RTTM algorithm must not be invoked for SYN/ACK or initial ACK segments of connections that have negotiated the Migrate-Permitted option.

The Timestamp *TSVal* field contains the 32 most-significant bits of the public key, while the *TSecr* field contains the next 32 most-significant bits. These two components, combined with the 136-bit EDCH Public Key field of the Migrate-Permitted option, constitute the host's public key, k. If the public key is less than 200 bits, it is left-padded with zeros. For any host, i, k_i is generated by selecting a random number, $X_i \in [1, n-1]$, where n is the order of P, and computing

$$k_i = X_i * P$$

The $*$ operation is the scalar multiplication operation over the field F. The security of the connection hinges on the secrecy of the negotiated key, hence X_i should be randomly generated and stored in the control block for each new connection. Any necessary retransmissions of the SYN or SYN/ACK must include identical values for the Migrate-Permitted and Timestamp option.

Upon receipt of an initial SYN with a Migrate-Permitted option, a host, j, with a compliant TCP stack must include a Migrate-Permitted option (and a Timestamp option if the secure variant is used) in its SYN/ACK segment. It similarly selects a random $X_j \in [1, n-1]$ which it uses to construct k_j, its public key, which it sends in the same fashion.

After the initiating host's reception of the SYN/ACK with the Migrate-Permitted and Timestamp options, both hosts can then compute a shared secret key, K, as specified in [36]:

$$K = k_i * X_j = k_j * X_i$$

This secret key is then used to compute a connection validation token. This token, T, is computed by hashing together the key and the initial sequence numbers N_i and N_j using the Secure Hash Algorithm (SHA-1) [24] in the following fashion (recall that host i initiated the connection with an active open, and host j is performing a passive open):

$$T = SHA1(N_i, N_j, K)$$

While SHA-1 produces a 160-bit hash, all but the 64 most-significant bits are discarded, resulting in a cryptographically-secure 64-bit token that is unique to the particular connection. Since SHA-1 is collision-resistant, the chance that another connection on the same $\langle address, port \rangle$ pair has an identical token is extremely unlikely. If a collision is detected, however, the connection must be aborted by sending a RST segment. (The host performing a passive open can check for collisions before issuing a SYN/ACK, and select a new random X_j until a unique token is obtained. Hence the only chance of collision occurs on the host performing the active open.)

4.4 Migrate option

The Migrate option is used to request the migration of a currently open TCP connection to a new address. It is sent in a SYN segment to a host with which a previously-established connection already exists (in the ESTABLISHED or FIN_WAIT states), over which the Migrate-Permitted option has been negotiated.

There are two 64-bit fields in a Migrate option: a *token*, and a *request*. In addition, there is an 8-bit sequence number field, *reqNo*, which must be monotonically increasing with each new migrate request issued by an end host for a connection. (The sequence number allows correspondent hosts to ensure Migrate SYNs were not reordered by the network. Sequence space wrap-around is dealt with in the standard fashion.) The token is simply the 64 most-significant bits of the connection's SHA-1 hash as computed in the Migrate-Permitted option exchange. The request, R, is similarly the 64 most-significant bits of a SHA-1 hash calculated from the sequence number of the connection initial sequence numbers N, Migrate SYN segment, S, the connection key, K, and the request sequence number, I.

$$R = SHA1(N_i, N_j, K, S, I)$$

SYN segments may now correctly arrive on a bound port not in the LISTEN state. They should be processed only if they contain the Migrate option as specified above. Otherwise, they should be treated as specified in [31]. Upon receipt of a SYN packet with the Migrate option, a TCP stack that supports migration attempts to locate the connection on the receiving port with the corresponding token. The token values for each connection were precomputed at connection establishment, reducing the search to a hash lookup.

If the token is valid, meaning an established connection on this ⟨*address*, *port*⟩ pair has the same token, and the *reqNo* is greater than any previously received migrate request, the fixed host then computes $R = SHA1(N_i, N_j, K, S, I)$ as described above, and compares it with the value of the request in the Migrate SYN. If the comparison fails, or the token was invalid, a RST is sent to the address and port issuing the Migrate SYN, and the SYN ignored. If, on the other hand, the token and request are valid, but the reqNo is smaller than a previously received request, the SYN is assumed to be out-of-order and silently discarded. If the reqNo is identical to the most recently received migrate request this SYN is assumed to be a duplicate of the most recently received SYN, and processed accordingly.

Otherwise, the destination address and port[5] associated with the matching connection should be updated to reflect the source of the Migrate SYN, and a SYN/ACK packet generated, with the ACK field set to the last received contiguous byte of data, and the connection placed in the SYN_RCVD state. Upon receipt of an ACK, the connection continues as before.

4.5 MIGRATE_WAIT state

This section assumes that the reader is familiar with the TCP state machine and transitions [33, Chapter 18].

Special processing of TCP RST messages is required with migrateable connections, as a mobile host's old IP address may be reassigned before it has issued a migrate request to the fixed host. Figure 4 shows the modified TCP state transition diagram for connections that have successfully negotiated the Migrate-Permitted option. The receipt of a RST that passes the standard sequence number checks in the ESTABLISHED state does not immediately terminate the connection, as specified in [31]. Instead, the connection is placed into a new *MIGRATE_WAIT* state. (A similar, but far less likely situation can occur if the fixed host is in the FIN_WAIT1 state—the application on the fixed host has closed the connection, but there remains data in the connection buffer to be transmitted. For simplicity, these additional state transitions are not shown in figure 4.)

Connections in the MIGRATE_WAIT state function as if they were in the ESTABLISHED state, except that they do not emit any segments (data or ACKs), and are moved to CLOSED if they remain

[5]Migrated connections will generally originate from the same port as before. However, if the mobile host is behind a NAT, it is possible the connection has been mapped to a different port.

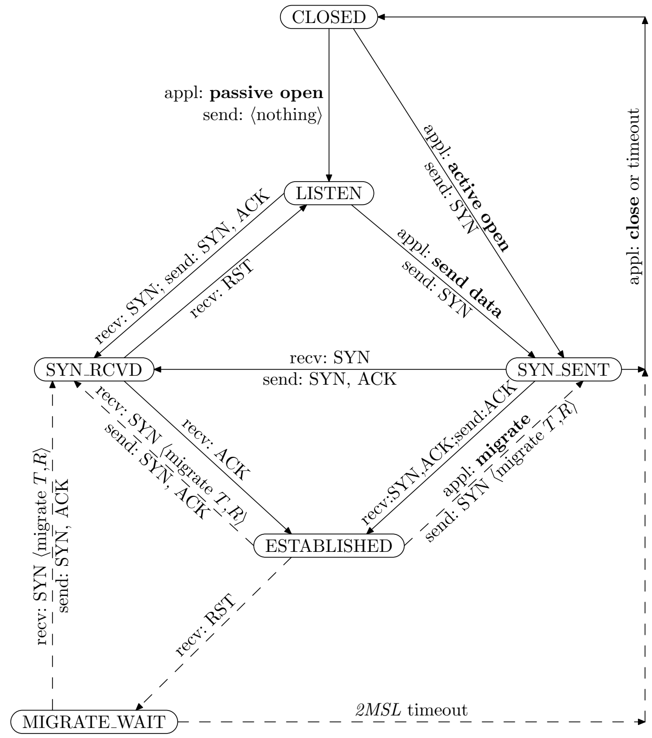

Figure 4: Partial TCP state transition diagram with Migrate transitions (adapted from [33, figure 18.12])

in MIGRATE_WAIT for over a specified period of time. We recommend using the *2MSL* ([31] specifies a Maximum Segment Lifetime (MSL) as 2 minutes, but common implementations also use values of 1 minute or 30 seconds for MSL [33]) period of time specified for the TIME_WAIT state.

Any segments received while in the MIGRATE_WAIT state should be processed as in the ESTABLISHED state, except that no ACKs should be generated. The only way a connection is removed from the MIGRATE_WAIT state is on the receipt of a Migrate SYN with the corresponding connection key. The connection then responds in the same fashion as if it were in the ESTABLISHED state when it received the SYN.

The MIGRATE_WAIT state prevents connections from being inadvertently dropped if the address allocation policy on the mobile host's previous network reassigns the mobile host's old IP address before the mobile host has reconnected at a new location and had a chance to migrate the connection. It also prevents the continued retransmission of data to an unreachable host.

This passive approach to disconnection discovery is preferred over an active, mobile-initiated squelch message because any such message could be lost.[6] Furthermore, a mobile host may not have sufficient (if any) notice of address reassignment to issue such messages. As an added performance enhancement, however, mobile hosts aware of an impending migration may themselves emit a special RST to the peer, which will force the connection into MIGRATE_WAIT, preventing additional packet transmission until the

[6]And any guaranteed-reliable transmission mechanism could take unbounded time.

mobile host has successfully relocated, although such action invokes the strict 2MSL time bound on the allowable delay for host relocation and connection migration.

5 Security issues

An end-to-end approach to mobility simplifies the trust relationships required to securely support end-host mobility compared to network-layer approaches such as Mobile IP. In addition to the relationship between a mobile host and any proxies or home agents, several Mobile IP-based proposals require that a correspondent host in communication with a mobile host assume the responsibility of authenticating communication with an arbitrary set of foreign agents. In their route optimization draft [28], Perkins and Johnson state:

> One of the most difficult aspects of Route Optimization for Mobile IP in the Internet today is that of providing authentication for all messages that affect the routing of datagrams to a mobile node.

Since no third parties are required or even authorized to speak on the mobile host's behalf in an end-to-end architecture, the only trust relationship required for secure relocation is between the mobile and correspondent host. Clearly they already must have a level of trust commensurate with the nature of their communications since they chose to communicate in the first place.

Regardless of the simplicity of trust relationships, there remains the possibility that untrusted parties could launch attacks against the end hosts or connections between them utilizing either dynamic DNS updates or the Migrate and Migrate-Permitted options. The security of dynamic DNS updates is addressed in RFC 2137 [8], resting on the strength of the digital signature scheme used to authenticate mobile hosts.

Possible attacks against the Migrate TCP options include both denial-of-service attacks and methods of migrating connections away from their appropriate end hosts. We discuss these attacks below, and either show why the Migrate options are not vulnerable, or explain why the attack presents no additional threat in relation to standard TCP.

5.1 Denial of service

SYN flooding is a common form of Denial-of-Service (DoS) attack, and most modern TCP implementations have taken great care to avoid consuming unnecessary resources unless a three-way handshake is complete. To validate a Migrate request, the correspondent host performs a significant computation (the SHA-1 hash), which implies we need to be especially vigilant against DoS attacks that attempt to deplete the CPU resources of a target host. The validation is not performed unless an attacker succeeds in guessing a valid, pre-computable token (with a 1 in 2^{64} probability); since a RST message is generated if either the token or the request is invalid, an attacker has no way to identify when it has found a valid token. Because a would-be attacker would therefore have to issue roughly 2^{63} Migrate SYNs to force a request validation, we argue that the TCP Migrate option does not introduce any additional DoS concerns above standard TCP.

5.2 Connection hijacking

Since a Migrate request contains a hash of both the SYN segment's sequence number and migrate request sequence number, a replayed Migrate option can only be used until either a new byte of data or another migrate connection is sent on the connection. Since self-migration is not allowed, duplicate Migrate SYNs (received outside of the three-way handshake) are ignored by the peer TCP. If, however, the mobile host moves rapidly to a another new location, a replayed Migrate SYN could be used to migrate the connection back to the mobile host's previous IP, which may have been subsequently assumed by the attacker. In order to prevent this attack, the Migrate Request option processing ignores the source address and port in duplicate packets, as a valid request from a relocated mobile host would include a higher request number.

More worrisome, however, is the fact that once a Migrate SYN has been transmitted, the token is known by any hosts on the new path, and denial-of-service attacks could be launched by sending bogus Migrate SYNs with valid tokens. If a mobile host includes a new Migrate-Permitted option in its Migrate SYN, however, the window of opportunity when the previous connection token can be used (if it was snooped) is quite small—only until the new three-way handshake is successfully completed.

5.3 Key security

The connection key used by the Migrate option is negotiated via Elliptic Curve Diffie-Hellman to make it extremely difficult even for hosts that can eavesdrop on the connection in both directions to guess the key. Without sufficient information to verify possible keys off-line, an attacker would have to continually generate Migrate SYNs and transmit them to one of the end hosts, hoping to receive a SYN/ACK in response to a correct guess. Clearly such an attack is of little concern in practice, as the expected 2^{63} SYN packets required to successfully guess the key would generate sufficient load as to be a DoS problem in and of themselves.

Hosts that lie on the path between end hosts, however, have sufficient information (namely the two Elliptic Curve Diffie-Hellman components) to launch an attack against the Elliptic Curve system itself. The best known attack is a distributed version of Pollard's rho-algorithm [30], which [17] uses to show that a 193-bit EC system would require $8.52 \cdot 10^{14}$ MIPS years, or about $1.89 \cdot 10^{12}$ years on a 450Mhz Pentium II, to defeat.

While this seems more than secure against ordinary attackers, an extremely well-financed attacker might be able to launch such an attack on a long-running connection in the not-too-distant future. The obvious response is to increase the key space. Unfortunately, we are restricted by the 40-byte limitation on TCP options. Given the prevalence of the MSS (4 bytes), Window Scale (3 bytes), SACK Permitted (2 bytes), and Timestamp (10 bytes) options (of which we are already using 8 bytes) in today's SYN segments, the 20-byte Migrate-Permitted option is already as large as is feasible. We argue that further securing the connection key against brute-force attacks from hosts on the path between the two end hosts is largely irrelevant, given the ability of such hosts to launch man-in-the-middle attacks against TCP with much less difficulty!

The security of TCP connections, migrateable or not, continues to remain with the authentication of end hosts, and the establishment of strong session keys to authenticate ongoing communication. Although we have taken care to ensure the Migrate option does not further decrease the security of TCP connections, the latter are inherently insecure, as IP address spoofing and sequence number guessing are not very difficult. Hence we strongly caution users concerned with connection security to use additional application-layer cryptographic techniques to authenticate end points and the payload traffic.

5.4 IPsec

When used in conjunction with IPsec [4], there are additional issues raised by the use of the Migrate options. IPsec Security Associations (SAs) are established on an IP-address basis. When a connection with an associated SA is migrated, a new SA must be established with the new destination address before communication is resumed. If the establishment of a this new SA conflicts with existing policy, the connection is dropped. This seemingly unfortunate result is actually appropriate. Since IPsec's Security Policy Database (SPD) is keyed on IP network address, the policies specified within speak to a belief about the trustworthiness of a particular portion of the network.

If a mobile host attaches to a foreign network, any security assumptions based on its normal point of attachment are invalid. If the end host itself continues to have sufficient credentials independent of its point of attachment, an end-to-end authentication method should be used, and a secure tunnel established for communication over the untrusted network. A discussion of such techniques is outside of the scope of this document.

6 Implementation

We have implemented this architecture in the Linux 2.2.15 kernel, using Bind 8.2.2-P3 as the name server for mobile hosts. The IPv4 TCP stack has been modified to support the Migrate options. Connection migration can be affected through two methods. Applications with open connections may explicitly request a migration by issuing an `ioctl()` on the connection's file descriptor specifying the address to migrate to. Most current applications, however, lack a notification method so the system can inform them the host has moved. Hence we also provide a mechanism for processes to migrate open connections, regardless of whether they have the file descriptor open or not.

This is done through the Linux `/proc` file system. A directory `/proc/net/migrate` contains files of the form *source address*:*source port*−>*dest address*:*dest port* for each open connection that has successfully negotiated the Migrate-Permitted option. These files are owned by the user associated with the process that opened the connection. Any process with appropriate permissions can then write a new IP address to these files, causing the corresponding connection to be migrated to the specified address. This method has the added benefit of being readily accessed by a user directly through the command line.

It is expected that mobile hosts will run a mobility daemon that tracks current points of network attachment, and migrates open connections based on some policy about the user's preference for certain methods of attachment. For instance, when an 802.11 interface comes up on a laptop that previously established connections on a CDPD link, it seems likely that the user would opt to migrate most open connections to the address associated with the 802.11 link. Similarly the daemon could watch for address changes on attached interfaces (possibly as a result of DHCP lease expirations and renewals) and migrate connections appropriately. We plan to implement such a daemon in the near future.

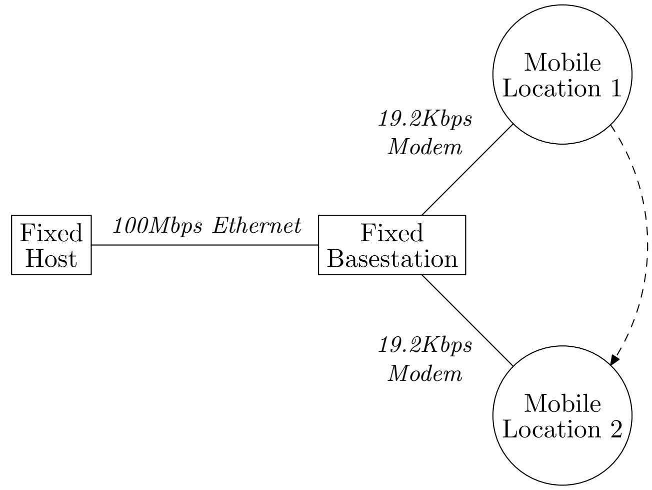

Figure 5: Network topology used for migration experiments

6.1 Experiments

Figure 5 shows the network topology used to gather the TCP traces shown in figures 6 and 7. The traces were collected at the fixed basestation, which is on the path between the fixed host and both mobile host locations. We conducted TCP bulk transfers from a server on the fixed host to a client on the mobile host. The client initiates the connection from one location, and migrates to another location at some later point. Both mobile host locations use identical connections, a 19.2Kbps serial link with ≈100ms round-trip latency. The basestation and fixed host are on a 100Mbps Ethernet segment, hence the link to the mobile host is the connection bottleneck. This topology is intentionally simple in order to isolate the various subtleties of migrating TCP connections, as discussed below.

Figure 6 shows the TCP sequence trace of a migrated TCP connection. At time $t \approx 4.9s$ the mobile host moved to a new address and issued a Migrate SYN, as depicted by the dotted line. Since the host is no longer attached at its previous address, all of the enqueued segments at the bottleneck are lost. (The amount of lost data is bounded by the advertised receive window of the mobile host. A host that moves frequently across low-bandwidth connections may wish to advertise a smaller receive window to reduce the number of wasted segments.) Finally, at $t \approx 6.8s$ the fixed host's SYN/ACK passes through the bottleneck, and is ACKed by the fixed host a RTT later.

The fixed host does not immediately restart data transmissions because the TCP Migrate options do not change the congestion-avoidance or retransmission behavior of TCP. The sender is still waiting for ACKs for the lost segments; as far as it is concerned, it has only received two (identical) ACKs—the original ACK, and one duplicate as part of the Migrate SYN three-way handshake.

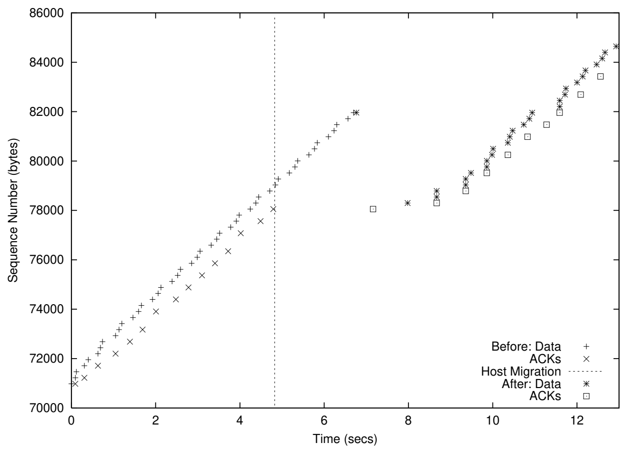

Figure 6: A TCP connection sequence trace showing the migration of an open connection

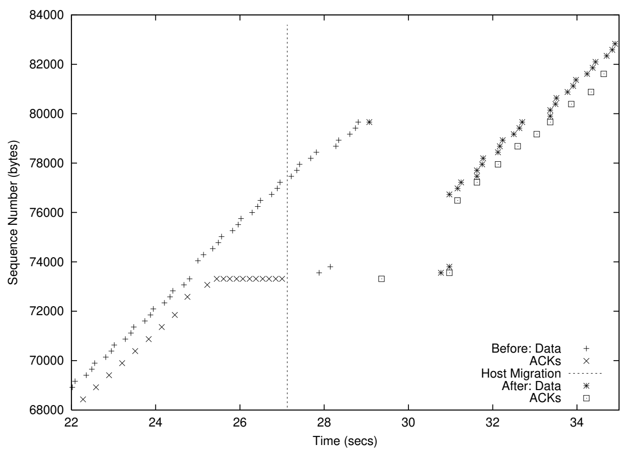

Figure 7: A TCP Migrate connection (with SACK) sequence trace with losses just before migration

Finally, at $t \approx 7.8s$ the retransmission timer expires (the interval is from the first ACK, sent earlier at $t \approx 4.9s$) and the fixed host retransmits the first of the lost segments. It is immediately acknowledged by the mobile host, and TCP resumes transmission in slow-start after the timeout.

Figure 7 shows the TCP sequence trace of a similar migrate TCP connection. As before, the dashed line indicates the mobile host issued a migrate request at time $t \approx 27.1s$. This time, however, there were additional losses on the connection that occurred just before the migration, as can be seen at $t \approx 24.9s$. These segments are fast-retransmitted, and pass through the bottleneck at $t \approx 28s$ due to the DUP-ACKs generated by the remaining SYNs. Unfortunately, this is after the mobile host has migrated, so they, along with all the segments addressed to the mobile host's initial address after $t \approx 27.1s$, are lost.

At $t \approx 29s$, the Migrate SYN/ACK makes it out of the queue at the bottleneck, and the mobile host immediately generates an ACK. As in the previous example, however, the fixed host is still awaiting ACKs for previously transmitted segments. It is only at $t \approx 31s$ that the timer expires and the missing segments are retransmitted. Notice that because SACK prevents the retransmission of the previously-received segments, only those segments lost due to the mobile host's address change are retransmitted, and the connection continues as before. The success of this trace demonstrates that the Migrate options work well with SACK due to the consistency of the sequence space across migrations.

6.2 Performance enhancements

Several enhancements can be made by implementations to improve overall connection throughput during connection migration. The most obvious of these is issuing three DUP-ACKs immediately after a migrate request, thereby triggering the fast-retransmit algorithm and avoiding the timeout seen in the previous example [6]. By preempting the timeout, the connection further avoids dropping into slow-start and congestion avoidance.

Such techniques should be used with care, however, as they assume the available bandwidth of the new path between mobile and fixed host is on the same order-of-magnitude as the previous path. For migrations across homogeneous technologies this may be a reasonable assumption. When moving from local to wide-area technologies, however, there may be order-of-magnitude discrepancies in the available bandwidth. Hence we do not include such speed-ups in the TCP Migrate specification, and leave it to particular implementations to responsibly evaluate the circumstances and provide behavior compatible with standard TCP.

7 Deployment Issues

As with any scheme for mobility support, there are some deployment issues to be addressed. By pushing the implementation of mobility mechanisms—connection migration in particular—to the end points, our system requires changes to each transport protocol. Fortunately, our TCP connection migration protocol can be generalized to other UDP-based protocols with little difficulty. Significant examples include streaming protocols such as RTP and proprietary protocols like Real, Quicktime and Netshow. We note that most of these already have a control channel used for congestion and quality control, and such applications would likely wish to be informed of changes due to mobility as well. Furthermore, we argue that not all applications *require* network-layer mobility, especially those characterized by short transactions where an application-level retry of the transaction is easy to perform; we therefore make the case using the end-to-end argument that mobility might be best implemented as a higher-level, end-to-end function just like reliability.

Perhaps the biggest limitation of our approach is that both peers cannot move *simultaneously*.[7] Because our scheme has no anchor point like Mobile IP's home agent, any IP address change must be completed before the other can proceed. We do not view this as a serious limitation to the widespread applicability of the protocol, since we are primarily targeting infrastructure-based rather than ad-hoc network topologies in this work.

In addition to these two limitations, there are several issues that crop up when one considers presently-deployed applications. While it is currently possible for Internet hosts to be re-addressed while

[7]"Simultaneously" is defined as whenever the intervals between address change and the (would-be) reception of the Migrate SYN by the corresponding host for both end hosts overlap.

operating (due to a DHCP lease expiration or similar event), it is quite rare. Hence some applications have made assumptions about the stability of network addresses, which are no longer valid in our architecture. We discuss some of these issues below.

7.1 Address caching

There is a class of applications that store IP addresses within the application, and communicate these addresses to a remote host. Such applications would not function properly under our architecture. They are readily identifiable, however, as another currently widely-deployed technology also breaks such applications: Network Address Translators (NATs). While the wisdom of Network Address Translation is a hotly debated topic, there is little chance it will disappear any time soon. Hence most applications designed today take care not to transmit addresses as part of the application-layer communication, and therefore will also work in our architecture. In fact, one can make the case that such applications are broken, since IP addresses are only identifiers of attachment points, not hosts.

Another, larger class of applications cache the results of *gethostbyname()*, and may not perform further hostname resolution.[8] Furthermore, DNS resolvers themselves cache hostname bindings as discussed in Section 3. Unfortunately many older name servers enforce a local TTL minimum, often set to five minutes. Since newer versions of popular name servers adhere to the TTL specified in the returned resource record, this problem should disappear as upgrades are made.

7.2 Proxies and NATs

Proxies actually help the deployment of our scheme, as we only need to modify the proxy itself, and all communications through the proxy will support mobility. Similarly, NATs can also provide transparent support without remote system modification. In fact, a NAT doesn't even need a modified TCP stack. It need only snoop on TCP SYNs (which it does anyway), note the presence of a Migrate-Permitted option, and snoop for the SYN/ACK (which it does anyway). If the SYN/ACK does not contain a Migrate-Permitted option, the NAT can support connection migration internal to its network by inserting a corresponding Migrate-Permitted option, and continuing to snoop the flow looking for any Migrate SYNs. It need only fabricate a corresponding SYN/ACK and update its address-to-port mappings, without passing anything to the end host. Further, by avoiding any explicit addressing in migrate requests, the Migrate options function properly though legacy NATs, and even allow a mobile host to move between NATs, as connections may change not only address but port as well.

7.3 Non-transactional UDP applications

Many UDP applications are transactional in nature. UDP is, by definition, a datagram protocol, and an inopportune change of IP address is only one of many reasons for an unsuccessful UDP transaction. The transaction will need to be retried, although a new hostname binding should be obtained first.

There exists at least one glaring exception to this rule. The Network File System protocol (NFS) represents one of the most prevalent UDP applications in use today and uses IP addresses in its mount points.[9] We believe, given the characteristics of network links likely to be encountered by mobile hosts, it is likely that NFS-over-TCP is a better choice than UDP. Otherwise, a mobile host would need to dismount and re-mount NFS filesystems upon reconnection.

[8] Some popular Web browsers display this behavior.

8 Conclusion

This paper presents an end-to-end architecture for Internet host mobility that makes no changes to the underlying IP communication substrate. It uses secure updates to the DNS upon an address change to allow Internet hosts to locate a mobile host, and a set of connection migration options to securely and efficiently negotiate a change in the IP address of a peer without breaking the end-to-end connection. We have implemented this architecture in the Linux operating system and are encouraged by the ease with which mobility can be achieved without any router support, the flexibility to mobile hosts provided by it, and performance comparable to Mobile IP with route optimization.

Our architecture allows end systems to choose a mobility mode best suited to their needs. Routing paths are efficient with no triangle routing, and any connection involving the mobile host shares fate only with the communicating peer and not with any other entity like a home agent. When a mobile host is in a foreign network and communicating with another host, the disruption in connectivity caused by a sudden IP address change is proportional to the round-trip time of the connection. When a mobile host accepts no passive connections, the protocol does not require even the DNS update notification, and seamless connectivity across host mobility is achieved using completely end-to-end machinery.

The security of our approach is based on a combination of the well-documented secure DNS update protocol in conjunction with a new secure connection migration mechanism. Our architecture and implementation function across a variety of other components of the Internet architecture, including firewalls, NATs, proxies, IPsec, and IPv6. We believe that our architecture scales well even when most Internet hosts become mobile because lookups and updates are distributed across administratively-delegated, replicated DNS servers.

We note that our connection migration scheme, the MIGRATE_WAIT state in particular, avoids address assignment race conditions, but does *not* support host disconnectivity. Hence, as with Mobile IP and other mobility schemes, TCP connections may be lost if the mobile host's relocation is accompanied by a prolonged period of disconnectivity. We are hopeful our end-to-end approach may be extended to support general host disconnectivity as well.

Acknowledgements

We thank John Ankcorn, Frans Kaashoek, Eddie Kohler, Robert Morris, Srinivasan Seshan, Tim Sheppard, and Karen Sollins for helpful comments on earlier drafts of this paper. We are indebted to David Andersen, who helped improve the security of our initial Migrate scheme, and David Mazieres, who suggested we use Elliptic Curve Diffie-Hellman key exchange for additional key strength.

[9] We note that most other advanced file systems, such as Coda [22] and newer versions of NFS use TCP, which gives good congestion control and reliability behavior.

References

[1] ADJIE-WINOTO, W., SCHWARTZ, E., BALAKRISHNAN, H., AND LILLEY, J. The design and implementation of an intentional naming system. In *Proc. ACM SOSP '99* (Dec. 1999), pp. 186–201.

[2] AKAMAI TECHNOLOGIES, INC. `http://www.akamai.com`.

[3] AMERICAN NATIONAL STANDARDS INSTITUTE. Public key cryptography for the financial service industry: The elliptic curve digital signature algorithm. ANSI X9.62 - 1998, Jan. 1999.

[4] ATKINSON, R. Security architecture for the internet protocol. RFC 1825, IETF, Aug. 1995.

[5] BALAKRISHNAN, H., SESHAN, S., AND KATZ, R. H. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks 1*, 4 (Dec. 1995), 469–481.

[6] CACERES, R., AND IFTODE, L. Improving the performance of reliable transport protocols in mobile computing environments. *IEEE JSAC 13*, 5 (June 1995).

[7] DROMS, R. Dynamic Host Configuration Protocol. RFC 2131, IETF, Mar. 1997.

[8] EASTLAKE, 3RD, D. E. Secure domain name system dynamic update. RFC 2137, IETF, Apr. 1997.

[9] FERGUSON, P., AND SENIE, D. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, IETF, Jan. 1998.

[10] GUPTA, S., AND REDDY, A. L. N. A client oriented, IP level redirection mechanism. In *Proc. IEEE Infocom '99* (Mar. 1999).

[11] HUITEMA, C. Multi-homed TCP. Internet Draft, IETF, May 1995. (expired).

[12] IEEE. Wireless medium access control (MAC) and physical layer (PHY) specifications. Standard 802.11, 1999.

[13] JACOBSON, V. Congestion avoidance and control. In *Proc. ACM SIGCOMM '88* (Aug. 1988), pp. 314–329.

[14] JACOBSON, V., BRADEN, R., AND BORMAN, D. TCP extensions for high performance. RFC 1323, IETF, May 1992.

[15] JOSEPH, A. D., TAUBER, J. A., AND KAASHOEK, M. F. Mobile computing with the rover toolkit. *IEEE Trans. on Computers 46*, 3 (Mar. 1997), 337–352.

[16] KARN, P. Qualcomm white paper on mobility and IP addressing. `http://people.qualcomm.com/karn/papers/mobility.html`, Feb. 1997.

[17] LENSTRA, A. K., AND VERHEUL, E. R. Selecting cryptographic key sizes. `http://www.cryptosavvy.com`, Nov. 1999.

[18] MALTZ, D., AND BHAGWAT, P. MSOCKS: An architecture for transport layer mobility. In *Proc. IEEE Infocom '98* (Mar. 1998).

[19] MATHIS, M., MAHDAVI, J., FLOYD, S., AND ROMANOW, A. TCP selective acknowledgment options. RFC 2018, IETF, Oct. 1996.

[20] MOCKAPETRIS, P. V., AND DUNLAP, K. Development of the domain name system. In *Proc. ACM SIGCOMM '88* (Aug. 1988), pp. 123–133.

[21] MORRIS, R. T. A weakness in the 4.2BSD UNIX TCP/IP software. Computing science technical report 117, AT&T Bell Laboratories, Murray Hill, New Jersey, Feb. 1985.

[22] MUMMERT, L. B., EBLING, M. R., AND SATYANARAYANAN, M. Exploiting weak connectivity for mobile file access. In *Proc. ACM SOSP '95* (Dec. 1995), pp. 143–155.

[23] MYSORE, J., AND BHARGHAVAN, V. A new multicasting-based architecture for internet host mobility. In *Proc. ACM/IEEE Mobicom '97* (Sept. 1997), pp. 161–172.

[24] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. The Secure Hash Algorithm (SHA-1). NIST FIPS PUB 180-1, U.S. Department of Commerce, Apr. 1995.

[25] NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. Agile application-aware adaptation for mobility. In *Proc. ACM SOSP '97* (Oct. 1997), pp. 276–287.

[26] PERKINS, C. E., AND CALHOUN, P. R. Mobile IP challenge/response extensions. Internet Draft, IETF, Feb. 2000. `draft-ietf-mobileip-challenge-09.txt` (work in progress).

[27] PERKINS, C. E., AND JOHNSON, D. B. Mobility support in IPv6. In *Proc. ACM/IEEE Mobicom '96* (Nov. 1996), pp. 27–37.

[28] PERKINS, C. E., AND JOHNSON, D. B. Route optimization in mobile IP. Internet Draft, IETF, Feb. 2000. `draft-ietf-mobileip-optim-09.txt` (work in progress).

[29] PERKINS, ED., C. E. IP mobility support. RFC 2002, IETF, Oct. 1996.

[30] POLLARD, J. Monte carlo methods for index computation mod p. *Mathematics of Computation 32* (1978), 918–924.

[31] POSTEL, ED., J. Transmission Control Protocol. RFC 793, IETF, Sept. 1981.

[32] SALTZER, J. H., REED, D. P., AND CLARK, D. D. End-to-end arguments in system design. *ACM TOCS 2*, 4 (Nov. 1984), 277–288.

[33] STEVENS, W. R. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, Reading, Massachusetts, 1994.

[34] THOMSON, S., AND NARTEN, T. IPv6 stateless address autoconfiguration. RFC 2462, IETF, Dec. 1998.

[35] VIXIE, P., THOMSON, S., REKHTER, Y., AND BOUND, J. Dynamic updates in the domain name system (DNS UPDATE). RFC 2136, IETF, Apr. 1997.

[36] ZUCCHERATO, R., AND ADAMS, C. Using elliptic curve Diffie-Hellman in the SPKM GSS-API. Internet Draft, IETF, Aug. 1999. `draft-ietf-cat-ecdh-spkm-00.txt` (work in progress).