

# DHARMA: Distributed Home Agent for Robust Mobile Access

Yun Mao, Björn Knutsson, Honghui Lu, Jonathan M. Smith  
Computer and Information Science Department  
University of Pennsylvania, Philadelphia PA 19104  
Email: {maoy,bjornk,hhl,jms}@dsl.cis.upenn.edu

## ABSTRACT

Mobile wireless devices have intermittent connectivity, sometimes intentional. This is a problem for conventional Mobile IP, beyond its well-known routing inefficiencies and deployment issues.

DHARMA selects a location-optimized instance from a distributed *set* of home agents to minimize routing overheads; set management and optimization are done using the PlanetLab overlay network. DHARMA's *session* support overcomes both transitions between home agent instances and intermittent connectivity. Cross-layer information sharing between the session layer and the overlay network are used to exploit multiple wireless links when available.

The DHARMA prototype supports intermittently connected legacy TCP applications in a variety of scenarios and is largely portable across host operating systems. Experiments with DHARMA deployed on more than 200 PlanetLab nodes demonstrate routing performance consistently better than that for best-case Mobile IP.

## I. INTRODUCTION

For mobile devices such as laptops and PDAs connected to the Internet by wireless links, intermittent connectivity can be intended (*e.g.* for saving energy during a pause) or unintended (*e.g.* as a consequence of low signal strength). In either case, these connectivity losses disrupt the stateful circuit model employed to provide TCP/IP's reliable byte stream service, as the mobility support of Mobile IP [1], [2] is aimed at maintaining constant addressability rather than constant connectivity.

The circuit or connection model is appropriate for some applications such as VoIP or continuous media while roaming. It does not reflect what has become, in our experience, the typical behavior of laptop or PDA users, who maximize the utility of their mobile devices with "as-needed" use as opposed to the "always-on" use of the office desktop. Users bring devices online to perform tasks and then suspend them, in a "transactional" style of use. This is dictated by both the style of applications in some cases (such as e-mail) and battery life in others. Unintentional disconnects can of course also result from problematic wireless conditions such as low

signal strength, physical impediment, or interference. These suspension/resumption behaviors are distinctly different from a circuit or connection model.

Mobile IP can hide addressing consequences of the movement of mobile hosts, but cannot shield the transport layer, *e.g.*, TCP, or application layer protocols from the effects of disconnection.

Four key properties are required for a robust solution to this "laptop mobility" scenario, which Mobile IP and some recent proposals [3], [4], [5], [6] lack one or more of:

- *connection semantics with intermittent connectivity*: TCP connections should be maintained during intended / unintended disconnects;
- *efficient routing*: Host-host paths should have latency close to the IP routing latency;
- *legacy support*: Legacy applications and legacy hosts should be supported;
- *application-aware adaptation*: Application requirements should be met using the best path when multiple network interfaces are available.

This paper explores this design space and trade-offs to meet these four goals. Distributed Home Agent for Robust Mobile Access (DHARMA), a new approach to achieving these properties, addresses both the issues of intermittent connectivity and routing for mobile computing systems.

## II. PROBLEM STATEMENT

### A. Mobile Computing

Laptops, tablet PCs and PDAs have assumed a different usage pattern from circuit-oriented communication, namely "as-needed" as opposed to "always-on". Along with connectivity challenges inherent in wireless communication, this has led to both user-driven and link-driven intermittent connectivity.

Applications expecting connections, or more specifically "always-on" semantics such as those inherent in a circuit, are disrupted by intermittent connectivity. Immediate examples are the interactive terminal applications such as `ssh` or `telnet` used for remote access in the Internet. Ideally, a user might initiate an `ssh` session to a server, close the laptop and bicycle to the office. After reattaching to the network, the server *should* continue the connection, but in reality if there was any traffic the connection will be closed and security associations will need to be re-established. Other applications besides

those overlaid on `ssh` suffer similar reactions to intermittent connectivity.

Experience and a timer show that 30 seconds of disconnection for Windows XP or 15 minutes for Linux is enough to break all live TCP connections<sup>1</sup>, which is relatively short in real scenarios: going through a tunnel without wireless signal, ISP or access point failure, commuting between home and office, *etc.* Other protocols that would seem to have short-lived connections may also suffer from intermittent connectivity in mobile environments as a consequence of limited bandwidth. For example, for sending large e-mail attachments in a GPRS network<sup>2</sup>, any disconnection will lead to a failure requiring the SMTP protocol to start over.

We seek a mobile computing model that is more ambitious than simply tolerating and adjusting to these inadequacies.

### B. Layering for laptop mobility

Support for mobility is distributed across layers in the Mobile IP protocol stack. Link layer solutions (*e.g.*, 802.11 WLANs, cellular networks) and network layer solutions (*e.g.*, Mobile IP) end up working in much the same way: mobility is *hidden* by relaying traffic from the mobile host<sup>3</sup> to an entry point (gateway/proxy) into the Internet that has a fixed IP address. Even with many recent advances (discussed further in Section VIII), the key assumption in the model of mobility is seamless hand-offs or very short disconnected periods. For more sporadic connectivity, we must compensate at the transport layer, or higher.

While poorly addressed in today’s Mobile IP architecture, intermittent connectivity was addressed in the comprehensive seven layer OSI model [7] in the *session* layer. A session typically consists of one or more logical connections with appropriate abstractions and state for applications to manage states end to end.

Applications on today’s Internet either assume a null session layer (*e.g.* by assuming that TCP/IP’s semantics are sufficient) or build ad-hoc session abstractions, such as the “cookies” used on the Web, into the applications and application protocols.

Lacking an explicit session layer, applications requiring such semantics must develop mechanisms for handling mobility events, such as changes of end host attachment point and periods of disconnections, unless they are *completely* masked by lower layers. An example of this latter approach is Snoeren’s session layer mobility [8], an end-to-end system-wide session abstraction designed to cope with these mobility events, and alleviate the process of mobile application development.

<sup>1</sup>Inactive connections can last longer, typically determined by the `TCP_KEEPAKIVE` settings

<sup>2</sup>Upload bandwidth limits from 9.6kbps to 28.8kbps

<sup>3</sup>**Terminology:** in this paper, we use the terms *mobile host* (MH) (also known as a *mobile node*, or MN), *home agent* (HA) and *correspondent host* (CH) as in the Mobile IP context: the MH is a mobile device that can change its “real” IP address, while the CH usually has a fixed address and acts as a server; The HA is an indirection point in the network with a fixed IP address.

### C. Legacy support and service placement

Adding a session layer affects legacy application and server software, where “legacy” denotes a lack of session-layer support. Salz *et al.* [9] have examined supporting *legacy applications*, but handling *legacy servers*, those hosts that do not and cannot support session extensions, is more difficult, as it requires deployment of new session layer mobility logic and its insertion into CHs. This is, and will likely remain, beyond the mobile user’s control due to administrator-dictated stability.

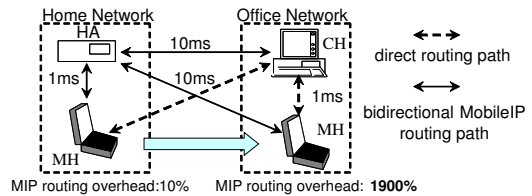


Fig. 1. Mobile IP-style home agent placement leads to inefficient routing when the mobile host moves away from the home network.

Mobility with disconnected operation for legacy servers requires creating a point of indirection or stateful gateway supporting session connectivity, analogous to Mobile IP’s *home agent* solution to MH addressing. As with any indirection, there is the risk of routing inefficiencies, such as the triangular routes resulting from transit to and from the Mobile IP home agents. The amount of added latency and wasted bandwidth is determined by the extent of the detour. Note that the inefficiency can be high even when a mobile host (MH) travels short distances, as illustrated by Figure 1, a simple but practical example where a MH travels from home to office.

## III. DISTRIBUTED HOME AGENT FOR ROBUST MOBILE ACCESS

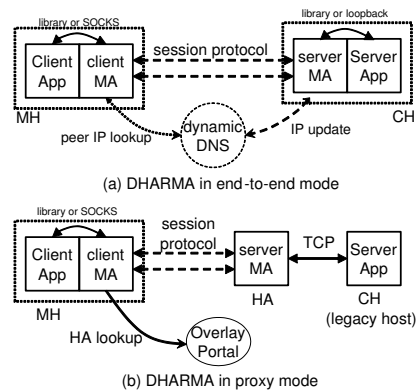


Fig. 2. The session-based mobility architecture of DHARMA. In end-to-end mode, both end points cooperate to handle mobility. In proxy mode, the legacy host is managed and shielded by a separate indirection point, the HA, selected from an overlay.

DHARMA has a session-based mobility architecture as illustrated in Figure 2, operational in two different modes: in Figure 2(a), when CH-side deployment is possible, the client

and server side of the *mobility agents* (MAs) are deployed on MH and CH respectively. MAs act as the end points of our session protocol and manage mobility events; in Figure 2(b) when CH is a legacy host, the server MA is deployed on a separate stationary machine, called the HA after its Mobile IP equivalent, serving as a mobility-aware indirection point between the MH and the CH.

The architecture has two key components: a session abstraction and dynamic home agent selection. Session abstraction handles intermittent connectivity as well as protocol augmentation, while dynamic HA selection addresses the routing efficiency problems.

#### A. Session abstraction

The session abstraction is implemented with a session protocol between MAs. The MAs can run either as part of an application, or in a local proxy. This enables arbitrary applications to use the session abstraction. The session protocol supports suspension and resumption between different transport layer level connections, as well as detection of network failures. It is the key to DHARMA’s management of intermittent connectivity; see Section VI.

The MA provides an opportunity for cross-layer interactions. That is, users can not only interact with application interfaces, but also adapt the behavior of the connection based on information from lower layers. For example, we might use a costly cellular modem connection to examine email headers, but not to support bulk downloads. DHARMA’s system session abstraction allows us to attach policy to individual applications and connections. Switching amongst interfaces and network locations are other examples of cross-layer interaction — when these events are detected, the MA issues a secure DNS update [10] so that its corresponding MA is able to locate the moved node when the session resumes.

The session layer is logically positioned between the application and the transport layer. It thus becomes a convenient place to provide additional value added services such as compression, which may improve the end-to-end throughput in low bandwidth wireless networks. Encryption services are a second example; many important legacy services run unencrypted, but may not assume the broadcast inherent in wireless links. A policy that dictates encryption for any outgoing SMTP connection over a wireless interface is straightforward.

#### B. Dynamic HA selection

When one end point does not support the DHARMA session abstraction, many of the functions of the MA are placed instead in the interposed HA.

The placement of the HA affects routing efficiency. DHARMA overcomes the routing inefficiencies inherent in a static HA assignment by dynamically selecting the HA on a per-connection basis. This can greatly improve routing performance.

DHARMA uses an overlay network to deploy distributed HAs. As shown in Figure 3, when a MA can be deployed on the CH, we get *no* routing overhead for mobility, but this

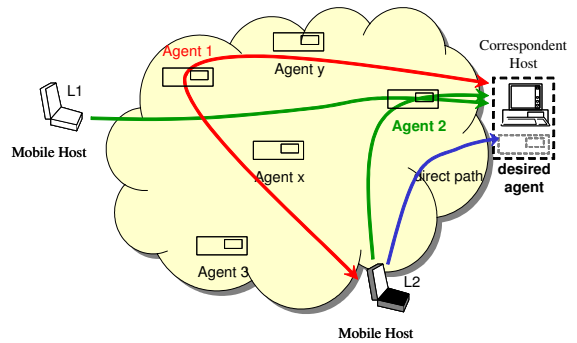


Fig. 3. The choice of location of the MA peer determines the routing efficiency. Ideally, it is co-located on the CH, but when not possible, the indirection point (HA) should be as close to the CH as possible.

is still rarely an option due to legacy issues. Instead, Agent 2 is chosen as the HA for this connection because it is the “closest” to the CH among the available agents in the overlay. As a result, when the MH moves between locations L1 and L2, although its distance to the proxy varies significantly, the path between the MH and the CH via the proxy is comparable to the direct path between the two end points. (Note the routing overhead if the agent closest to the MH’s initial position, L1, had been chosen.)

When the client application attempts to initiate a connection, the client MA intercepts it, and probes the CH for a MA instance, entering the end-to-end mode if successful. Concurrently, the client MA contacts the overlay portal service to request HA candidates close to the CH. If the probe for MA service at the CH fails, it will instead use one of the candidate HAs to establish the connection.

### IV. OVERLAY MANAGEMENT

#### A. Overlay requirements

We require three properties from an overlay supporting the DHARMA proxy mode:

- (i) Overlay node locations should be distributed corresponding to typical usage pattern of DHARMA users.
- (ii) Overlay nodes must have reasonable uptime, because a session’s lifetime is bounded by the uptime of the HA used for the session. A node failure only matters to the sessions that use it; new sessions can always use other available nodes as the HA.

(iii) The overlay should export a node lookup service which when passed an IP address returns a set of candidate overlay nodes closest to it in the IP underlay, as in [11].

The most difficult to achieve at scale is the third requirement, for measuring and predicting network distance in wide area networks. This problem is under active investigation due to its application in network distance sensitive applications such as content distribution, distributed collaboration and network games. A number of network positioning systems are proposed to predict network latency among Internet hosts [12], [13], [14], [15], [16], or to locate the closest nodes [17]. Similar systems have been built for network distance sensitive

```

home_agent_lookup(ch_ip, k, user, cred)
ch_ip: destination IP
k: requested candidates
user: username
cred: the credential of the user
Return value:
  token = [an array of HA's IPs,
           ch_ip, expiration_time]
  token signature by portal server

```

Fig. 4. The portal HA lookup service interface used by the DHARMA to locate HA candidates in the overlay

service discovery [18]. It is straightforward to provide the agent lookup service with these underlay algorithms, which we intend to do as implementations of one or more of these systems mature. In the meantime, we have implemented a simple efficient underlay service for the DHARMA prototype, as described in Section VI.

### B. Overlay portal interface

From the perspective of mobile users, any overlay network that DHARMA adopts should provide mobility support as a service to end hosts, without requiring the MA to participate directly in the overlay or understanding the inner working of the overlay network. The portal interface of a DHARMA overlay to a MA provides access to the overlay functions required by the host, such as HA lookups. This provides routing efficiency to the end host while making the routing decisions of the overlay transparent to the MA. For security, the interface should prevent mobile hosts from abusing the overlay resources.

The overlay interface to DHARMA is a server, possibly replicated, that acts as a portal to the overlay for the MAs. Part of the portal interface is shown in Figure 4. The `home_agent_lookup` RPC-API is a portal service for the overlay network. It is invoked by the client MA during connection setup in a similar fashion as a DNS lookup. The client MA requests a list of agents closest to the destination by sending the IP address of the CH to the portal server, as well as a user name and corresponding credential. The server will authenticate the MA based on the credential. If successful, it calls the underlay close node lookup service, and replies with a token consisting of a list of IP addresses of no more than  $k$  agents, the IP of the CH, together with an expiration time of the token. The server also signs the token with its private key. The listed agents can only be used to access the specified CH, which is enforced in the HA side by checking the token signature. The whole conversation between the portal server and the client MA is implemented by XML-RPC over https to protect against eavesdropping.

At the client MA, caching tokens can reduce connection setup time and the portal server load. As long as the cached token has not expired, the MA can choose any of the HA candidates listed in the return to setup a connection to the CH. Allowing the MA to choose among proxies makes the lookup more robust in case of agent failures.

One of the major purposes of having the portal service is to prevent malicious users from abusing its resources. If the MA is allowed to bypass the HA lookup service and contact any node in the overlay, a malicious user might not only consume a disproportionate amount of resources in these agents, but may also launch a DDoS attack from those agents. DHARMA polices access to the overlay by controlling the portal server's choices of which agents to return upon a client's query, and limiting  $k$  to a small integer. Because the MA cannot forge the portal server's signature of the token, it cannot use arbitrary HAs for redirecting traffic.

### C. Overlay service composition

To further extend the usefulness of DHARMA's session augmentation, and to exploit overlay routing schemes such as RON [19], DHARMA provides functions for creating per-interface *ingress agents* (IAs). The IA is looked up in the same way as the HA, except with the address of the interface as the "destination". All traffic going through the interface will now be routed through the IA, before it is forwarded to the appropriate HA. Hence, the routing path from a MH to a legacy CH becomes  $MH \rightarrow IA \rightarrow HA \rightarrow CH$ .

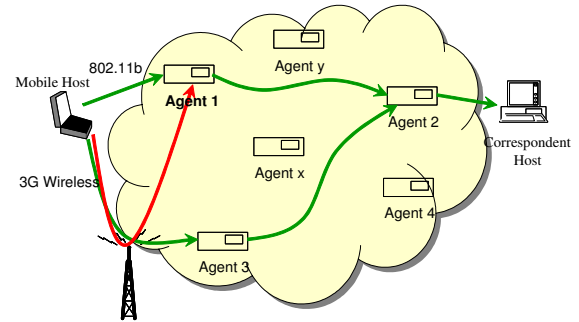


Fig. 5. A multi-homed mobile host can use different Ingress Agents (IA) for different interfaces to reduce routing overhead and to deploy interface/network specific services.

Figure 5 illustrates the two-hop communication for a multi-homed MH that involved both IAs and a HA in the overlay. Agent 1 is the IA for MH's 802.11 card while Agent 3 is the IA for the 3G wireless card. The use of an IA allows Forward Error Correction to be applied more effectively, and with less overhead and also enables other per-interface services to be more efficiently managed.

## V. DISCUSSION

*Location and authentication:* The location of the mobile host is hidden from the corresponding host in DHARMA. This differs from traditional Mobile IP, where a mobile host is associated with a unique IP home address. So while DHARMA provides IP-level anonymity, it disrupts IP-address based authentication as the the client IP may vary. In spite of flaws with this style of authentication, some services still use it for access control. An example is the ACM Digital Library, which grants access permission to universities based on IP addresses. If a HA resides in a university that happens to be

able to access the digital library, then mobile hosts that use this node will inherit it. To limit the impact, access control lists can be set up on the privileged home agents to control access to resources tied to local IP addresses through the HA.

*End-to-end TCP semantics:* Implementing a session abstraction with indirection points will break end-to-end semantics for TCP. The breaking of some TCP end-to-end semantics is not news: TCP proxies for satellite connections [20], transparent Web caches, NAT boxes and firewalls have done so for many years. The relevant issue is whether the enhancement warrants the cost. The most obvious losses are those of TCP *push* and *ACK* semantics. Examining existing applications and operating systems, we find that most do not normally export the TCP semantics that we affect to applications. A `write()` call to a TCP socket under UNIX will return not when the data has been successfully delivered, but rather when it has been queued in the local TCP. Even with strict TCP semantics, receiving an *ACK* only implies that the message has been delivered to the peer’s TCP kernel buffer, and does not guarantee delivery to the corresponding application layer.

A DHARMA session will also be affected by slow-starts. In end-to-end mode, a slow-start will happen between end points every time a session is re-initiated. In proxy mode, this is true both of the connection between client (or client MA) and HA, as well as, for longer disconnections, the connection between HA and legacy server. This will affect performance, but only when disconnections occur. It is also unavoidable, since the re-established connection will either be from a different client IP, or be likely to happen after a disconnection longer than one *RTO*, in which case TCP should re-run slow-start no matter what [21].

Finally, the repackaging of data that happens in proxy mode can have an impact on performance, especially if/when the *MSS* of the client-HA connection is different from the *MSS* of the HA-server connection. For short, infrequent, messages there may also be an impact from Nagel’s algorithm which if enabled on both the client-HA connection and the HA-server connection may run multiple times, causing higher than expected delays.

We believe the benefits of DHARMA outweigh the costs in lost strict TCP semantics, but more importantly, DHARMA offers a path by which the original semantics can be restored when deployment is ubiquitous.

*Server initiated sessions:* DHARMA currently does not support *legacy* server initiated sessions, which is hard to achieve without a static home agent. Fortunately, such cases are rare, or NAT boxes would have failed. However, mobility-aware servers can easily set up such sessions in DHARMA’s end-to-end mode by leveraging dynamic DNS, which also allows mobility of both end hosts.

*UDP support:* Legacy UDP is not currently supported by DHARMA. Because UDP is connectionless, the applications on top of it either are short-lived (*e.g.* DNS lookup), which are not bothered by mobility, or build their own connection semantics (*e.g.* RTSP) at a higher layer, which requires specific handling on a per-application basis. Hence, the best support

we could offer would only replicate that of Mobile IP, adding our improved routing.

## VI. IMPLEMENTATION

This section discusses some aspects of our prototype implementation, including the agent look up service, legacy application support, and protocols used to implement the session layer abstraction.

### A. HA lookup service

Given a specific CH, finding the best place for the HA is important for the efficiency of DHARMA. To support this, our system implements such an interface: The basic API is a single call, `HALookup()`, taking two arguments: a destination IP address and the number of desired candidate nodes  $k > 0$ . It returns up to  $k$  node descriptors for HA candidates suitable for this destination IP. The caller can use any of the candidate nodes. The main reason for having  $k > 1$  is to give the caller flexibility in case a node is down.

1) *Basic lookup service:* For our algorithm, the portal server implementing the service has a database of the  $n$  machines it manages, nodes are added/deleted from this database by an external process. We define *network distance* as the RTT reported by `ping` as our metric due to its simplicity. Using other metrics is feasible, as long as the overlay and MHs agree on the semantics of the metric.

The basic algorithm is based on random sampling:  $K > k$  nodes are selected from the  $n$  nodes in the database. These  $K$  nodes are instructed to probe the CH and report the RTT. In the simplest case, the  $k$  nodes reporting the shortest RTT are returned, but the portal service can also apply weights, based on criteria like *availability* and *load balancing* to this process.

2) *Optimizations and heuristics:* To decrease the latency for each lookup, results are cached in the lookup server. Because two nodes with IP addresses in the same /24 network are usually very close, a cached result are valid for further query within a same /24 network range. The results will be expired after a predefined timeout value  $t$ .

We apply several heuristics in addition to uniform random sampling for better selection of the  $K$  nodes. First, we take advantage of the relative stability of the network, and use expired cache results as a heuristic. It is reasonable to believe that nodes in the expired lookup are still close to the destination, and they should have a higher probability of selection in the sampling set than other nodes. The probability is inversely proportional to its prior measured distance to the destination and the elapsed time since the last measurement.

The second heuristic is based on domain names. It is applicable to overlays whose participants have topologically meaningful domain names, *e.g.*, PlanetLab where the nodes are distributed at different institutions, and are named under each corresponding administrative domain. The optimization first uses reverse DNS lookup to identify the domain name of the destination, then chooses overlay nodes based on their adjacency to the destination in the domain name space.

Distance in the domain name space is measured by simple suffix match. The probability of inclusion in the sample set increases quadratically with the length of the matched suffix. For example, if the overlay has two nodes `planetlab1.lcs.mit.edu` and `planetlab1.cs.stanford.edu`, and the destination IP is within the domain `mit.edu`, then the PlanetLab node in the MIT domain is more likely to be included in the sample set. We ignore the *generic* top level domains (TLDs), e.g., `.com` and `.net` since they lack sufficient topological information, but use most national TLDs, e.g., `.de`, `.cn`, `.uk` and `.jp`.

3) *Selection algorithm*: The above heuristics result in an empirical weighting function  $w$  defined in (1), in which  $w(n_i)$  is the unnormalized probability score that the node  $i$  will be chosen into the measurement set,  $m_i$  is the level of suffix domain the destination name matches to the node domain name,  $d_i$  is the RTT measured last time in milliseconds if node  $i$  exists in the cache but has expired, and  $x_i$  is the time it has been expired in days.

$$w(n_i) = \begin{cases} 1 + 3 \cdot 3^{m_i} \cdot e^{-x_i} & (d_i < 5) \\ 1 + 1.5 \cdot 3^{m_i} \cdot e^{-x_i} & (5 \leq d_i < 10) \\ 1 + 3^{m_i} \cdot e^{-x_i} & (10 \leq d_i < 50) \\ 1 - 0.3 \cdot 3^{m_i} \cdot e^{-x_i} & (d_i \geq 50) \end{cases} \quad (1)$$

When no useful information is available, it degenerates to random selection. By using a sufficiently large  $K$ , we can reduce the impact of occasional errors in the heuristics, e.g., when a node in the `.jp` domain is hosted in the US.

While the selection algorithm is simple, it is quite effective, as shown in Section VII. We have, however, anticipated the arrival of better lookup services [11], and designed DHARMA to allow easy replacement of the lookup service used. In fact, since the lookup is handled completely by the portal service, different lookup schemes can be deployed concurrently with no end-host modifications required. Also note that the selection algorithm assumes that CHs are fixed. When a CH moves, it is reasonable to assume that the CH is not a legacy host so that the end-to-end mode is applicable. In that case, the HA is always co-located with the CH.

### B. Legacy software support

New (versions of) applications can easily be adapted to use DHARMA to establish sessions where now they set up connections. The solution for legacy application support we choose is to use existing network proxy technology. This may cause increased overhead, but deployment can be done without changing existing applications or kernels. It also offers a simple way for the user to control where and how the new functionality is used. It is mostly OS independent, easier to deploy than kernel patches, and more practical than patching application source code.

We use the SOCKS protocol [22] and existing SOCKS support as a bridge from legacy applications to the mobile world. SOCKS protocol was designed to provide firewall traversal for TCP-based client-server applications by relaying TCP connections through SOCKS proxies. The mechanism is widely supported by various platforms, such as Windows [23]

and Linux [24]. Existing program wrappers can be used to “socksify” legacy applications by intercepting TCP connections at runtime, translating into the SOCKS protocol.

The task of deploying DHARMA is thus reduced to setting up a local SOCKS proxy on the mobile host that implements the DHARMA protocols. Because it can be set up on the loopback interface, it does not change its IP address during movement, and suffers neither packet loss nor network failures. This gives legacy applications the illusion that connections are always up, even when the actual interfaces are down.

### C. Session abstraction protocol

One of the key components of DHARMA is the session abstraction protocol, created to handle disconnections and allow deployment of session augmentations. The session protocol used is designed to work both as a full-fledged session layer protocol end-to-end, and as a proxy session protocol when one or both sides is a legacy application with no native support for sessions. The current version of the DHARMA session protocol uses a single TCP data stream for both data and detecting link failures, which simplified the design and avoids issues with firewalls.

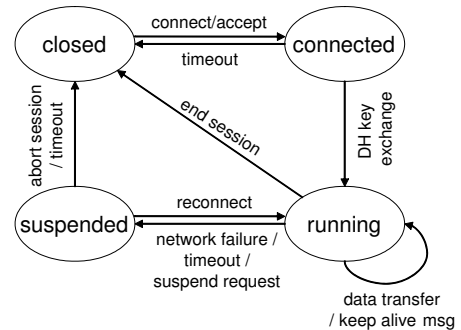


Fig. 6. State transition diagram of DHARMA session protocol

1) *Session establishment*: The session protocol has four session states, *closed*, *connected*, *running* and *suspended* as shown in Figure 6. Every session begins as *closed*. When the client application tries to connect, the client MA session manager initiates a connection to the server MA, and when established, both sides send a message identifying themselves and their capabilities, and transit to the *connected* state.

The client MA now requests that a new session be created, and perform a Diffie-Hellman key exchange [25] for later authentication. The server MA responds with a session established message containing a unique session ID. It then transits into the *running* state.

When the client MA receives the response, it also transits into the *running* state, and signals the application that the connection has been established. If it receives no response before timeout, or the server MA declines, the client MA falls back to the *closed* state.

2) *Active sessions*: When the session is active, each side runs a sender and receiver. Data is sent and received in frames, consisting of the length of the frame, the sequence number

of the highest *received* byte, *i.e.*, an ACK for received data, and the content of the frame. Sent and pending data is kept in buffers. When a frame is received, the data ACKed is discarded from the sending buffer.

If a peer has not sent any data in  $N$  seconds, or  $S$  bytes have been received since the last frame sent, a *keep-alive*/ACK frame will be sent. This is simply a zero-length frame, but sending it lets the other side know that the connection is still up, and also ACKs all received data.

3) *Session suspension*: The session transits to the *suspended* state if there is no active connection between the session peers. This can happen based on passive monitoring of the connection, *i.e.*, if the connection has been idle for  $M$  :  $M > N$  seconds, or if an error is reported from the underlying socket. It can also happen because of a suspend request, either from the peer or from the system/application, *e.g.*, to *hot switch* to another interface. Except when suspension is the result of a request from the peer, the session manager will attempt to send a *suspend* message before closing the connection.

4) *Session resumption*: A client MA may try to resume a suspended session either by an external *resume* request from the system/application, or when the reconnection timer expires.

To resume a suspended session, it does DNS lookup on its peer's host name, connects to it and sends session ID. They mutually authenticate each other through a challenge-response protocol based on the keys exchanged during session setup. If succeed, they both transit into *running* state and start retransmitting all unacknowledged data, including the data in the earlier TCP socket buffers that were not transmitted. The DNS lookup is not necessary when the peer is a HA, since a HA cannot move, and in this case a cached IP address is used directly.

5) *Session termination*: Session termination is the most difficult part of the session protocol, since without explicit notification, sessions can be kept active for a very long time before they are garbage collected. Our current implementation has only a partial termination protocol, but supports a negotiated maximum disconnect time. When a peer wants to terminate a session, it sends an explicit termination message before closing the connection and takes the session to the *closed* state. If the session is suspended, then no message will be sent.

## VII. EVALUATION

Our DHARMA prototype is implemented in Python for portability. It is currently deployed on all production nodes in the PlanetLab overlay testbed, and has been running for months. We have had our prototype in internal daily use for a number of months running on Windows XP and Linux with a multitude of applications, including remote terminals and X11 applications, instant messengers, file downloads, etc.

The prime function of DHARMA, to allow mobility with interrupted connectivity, does not lend itself to extensive

analysis. For the most part, it is a binary proposition — it either works or it does not.

We have used DHARMA extensively for short and medium term disconnection: To switch between wireless and wired networks, to survive short walks between buildings and for commuting from home to the office and back. We have also had a few occasions to try longer disconnections, while traveling across the USA, and during outages of our home networks. Except for the occasional bug in our code (confirmed and later fixed), it has worked as expected.

### A. Overlay properties

A widely distributed overlay platform like PlanetLab is ideal for deploying the DHARMA service. Table I shows the distribution of all PlanetLab nodes with DHARMA service deployed in March 2004 broken down by top level domains. Besides North America and Europe, the overlay also covers parts of Asia, Australia and South America. This geographically and topologically dispersed property is desirable for DHARMA.

TLD	.edu	.com	.org,.net	Europe	Others
Nodes	106	19	55	39	18

TABLE I  
OVERLAY NODES BROKEN DOWN BY TOP-LEVEL DOMAIN

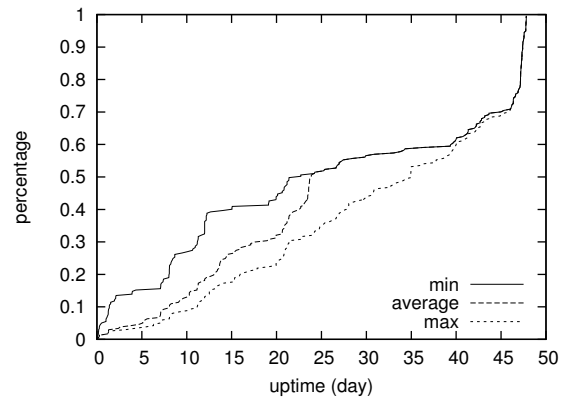


Fig. 7. CDF of minimum, average and maximum uptime of overlay nodes. Most nodes have sufficient uptime to be used as DHARMA home agents.

Next, we show that the availability of the overlay nodes is acceptable for running DHARMA. Because a session lifetime is bounded by the uptime of the HA used for the session, DHARMA requires that the overlay nodes have reasonable uptime. Figure 7 shows the cumulative distribution function (CDF) of average, minimum and maximum uptime per node in PlanetLab recording from Feb. 15, 2004 to Mar. 9, 2004. The result suggests that most of the PlanetLab overlay nodes have sufficiently long expected uptime and therefore are suitable for providing DHARMA service. For our experiments, we eliminated 15 nodes with short (less than 3 days) minimum uptime from our pool, and used the remaining 222 nodes.

While PlanetLab has been reliable enough for our experiments, we expect that a dedicated DHARMA overlay would have significantly better availability.

### B. Home agent placement

In this subsection, we use a simulator to evaluate our HA placement algorithm, the resulting routing efficiency, and compare it to a Mobile IP-like static HA approach.

45 geographically distributed servers from the top 50 engineering schools in the US<sup>4</sup> were chosen to be CH candidates in the simulation. The network distances between overlay nodes and these servers were measured, as well as the pairwise distances between overlay nodes themselves. The network distance metric used is the minimum reported RTT from 5 ICMP ECHO requests. These measurements were fed into our simulator and used for the experiments below. We believe that real network trace driven simulation makes more sense than a random topology, and the results accurately match observed behavior.

1) *Placement heuristic*: As discussed earlier, our HA placement algorithm consists of two stages. First,  $K$  candidate nodes from the  $N$  available overlay nodes are selected based on the proposed heuristics. Second, we *ping* the CH from the all the  $K$  candidates to find the  $k$  ( $< K$ ) closest nodes.

The optimal algorithm for placement would be to let  $K = N$ , i.e. *ping* the CH from all overlay nodes. This is prohibitively expensive in reality, and will become an effective DDoS attack on the CH with a sufficiently large overlay. It is, however, a good basis for comparison.

To evaluate our placement scheme, one of the 45 servers was randomly selected as the CH in each test, and the test was repeated for 4500 times. That is, on average each server was selected 100 times as the CH. The configuration was  $K = 20, k = 1$  for our algorithm. We compare the results to the optimal placement algorithm, and the random sampling algorithm without heuristics.

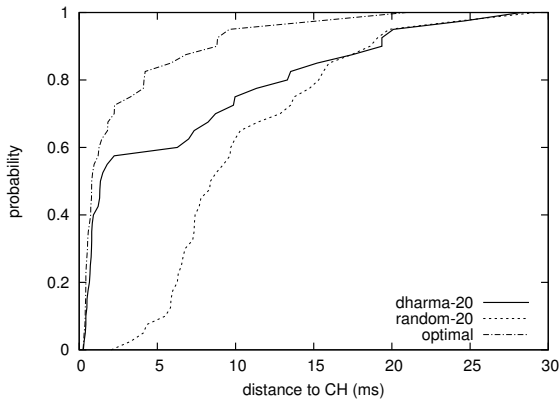


Fig. 8. CDF of distances from CHs to the estimated closest nodes. DHARMA approximates optimal placement in the majority of cases.

<sup>4</sup>The 50 server list is available at <http://www.usnews.com/usnews/edu/grad/rankings/eng/brief/engrankbrief.php>. We discarded 5 of them because they did not reply to ICMP ECHO requests used for measurements.

Figure 8 illustrates the cumulative distribution function (CDF) of the distances from CHs to the estimated closest nodes of the three algorithms, respectively. As can be seen, our algorithm is a very good approximation of the optimal in 56% of the cases. Because many universities participate in the PlanetLab overlay, the reverse DNS lookup heuristic helps locate HAs within the same campus network as the CH. For the remaining cases, our algorithm has an error of about 5 to 10 ms. In the cases where the destination domain name or IP mask do not map to any overlay nodes, our heuristic degenerates close to random selection.

2) *Routing efficiency*: To evaluate the routing efficiency of DHARMA, we compare DHARMA to the two schemes of Mobile IP, triangular routing and bidirectional tunneling (Figure 9(a) and Figure 9(b) respectively). Triangular routing in Mobile IP is often impractical, due to egress filtering of packets with non-local source addresses. It is presented for comparison and completeness. DHARMA uses bidirectional tunnels, but tries to optimize placement of the HA to reduce routing overhead.

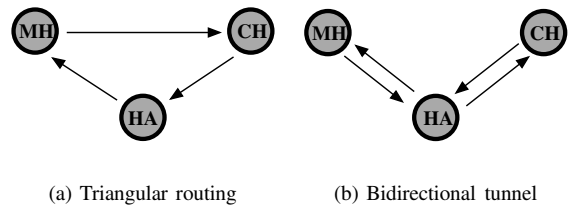


Fig. 9. Mobile IP routing options. Placement of the HA relative to MH/CH determines routing overhead in both cases, and due to concerns for spoofing attacks, triangular routing is today often not possible.

We use *routing stretch*<sup>5</sup> as the metric for our evaluation, which reflects the network routing overhead, but without the processing overhead in the HA.

For Mobile IP, we have co-located the home network (HN) and HA on a randomly selected PlanetLab node. MH movement between PlanetLab nodes is simulated using the Pareto model<sup>6</sup>. This simulates that the MH typically stays close to the HN, but occasionally moves far from the HN. The CHs are the same as in the closest node lookup test.

Figure 10 shows a comparison of 90th percentile expected routing stretches for Mobile IP and DHARMA. The sample size used by DHARMA's HA placement algorithm was varied from 5 to 222 nodes. For each CH and each sample size, we ran the DHARMA test 100 times, and show the average and standard deviation of 90th percentile routing stretches.

As expected, when DHARMA can choose from more nodes, it will be able to place its HA closer to the MH, thus reducing the expected stretch. With 20 nodes, DHARMA's routing

<sup>5</sup>The ratio of the path latency using a particular mobility scheme to the direct path latency on the underlying network topology.

<sup>6</sup>i.e. the unnormalized score when the MH is distance  $d$  from the HN is  $1/d^2$ . For special cases when  $d < 3$ ms, we arbitrarily assign the unnormalized score to  $1/9$

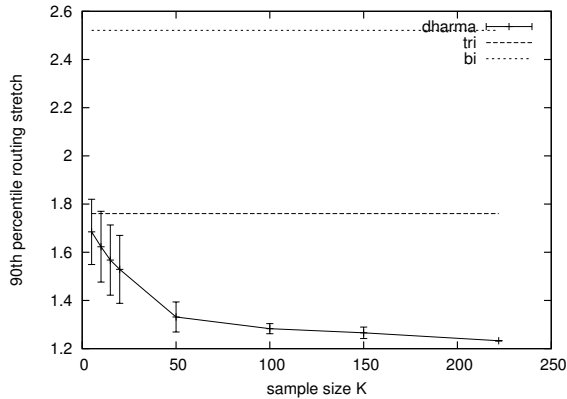


Fig. 10. 90th percentile routing stretches of MIP with triangular routing (“tri”), bidirectional tunneling (“bi”) and DHARMA. Better HA placement allows DHARMA to outperform standard Mobile IP in almost all situations.

avg. bandwidth	100BaseT	10BaseT
Standard TCP	82.1	8.82
DHARMA	74.6	8.62

TABLE II  
AVERAGE THROUGHPUT W/O DHARMA (MBPS)

overhead is 50%, while Mobile IP’s is 75% for triangular routing and 150% for bi-directional tunneling.

### C. Performance and overhead

The purpose of DHARMA is primarily to provide new functionality, but overhead and performance are still important factors in the evaluation of DHARMA. We have performed two performance experiments with our prototype, the first to estimate the overhead incurred by DHARMA compared to a direct connection, and the second to study DHARMA’s behavior in a real situation.

1) *DHARMA forwarding overhead*: In this experiment, we evaluate what the performance overhead of using DHARMA is. This means that we do not want to measure overhead from routing or network conditions. To achieve this, we deployed the CH and a HA on the same machine, and the MH on the same LAN. Then we used *iperf* to measure the end-to-end bandwidth between the CH and MH. The CH/HA runs on a 3.2GHz P4 running Linux 2.6.1 and is connected to the MH, a P3 1GHz running Linux 2.4.24, via a 10/100Mbps LAN. We then repeated the experiment without DHARMA.

We repeated the experiments 10 times, and present the average bandwidth in Table II. We see three major reasons for the differences: Overhead from duplicated TCP processing, the session protocol overhead and overhead in the socks wrapper in the MH. The session layer bandwidth overhead is 0.08%, so processing overhead is by far the most important factor. As we would expect from a processing overhead, the effect is more noticeable at higher bit rates, dropping from 9% at 100Mbps to 2% at 10Mbps. Part of this overhead can be avoided by implementing DHARMA in a language like C and eliminating

copying in the two proxies.

2) *Real usage scenario*: This experiment shows DHARMA’s behavior in a real usage scenario. The MH is running on a ThinkPad X31 with Pentium-M 1.4GHz laptop running Linux 2.4.21, with a current location on the US east coast. The CH is [www.kernel.org](http://www.kernel.org)<sup>7</sup>, and the HA is dynamically deployed on a PlanetLab node picked by our HA placement algorithm, `planetlab11.Millennium.Berkeley.EDU` in this case. The resulting setup is illustrated in Figure 11. The laptop is connected to the Internet via built-in 100Mbps Ethernet and 802.11b WLAN.



Fig. 11. The autonomous HA placement decision made by DHARMA. Notice how this allows the MH almost unrestricted movement without incurring additional routing overheads.

The experiment is to download the Linux 2.6.1 tarball using *wget* from the selected host, and switch interfaces during the download. We re-run the experiment over the WLAN without using DHARMA or switching networks for comparison.

This experiment could be handled by Mobile IP, but an end-to-end session layer could not, since the CH is a legacy node with no support for session layers.

DHARMA was configured to use a sample of 20 candidate nodes ( $K = 20$ ) to find the best ( $k = 1$ ) overlay node for this CH. The first time we ran this experiment, connection setup took about 4 seconds, the bulk of which was spent probing the RTT from the 20 candidate nodes. Once a node was picked as the best node to host the HA for this CH, this choice was cached and subsequent connection setups took 0.253 seconds, compared to the vanilla TCP connection setup time of 0.075 seconds. Since we expect sessions to be long lasting, the delay is acceptable.

Figure 12 plots the throughput of the two runs. The throughputs are almost identical until we switched interfaces after 20 seconds in the DHARMA-enabled case. At 500KB/s, we are not limited much by available host resources, and we can thus expect that for most usage scenarios, DHARMA will not cause a noticeable reduction in bandwidth.

In our current implementation, interface events are not reported to DHARMA. This means that DHARMA must use its discovery mechanism to determine that the WLAN link is down and the Ethernet is up, just as if the WLAN access point had been unplugged from the network and the router

<sup>7</sup>Which maps to `zeus-pub.kernel.org`.

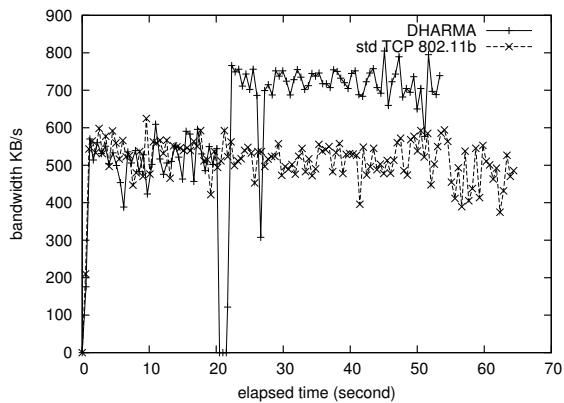


Fig. 12. Download throughput comparison. Until switching interface at  $t = 20$ , the throughputs are practically identical.

connecting the Ethernet had been plugged in. As can be seen in Figure 12, discovery and resuming the session takes about 2 seconds.

As we see in Figure 12, switching interfaces actually reduced the transmission time, an unforeseen consequence of a forced *cold switch*. DHARMA can also *hot switch* interfaces, *i.e.*, discover a new available network, and switch the session to this interface even if the original interface is still up and working correctly. In this scenario, there is a significant difference in network friendliness between Mobile IP and DHARMA. In a *cold switch*, we can expect that multiple packets will be lost, causing TCP to go into slow start. In a *hot switch*, no packet loss occurs, which for Mobile IP means that TCP will continue sending at the rate discovered for the old interface/link. DHARMA will re-probe the TCP rate when we switch interfaces.

## VIII. RELATED WORK

### A. End-to-end approaches

Snoeren *et al.* proposed an end-to-end host mobility solution Migrate [4]. Both the MH and CH use a modified TCP which can tolerate IP address changes during a connection. They use dynamic DNS to represent mobile host location information. The advantage of this end-to-end architecture is that efficient routing is guaranteed no matter where a mobile host moves. It also does not introduce a home agent as an additional point of failure. A later proposal of Migrate added session layer abstraction [26]. The reliable sockets (Rocks) proposal [5] resembles Migrate in use of end-to-end architecture, but providing a user-level library instead of kernel modification to facilitate transparent mobility support. Temporary disconnections can be masked by both Migrate and rocks by their session abstractions. DHARMA differs from them by supporting legacy servers with no modification. DHARMA is also flexible in the sense that when CH side deployment is feasible, DHARMA can run in an end-to-end fashion.

### B. Proxy based architecture

Mobile IP [1], [2] is the current IETF standard for Internet host mobility support. It employs a fixed indirection point in the network, the home agent, to forward packets between MH and CH. The original proposal of Mobile IP is subject to inefficient “triangle routing” and “bi-directional tunneling”, which occurs when an MH is far from home network, or when multiple active network interfaces are available with different IP addresses. A route optimization extension was proposed so that a MH is allowed to notify a CH directly of the MH’s current address. However, CH side modification is needed to achieve the optimization, therefore it is hardly deployable.

Recent work on Mobile IP seeks to enable dynamic placement of the HA. [27] Similar to DHARMA, the extension proposes to locate a dynamic HA when the MH moves to a new network. However, it chooses a close HA to the MH for each HA request. If the MH moves after acquiring the HA, it has to either use the old HA and suffers inefficient routing problem or break existing connections.

Despite of various extension of Mobile IP, because it handles mobility issues at network layer, it cannot address transport layer issues like TCP timing out, which makes it less useful for intermittently connected systems, such as most people’s laptops.

The MSOCKS system [3] shares the same agent forwarding idea as Mobile IP, but at the transport layer. The proxies split connections to allow TCP clients to move transparently. Because of the fixed proxy location, MSOCKS also suffers from the inefficient routing problem. More importantly, MSOCKS uses TCP splice [28] to preserve the end-to-end TCP semantics, which cannot deal with the disconnection problem.

### C. Overlay based architecture

The ROAM proposal [6] is built on top of a Internet Indirection Infrastructure (*i3*) [29]. Each trigger in *i3* can be viewed as a proxy, and each end host is able to choose the placement of its proxy according to its own location information. ROAM is fault-tolerant and efficient in terms of routing. Wrap [30] is another overlay based mobility solution. It uses structured P2P routing mesh Tapestry to form the hierarchies necessary for redirection. The focus of Wrap is to handle mobility crowd hand-off.

ROAM and Wrap are both network layer solutions, thus cannot deal with intermittent connectivity. Secondly, they are very tightly coupled with the underlying overlay network (*i3* and Tapestry respectively), while DHARMA tries to achieve efficient mobility using a general overlay with minimal requirement.

## IX. CONCLUSIONS AND FUTURE WORK

DHARMA uses an easily and incrementally deployable session layer approach to intermittent Internet connectivity for mobile computing. Current and previous versions of the software are available for download at <http://dharma.cis.upenn.edu/>.

The primary contributions are:

- 1) a unified session-based mobility architecture that works in both end-to-end and proxy modes, which addresses the dilemma between legacy host support and end-to-end routing;
- 2) an overlay network based dynamic HA selection mechanism to improve the routing performance in proxy mode;
- 3) a demonstration of the effectiveness, flexibility and ease-of-use of our prototype system in a wide area overlay network with more than 200 nodes.

Ongoing and future work based on DHARMA includes automated adaptation of the session protocol to both application requirements, as well as network and host conditions; improvements of the basic session protocol to support multi-path connections; and application specific handlers running on the indirection point for legacy applications.

#### ACKNOWLEDGMENTS

This research is supported in part by DARPA under contract F30602-99-1-0512, by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, and by DoD FA8750-04-2-0241. We would like to thank the anonymous reviewers for the helpful comments.

#### REFERENCES

- [1] John Ioannidis, Dan Duchamp, and Gerald Q. Maguire, Jr., "IP-based Protocols for Mobile Internetworking," in *Proceedings, SIGCOMM 1991*, September 4-6 1991.
- [2] Charles E. Perkins, *IP mobility support for IPv4*, RFC 3220, Internet Engineering Task Force, 2002.
- [3] David A. Maltz and Pravin Bhagwat, "MSOCKS: An architecture for transport layer mobility," in *Proceedings of INFOCOM '98*, 1998, pp. 1037-1045.
- [4] Alex C. Snoeren and Hari Balakrishnan, "An End-to-End Approach to Host Mobility," in *Proceedings of 6th ACM MobiCom Conference*, 2000.
- [5] Victor C. Zandy and Barton P. Miller, "Reliable network connections," in *Proceedings of ACM MobiCom'02 Conference*, Atlanta, GA, 2002.
- [6] Shelley Zhuang, Kevin Lai, Ion Stoica, Randy Katz, and Scott Shenker, "Host mobility using an internet indirection infrastructure," in *ACM/USENIX Mobisys*, May 2003.
- [7] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, second edition, 1988.
- [8] Alex C. Snoeren, *A Session-Based Approach to Internet Mobility*, Ph.D. thesis, Massachusetts Institute of Technology, December 2002.
- [9] Jon Salz, Alex Snoeren, and Hari Balakrishnan, "TESLA: A Transparent, Extensible Session-Layer Architecture for End-to-end Network Services," in *Proceedings of USITS03*, March 2003.
- [10] P. Vixie et al., *Dynamic updates in the domain name system*, Apr 1997, IETF RFC 2136.
- [11] Akihiro Nakao, Larry Peterson, and Andy Bavier, "A Routing Underlay for Overlay Networks," in *Proceedings of the ACM SIGCOMM Conference*, August 2003.
- [12] Hyuk Lim, Jennifer Hou, and Chong-Ho Choi, "Constructing internet coordinate system based on delay measurement," in *Proceedings of the Internet Measurement Conference*, Oct 2003.
- [13] Liying Tang and Mark Crovella, "Virtual Landmarks for the Internet," in *Proceedings of the Internet Measurement Conference*, Oct 2003.
- [14] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris, "Vivaldi: A decentralized network coordinate system," in *Proceedings of ACM SIGCOMM Conference*, Aug 2004.
- [15] T. S. Eugene Ng and Hui Zhang, "A network positioning system for the internet," in *Proceedings of USENIX ATC*, 2004.
- [16] Yun Mao and Lawrence K. Saul, "Modeling distances in large-scale networks by matrix factorization," in *Proceedings of the 2004 ACM SIGCOMM Internet Measurement Conference (IMC-04)*, Taormina, Sicily, Italy, Oct 2004, pp. 278-287.
- [17] C. Kommareddy, N. Shankar, and B. Bhattacharjee, "Finding close friends on the Internet," in *Proceedings of IEEE ICNP*, November 2001.
- [18] A. C. Huang and P. Steenkiste, "Network-sensitive service discovery," in *Proceedings of USENIX USITS*, 2003.
- [19] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris, "Resilient overlay networks," in *Proceedings of 18th ACM SOSP*, 2001.
- [20] J. Border et al., *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations*, June 2001, IETF RFC 3135.
- [21] W. Richard Stevens, *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, RFC 2001, Internet Engineering Task Force, 1997.
- [22] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, "SOCKS Protocol Version 5," RFC1928, March 1996.
- [23] "SocksCap software," <http://www.socks.permeo.com>.
- [24] "TSOCKS project," <http://tsocks.sf.net>.
- [25] A.J. Menezes, P. Van Oorschot, and S Vanstone, *Handbook of Applied Cryptography*, 1996, CRC Press.
- [26] Alex C. Snoeren, Hari Balakrishnan, and M. Frans Kaashoek, "Reconsidering IP Mobility," in *Proceedings of 8th HotOS*, May 2001.
- [27] Milind Kulkarni, Alpesh Patel, and Kent Leung, "Mobile IPv4 Dynamic Home Agent Assignment," Mobile IP Working Group Internet Draft, 2004.
- [28] David A. Maltz and Pravin Bhagwat, "TCP splicing for application layer proxy performance," IBM Technical Report RC 21139, 1998.
- [29] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana, "Internet Indirection Infrastructure," in *Proceedings of ACM SIGCOMM 2002*, 2002.
- [30] Ben Zhao, Lin Huang, Anthony D. Joseph, and John Kubiatowicz, "Rapid Mobility via Type Indirection," in *Proceedings of IPTPS'04*, Feb 2004.