

A Split Stack Approach to Mobility-Providing Performance-Enhancing Proxies

Brian D. Davison, Kiran Komaravolu, and Baoning Wu

Computer Science & Engineering

Lehigh University

Bethlehem, PA 18015

{davison, kkk2, baw4}@cse.lehigh.edu

November 2002

Abstract

Many varieties of performance-enhancing proxies (PEPs) have been proposed to improve TCP performance and/or provide seamless mobility. One simple, albeit limited technique is the application-layer proxy. It too can isolate degraded last-mile link performance from the transmission of data across the rest of the network. While Web services on proxies are common, not all network applications will fit the traditional proxy model. We suggest using the proxy in a fashion that is fully compatible with all applications. In the *split stack* approach, the application runs on the client, but client networking library calls are executed on the proxy. The split stack architecture provides three major benefits, even with limited deployment: seamless handling of last-mile disconnects; mobility without MobileIP; and improved network performance. This paper outlines the split stack architecture, relates it to previous techniques, and provides some estimates of potential performance improvement.

1 Introduction

Many varieties of performance-enhancing proxies (PEPs) have been proposed to improve TCP performance and/or provide seamless mobility [4, 7]. The typical approach requires protocol modification or stateful examination of packets at the IP or TCP layers. A simpler, albeit limited technique is the application-layer proxy. It too can isolate degraded last-mile link performance from the transmission of data across the rest of the network. While

Web services on proxies are common, not all network applications will fit the traditional proxy model.

We suggest using the proxy in a fashion that is fully compatible with all applications. In the *split stack* approach, the application runs on the client, but client networking library calls are executed on the proxy. Thus, the outside world will see the IP address of the proxy. In fact, by terminating the server’s connection at the proxy, the opportunity exists to optimize the last link between proxy and client. Therefore, it is possible that the connection between proxy and client might be handled by a protocol other than IP.

The split stack architecture provides three major benefits:

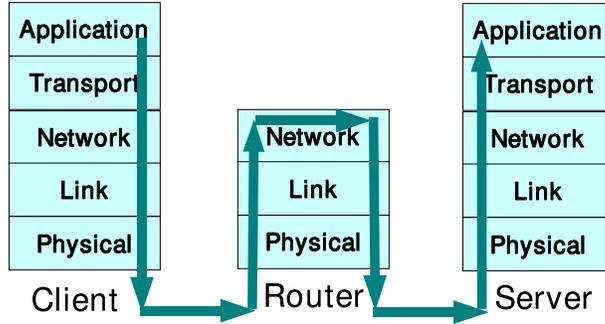
- **Seamless handling of last-mile disconnects.** Since the last-mile is no longer a part of server TCP connections, the client can re-establish a connection with the proxy and resume processing.
- **Mobility without MobileIP.** The negotiation between client and proxy does not depend on the IP address of the client — identification can be shared information when the first connection was established. As a result, the client can reconnect to the proxy from different locations and still maintain the same connections, services, and identity to the rest of the world.
- **Improved network performance.** Indirection provided by the proxy isolates the packet losses, disconnects, and latencies of last-mile links [1]. It can optionally provide significant buffering services, potentially freeing server connections earlier. Optimized protocols can improve last-mile performance further. With sufficient deployment, packets can be sent via the shortest route.

Many of these benefits are possible even with only limited deployment.

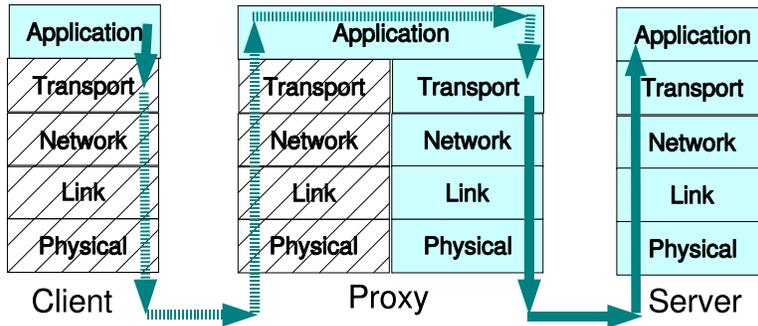
In the remainder of this paper we provide additional details of the split stack architecture, demonstrate through *ns* [5, 26] simulations some of the potential performance improvements, and relate our approach to existing work.

2 Proposed Architecture

Communications using the traditional Internet Protocol stack are shown in Figure 1(a). It shows how data is passed from the application layer down to the transport, network, link and hardware layers within a host for



(a) Traditional Internet Protocol stack communications.



(b) Split stack communications.

Figure 1: Comparison of stack usage: traditional IP stack versus split stack communication.

transmission and then interpretation back up the stack when received. In contrast to a router operating at the network layer, we insert a proxy that operates back up at the application layer.

Using the split stack architecture, the client application makes networking system calls that are not executed on the local stack. Instead, those calls are sent to a proxy for execution on its stack, as shown on Figure 1(b). The transport of the calls is shown using dashed lines since it is hidden from the client, and the stack used for communication between client and proxy is hashed to signify that non-IP protocols may be used.

To illustrate, consider the dialup client for a large ISP. In this scenario, we might replace part of the client’s standard TCP/IP stack at the OS level

(such as the WinSock 2 SPI in Microsoft Windows [19]) with calls that communicate over a proprietary, optimized protocol to an ISP server that will implement the remainder of the stack. Thus, on the client side, when an application opens a socket to a remote host, the request travels over the modem link to the ISP server where the socket is opened (creating a connection to the outside host). Likewise, when the ISP server receives a packet destined for the client, it sends the appropriate signal (and potentially the data) back over the modem to the client so that the client application will have access to it.

In general, the client-side changes could be within an application or within the OS. Actual implementation will likely vary — a system running shrink-wrapped applications will likely want to modify the OS, while more closed systems like cell-phones and PDAs might implement specialized applications with split stack functionality. In either case, the proxy contacted will still provide the same services. It establishes connections for the client application, and can even receive connections, with the caveat that only one application can run a server on a particular port (just as on any other shared system). Alternately, the proxy could be assigned many IP addresses, in which case some clients could be assigned individual IP addresses.

While potentially complex, the advantages to this type of arrangement are manifold. First, by managing a part of the virtual TCP/IP stack on the server, the client can disconnect and reconnect at will (perhaps even moving locations) via the proprietary network and connections made to the client's IP address can persist throughout. Secondly, performance to and from the virtual stack to the rest of the world will be improved, as it can handle connecting, buffering, acknowledgments, etc. at full network speed, not at modem speeds. Thus, the throughput (at least for data being received) would be throttled by buffer sizes and not by slow modem round trip times. Third, the proprietary transport between client and proxy can be optimized specifically for the particular last-mile characteristics (such as cellular modem use), rather than using the default TCP/IP timeouts and data sizes. Similarly, client-side TCP or TCP-like connections can be optimized using connection multiplexing [8] (e.g., to share information [20, 21]).

While the architecture proposed above does provide for seamless mobility of clients, performance may suffer from the need to always communicate via the home proxy, as in Figure 2(b). If, however, the remote provider has also deployed this architecture, then when a new application is started, it can bind to the new proxy. Connections held by old proxies can be retained and forwarded to the client via the new proxy. In this way, we can provide

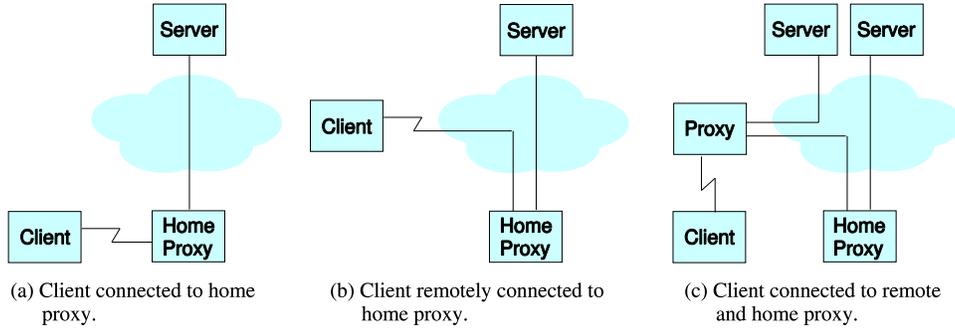


Figure 2: Various split stack usage scenarios.

optimal routing when a local proxy is available, but preserve mobility even when it is not, as shown in Figure 2(c).

One of the main concerns while designing a split proxy connection is the end-to-end semantics of the connection, e.g., data which has not yet been by the end client is acknowledged at the server to have been received. Maltz and Bhagwat [14] have proposed a mechanism for splicing two TCP connections, between the client and the proxy and the proxy and the server, to create an end-to-end flow, while maintaining the benefits of introducing a split proxy. The same technique could be applied here to maintain an end-to-end flow. The difference is that we splice a TCP connection with a transport layer connection of a proprietary protocol. Splicing connections at the proxy may limit the extent of benefit we might obtain by using a split proxy connection. But nevertheless it helps us maintain an end-to-end flow. In scenarios where an end-to-end flow is not required the network administrator may choose to turn it off.

3 Related Work

We compare and contrast the various approaches in Table 1. One of the most important features of the split stack approach is that it can hold connections between proxy and server even when the connection is broken between host and the proxy. So when the host reconnects to the proxy, it can recover the connection to the origin server immediately. This feature is common to most of the approaches concerned with mobility.

In addition, the split stack approach allows the client to reconnect from a different location (including a different ISP). This means the TCP connection between server and mobile host can be retained even when the mobile

	Split stack	I-TCP, M-TCP [2], [6]	Mob.IP in IPv4 [17, 16]	Mob.IP in IPv6 [18, 11]	VIP-1 [25]	VIP-2 [27]	Satel. PEP [10, 9]	Mowgli [12]	Snoop [3]
Connection management and dynamic addresses									
Retains TCP conns.	yes	yes	yes	yes	yes	yes	no	yes	yes
Handles new IP add.	yes	no	yes	yes	yes	yes	no	yes	no
Allows IP add. reuse	yes	N/A	no	no	yes	yes	N/A	no	N/A
Forwards non-TCP	yes	no	yes	yes	yes	yes	yes	yes	no
Network performance									
Routing solutions	AH/SP	N/A	AH/TR	SP	TR/SP	SP	N/A	AH	N/A
Improves static net.	yes	yes	no	no	no	no	yes	yes	yes
Deployment/Infrastructure Concerns									
Allows min. depl.	yes	no	no*	no	no*	no**	N/A	no*	yes
Change client/infra.	both	both	both	both	both	both	infra.	both	infra.
Layer/prot. affected	Appl.	TCP	IP	IP	IP	IP	TCP	TCP,IP	IP

Table 1: Comparison of features and approaches. Key: AH=all packets need to be passed by home agent or proxy. TR=triangle routing. SP=shortest path. N/A=not applicable. *In order to work in an unmodified foreign network, the client requires the unlikely ability to send packets with a non-local source address (i.e., the lack of ingress filters). **VIP-2 does not support mobility when communicating with non-VIP hosts.

host has moved. Since I-TCP [2] and M-TCP [6] focus on mobility within a particular provider, they do not consider changing client IP addresses. Unfortunately, while MobileIP [17, 16] supports wide mobility, allowing clients to get temporary remote IP addresses, it requires that the home IP address be static – it cannot be re-assigned to any other client. Likewise, while Mowgli [12, 13] uses essentially the split stack technique, it depends on MobileIP for some of its mobility support, so it has the same drawback. This is not the case for the split stack approach, nor the VIP techniques [25, 27]. Most approaches stack, will also forward non-TCP packets to the mobile host, even when it has moved.

When mobility using a home agent is proposed, it raises the concern for non-optimal packet routing. We have considered three scenarios: 1) all packets are passed through home agent or proxy (we call this AH); 2) all packets are passed through local router (finding the shortest path, SP); and, 3) packets from server to client are passed through the home agent while the packets from client to server are sent using the shortest route (often termed triangle routing, TR). Depending on the implementation, the split stack approach can be AH or SP — if we must always contact the home proxy, then it is AH; if a local proxy is available, then it can be SP. MobileIP is

well-known to at best use triangle routing, while most other techniques offer better solutions.

Like other PEPs for satellite [10, 9] and mobility (I-TCP, M-TCP, Mowgli), the split stack approach is intended to improve network performance. By terminating all external networking at the proxy, we can optimize internal networking to the client, and we can forward all packets (not just TCP) to the current location of the client. In addition, in some cases we can avoid or minimize significant delays associated with packet loss during connection establishment, as the unreliable last-mile connection is isolated, and may use a persistent connection to the proxy. In contrast, MobileIP for IPv4 and IPv6 [18, 11], VIP-1 and VIP-2 are concerned primarily with mobility, not network performance.

The split stack approach does not need widespread deployment to be valuable. Mobile hosts can contact their home proxy to support reconnection, mobility and local network performance improvement without having many machines or proxies changed in the Internet. In contrast, IPv6, VIP-1 and VIP-2 need widespread deployment, including both end systems and routers. Of course, as described above, with deployment of the split stack approach comes support for shorter routing for mobile hosts. Almost all approaches require support in both clients and some infrastructure, except for Satellite PEPs using TCP modifications — they only need to modify routers or proxies on either side of the satellite connection.

The Berkeley Snoop protocol [3] is a “TCP link aware protocol” aimed at improving the TCP performance of mobile computers. Snoop modifies the network layer at the base station to cache TCP packets from the foreign host to the local host. If a packet is lost between the client and the base station, the base will know this by looking at the ACK packets from the client and detects a duplicate ACK. The base station will suppress this duplicate ACK and retransmit the packet from to the server. A short local timer is used for the timeout.

The changes made are restricted to the base station alone. This is probably the most significant advantage, as it simplifies deployment of Snoop. The end-to-end semantics of the TCP connection are also preserved. The problem with Snoop is that it cannot handle IPsec encrypted data (since the Snoop module needs to look at the TCP headers). All the packets from the client have to be in an un-encrypted form. This leaves the client in a vulnerable state, making it an easy target to crack. Our approach provides for encryption of data. The packets from the client to the proxy may be encrypted, separately from whatever encryption is present between proxy and server. Of course our proxy aims at more than just improving the TCP

performance on end clients.

We proposed the idea of using a customized communication protocol architecture between the client and the proxy. Similar client-side protocol modifications have been proposed and implemented in the WAP architecture [15, 4]. The WAP architecture addresses the specific problems with low-end mobile devices like cellular phones and PDAs. The WAP client is connected to the WAP proxy through a specialized layered protocol architecture. Standard TCP/IP is used to communicate between the WAP proxy and the end server.

Different methods work at different layers. Our method focuses on the application layer. Other approaches change TCP layer protocols, such as I-TCP, M-TCP, and Satellite PEP/TCP, while others change IP layer protocols, such as MobileIP, VIP-1 and VIP-2.

In addition to the work identified in Table 1, we note that Stoica *et al.* have recently proposed a new Internet Indirection Infrastructure (*i3*) [22]. It uses a model in which receivers insert triggers for certain identifiers, and senders insert packets destined for an identifier. The infrastructure matches packets with triggers, and forwards them to the identified destination(s). Like the split stack approach, *i3* handles multicast, anycast, mobility (at either the client or server) as well as unicast. The *i3* architecture assumes the use of an efficient mechanism to map objects to servers (such as Chord [23]) so that packets and triggers are handled by the same server. Hosts need only know of one *i3* node — the nodes know how the data is mapped and forwards appropriately. This approach may have difficulty scaling to large volumes — every packet must be processed by at least one *i3* node which is not likely to be on the shortest path between client and server. Unlike our approach, it also requires significant deployment both in *i3* servers and in end-hosts for it to be useful.

Sultan *et al.* [24] propose the use of autonomous transport protocols to support a content-based network. In the endpoint name space of their network, users are identified and can move from machine to machine without losing the connection (since connections are to the users and not to the machines). This is possible because the endpoints must register themselves with the network, much like registering triggers in *i3* [22].

4 Experiments

We ran limited simulations using ns [5, 26] for four different last-mile scenarios: dialup modems, cellular modems, WiFi 802.11b laptops, and satellites.

scenario	latency	bandwidth	packet loss
dialup modem	150ms	33kbps	2%
cellular modem	1000ms	14.4kbps	25%
802.11b laptop	2ms	5.5mbit	20%
satellite	500ms	1mbit	2%
LAN	1ms	100mbit	0%
WAN	100ms	1mbit	1%

Table 2: Attributes used for the various simulation scenarios.

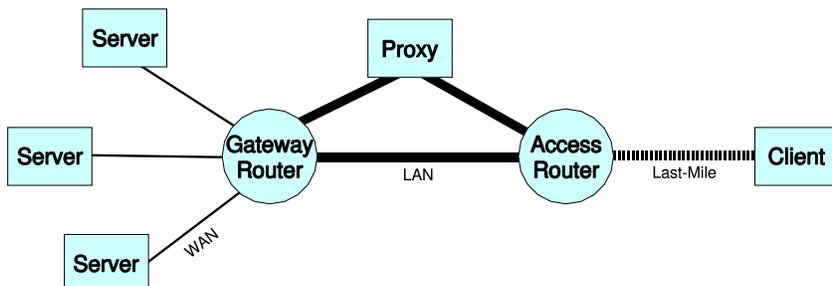


Figure 3: Network topology.

The attributes for the links in each scenario are shown in Table 2. The topology we used is shown in Figure 3. The metric we used to measure the network performance is the time taken to transfer a certain amount of bytes across the network. We measure this time from the sender’s point of view. The time of completion is marked by the final ACK received by the sender (e.g., the proxy in the split stack approach). We modeled the scenario typically as an Internet connection, with the server nodes being connected via an Internet gateway router and the client being connected to an access router. The proxy is in between the gateway and the access router. We are interested in comparing the performance of this setup is to the one without the proxy. All data transfers taking place are ftp transfers. The ftp flows are of arbitrary size and start at arbitrary times. We experimented with data flows ranging from 100Kb to 10Mb. The results are shown in Figure 4. We had improvements ranging from 8% to almost 60%. We expect the improvements to be greater in scenarios with higher loss rates and higher transfer sizes.

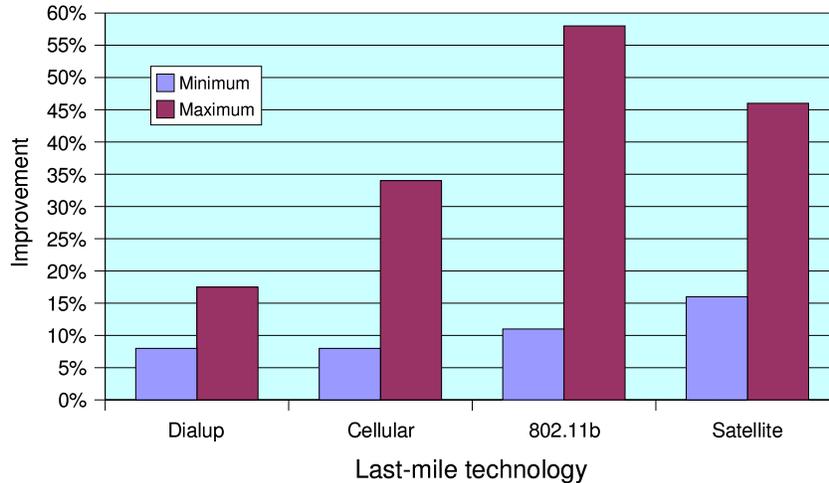


Figure 4: Simulation results: relative latency improvement for 100KB to 10MB file transfers when using the split stack architecture.

5 Summary

This paper has argued for the split stack mechanism to improve TCP performance at the last-mile and provide seamless mobility. Unlike other techniques, the split stack approach operates at the application layer, serving to replace OS networking calls on the client with remote execution on a proxy. This approach can provide benefits even with incremental deployment, and can be implemented in the OS (to run applications unmodified) or in individual applications.

While a typical proxy is limited in functionality, the split stack proxy can support both TCP and UDP and even allow the client to operate servers that handle requests. The use of a proxy with a specialized connection to the client allows the architecture to support client reconnection, even with a different network address, and still retain existing connections and services. It also permits local performance optimization of the protocol between client and server.

In addition to outlining the split stack architecture, this paper has presented the results of simulation to estimate some of the performance improvements possible under various scenarios. A prototype implementation is left as future work.

Acknowledgments

This work has been supported in part by NSF grant ANI 9903052, and by an AT&T Foundation grant to wireless networking at Lehigh.

References

- [1] B. R. Badrinath, A. V. Bakre, T. Imielinski, and R. Marantz. Handling mobile clients: A case for indirect interaction. In *Proceedings of the 4th Workshop on Workstation Operating Systems*, pages 91–97, Oct. 1993.
- [2] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *15th International Conference on Distributed Computing Systems*, 1995.
- [3] H. Balakrishnan, S. Seshan, and R. H. Katz. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks*, 1(4), Dec. 1995.
- [4] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations. RFC 3135, IETF, June 2001.
- [5] L. Breslau, D. Estron, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.
- [6] K. Brown and S. Singh. M-TCP: TCP for mobile cellular networks. *ACM Computer Communication Review*, 27(5), 1997.
- [7] H. Elaarag. Improving TCP performance over mobile networks. *ACM Computing Surveys*, 34(3):357–374, Sept. 2002.
- [8] J. Gettys and H. F. Nielsen. SMUX protocol specification. W3C Working Draft <http://www.w3.org/TR/WD-mux>, July 1998.
- [9] N. Ghani and S. Dixit. TCP/IP enhancements for satellite networks. *IEEE Communications Magazine*, 37(7):64–72, July 1999.
- [10] J. Ishac and M. Allman. The performance of TCP spoofing in satellite networks. In *Proceedings of IEEE MILCOM*, 2001.
- [11] D. B. Johnson, C. Perkins, and J. Arkko. Mobility support in IPv6. Internet-Draft draft-ietf-mobileip-ipv6-17.txt, May 2002.

- [12] M. Kojo, K. Raatikainen, and T. Alanko. Connecting mobile workstations to the Internet over a digital cellular telephone network. In *Proc. of the Mobidata Workshop*, Rutgers Univ., NJ, Nov. 1994.
- [13] M. Liljeberg, H. Helin, M. Kojo, and K. Raatikainen. Mowgli WWW software: Improved usability of WWW in mobile WAN environments. In *IEEE Global Internet*, London, UK, Nov. 1996.
- [14] D. Maltz and P. Bhagwat. TCP splicing for application layer proxy performance. *Journal of High Speed Networks*, 1999.
- [15] Open Mobile Alliance. Wireless application protocol (wap) 2.0 technical white paper. Available from <http://www.wapforum.org/>, Jan. 2002.
- [16] C. Perkins. IP encapsulation within IP. RFC 2003, IETF, Oct. 1996.
- [17] C. Perkins. IP mobility support. RFC 2002, IETF, Oct. 1996.
- [18] C. E. Perkins and D. B. Johnson. Mobility support in IPv6. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking (MobiCom'96)*, pages 27–37, Rye, NY, Nov. 1996.
- [19] B. Quinn and D. Shute. *Windows Sockets Network Programming*. Addison-Wesley, 1995.
- [20] H. S. Rahul, H. Balakrishnan, and S. Seshan. An end-system architecture for unified congestion management. In *Proceedings of 7th Workshop on Hot Topics in Operating Systems*, pages 52–57, Mar. 1999.
- [21] S. Savage, N. Cardwell, and T. Anderson. The case for informed transport protocols. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems*, pages 58–63, Rio Rico, AZ, Mar. 1999.
- [22] I. Stoica, D. Adkins, S. Ratnasamy, S. Shenker, S. Surana, and S. Zhuang. Internet indirection infrastructure. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, Mar. 2002.
- [23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Deigo, CA, Aug. 2001.
- [24] F. Sultan, A. Bohra, and L. Iftode. Autonomous transport protocols for content-based networks. Technical Report DCS-TR-479, Dept. of Computer Science, Rutgers University, Mar. 2002.

- [25] F. Teraoka, K. Uehara, H. Sunahara, and J. Murai. VIP: A protocol providing host mobility. *Communications of the ACM*, 37(8):67–75, Aug. 1994.
- [26] UCB/LBNL/VINT. Network simulator ns. <http://www.isi.edu/nsnam/ns/>, 2001.
- [27] P. Yalagandula, A. Garg, M. Dahlin, L. Alvisi, and H. Vin. Transparent mobility with minimal infrastructure. Tech. Rep. TR-01-30, Dept. of Computer Sciences, Univ. of Texas at Austin, July 2001.