Project Description

1 Introduction

The original model of the Internet is simple and homogeneous: every interface is reachable via a long-lived IP address, routers do nothing but move destination-routed packets as fast as possible over stable routes, hosts are responsible for securing themselves, and any complicated function is implemented end-to-end by applications running on hosts.

In recent years both ISPs and customers have deployed various forms of network "intermediaries" with accelerating frequency. Typically, customers deploy intermediaries primarily for security, while network operators deploy intermediaries to manage traffic better—thereby lowering their costs—and to provide services that can be used for competitive advantage. Examples of intermediaries include proxies, NATs, web caches, firewalls, web site load balancers, and protocol converters (e.g., WAP-to-IP). The result of these deployments is a more complicated *Discrete Internet*: an Internet composed of discrete subnetworks, attached at a relatively small number of choke points, lacking universal addressability, and including stateful in-network intermediaries as important service components.

Below we explain why the original "Homogeneous Internet" model is gone forever, replaced by the Discrete Internet. Given the permanent change in network architecture, this proposal develops a novel technical mechanism—a new definition of the "session layer"—as a way to manage the new complexity, retrieve some of the lost benefits of the homogeneous model, and open a variety of new possibilities.

The proposal proceeds as follows. Section 2 provides **background** and explains the assumed future environment targeted by the proposed work. Section 3 succinctly states the **problem**. Section 4 develops the **proposed solution**. Section 5 reviews the **plan for carrying out and evaluating the proposed work**. The plan includes specific **hypotheses** to test, the **experiments** to test them, and **what will be learned**. Section 6 examines prior and related work, explaining why **existing and likely-future technology is inadequate**, and therefore why the proposed work is necessary. Section 7 outlines the **PI's qualifications for successfully undertaking the work**. Section 8 restates the possible broad impact of the work. Section 9 is a simple educational plan.

2 Background

Many lament the rise of intermediaries—especially NATs, but any stateful component—and argue that the Internet can, should, or will return to its original model. As desirable as such a future may be, intermediaries are a fact of life, and are likely to become more, not less, popular for a variety of reasons that are technical, commercial, political, and operational:

1. ISPs and/or companies that control content naturally prefer the "walled garden" model wherein they receive enhanced revenue in return for enhanced services available only within their own subnetwork. A walled garden is more easily implemented by locking customers in and non-customers out at a few network choke points, rather than at each service site.

2. Although network architects revile NATs, network administrators like them. There are perceived security benefits to maintaining a private address space behind a NAT: attackers do not even know which addresses to attack or mention in their attacks against other parties. Similarly, general increased distrust and more bad actors attached to the Internet make things like firewalls, virus filters, and spam filters seem attractive. NATs vastly ease renumbering: an organization needs to renumber only its external NAT interfaces if it switches ISPs or must return addresses to IANA. Similarly, NATs simplify the simultaneous use of multiple ISPs.

3. Corporations want substantial control over the use of their intranets to limit employee actions (the Web sites they can visit, the data they can access) to prevent goofing off and avoid liability concerns. Such control is more easily implemented at a central network choke point such as a firewall.

4. Governments are likely to want to "tap" networks for reasons both laudable and not. Examples include enforcing laws governing taxation, wiretapping criminal activity, and China's censorship of Google. US Federal law requires telephone networks to be tap-able for criminal investigation; this will affect the design of IP telephony.

5. Finally, the equation for end-to-end latency of a packet is:
   $latency = propagation + transmission + processing$, where *propagation* is the time for the first bit of a packet to travel from one end to the other, and is computed as $\frac{distance}{speed-of-light}$; *transmission* is the time for the rest of the packet to propagate, computed as $\frac{size}{bandwidth}$; and *processing* is the time spent at routers, including queuing:

   $$latency = \frac{distance}{c} + \frac{size}{bandwidth} + router\ latency$$

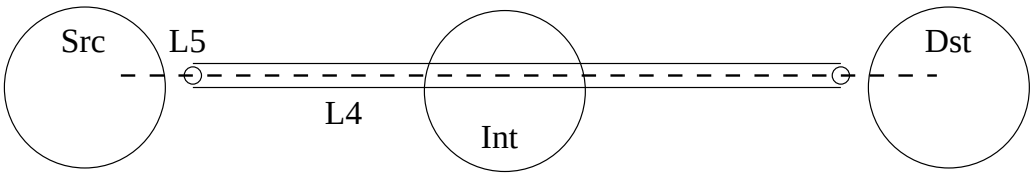
   Equipment improvements are lessening the third term, while materials improvements are making the second term smaller, meaning that as time passes the first term is becoming an increasingly large portion of the overall latency. Therefore, improvements in service latency will increasingly depend on locating the service closer to the client. This motivates techniques like web caches.

All the trends mentioned above militate toward an Internet that is more discrete: a set of separately managed subnetworks, attached at few points in order to support monitoring and control, intentionally made different in protocols and services, lacking universal addressability, and including stateful intermediaries as freestanding services or parts of end-site services. Trends such as separate ownership, owners motivated to compete with each other, corporate/governmental oversight, and service latency depending on distance, are fundamental and likely to accelerate, not decelerate. Therefore, the homogeneous model of the pre-commercial Internet is probably gone forever.

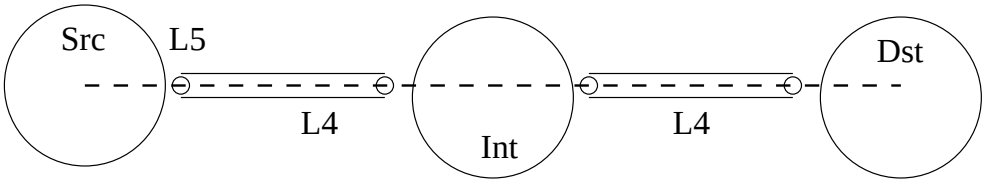
The Discrete Internet model poses some deep technical challenges. One challenge is that any intermediary that holds state on behalf of either endpoint "shares fate" with that endpoint. Making the intermediary effectively part of the application introduces another component that must be configured, secured, that can fail, and that may introduce new failure modes.

Intermediaries that are transparent constrain physical network design, route establishment, and traffic management. The network must be built, and routes established, so that it has choke points where an intermediary can see all traffic as a side effect of routing. The network design that is best for cost, performance, or customer needs might not have choke points, or might not have them in the best places for deploying network services. In such a case, the network operator must face the choice of either rebuilding its network to provide choke points (possibly to the detriment of some important characteristic such as cost, performance, etc.) or not deploying the service. A second important disadvantage of transparency is that it limits which intermediate services can be deployed. A transparent intermediary cannot query the end user or the application for extra information, nor can it inform the user or application of errors.

Also, transparency is an inelegant programming model for intermediation: intermediary code must be conscious of issues such as IP fragmentation, packet loss, and ICMP; it must inspect and possibly change payload and/or transport headers, sometimes even falsifying basic information such as the source IP address. Such a method of operation, besides being inelegant, makes intermediate services harder to write, thereby constraining the services that developers build or even contemplate.



Current method: a single end−to−end transport (L4) connection intercepted transparently by the intermediary. The intermediary spies on the connection to provide its services, necessarily handling ICMP messages, IP fragmentation, and knowing how to parse L3 and L4 headers.



Proposed method: a single end−to−end session (L5) connection running over two transport connections in series. The intermediary reads complete application records from one socket and writes complete application records to the other socket.

Figure 1: Overview of Session Architecture

3 The Problem

This proposal presupposes that—for the reasons discussed above—the Internet will become ever more discrete. If true, then it will be profitable to figure out how better to manage the discreteness. Therefore, the problem addressed by this proposal is: *how to make it easier to write, deploy, and manage in-network intermediate services.*

4 Proposed Solution: Session Layer

We propose a new Internet architecture that includes a "session" protocol at layer 5. In this architecture, the possibility of intermediaries operating along a network path between two endpoints is explicitly recognized and provided for. The definition of the session layer is our own, and does not correspond closely to the OSI definition [20].

As shown in Figure 1, this new end-to-end protocol runs on top of one or more transport connections, each of which is connected in series between end systems and/or intermediaries. For instance, if *Src* and *Dst* were the endpoints, and *Int* were an intermediary, then two transport (L4) connections, *Src-Int* and *Int-Dst*, would be used in this mode of operation. The session protocol (L5) would ensure delivery between *Src* and *Dst*, and the intermediate node *Int* would benefit from a cleaner programming model: two sockets terminating transport connections, one in each direction. The two transport connections may even use different transport protocols if that is the right thing to do.

There can be any number of intermediate servers in a session, including zero. A session employing no intermediate service would operate on top of a single transport connection, much like usual socket-to-socket communication, except with a session header as part of the transport payload. More typically, one might expect a session running over, say, a 15-hop path to use a handful of intermediate services. Hops not involved in providing service are unmodified routers forwarding

IP packets as usual. The sites that implement intermediate services might be modified routers or dedicated hosts; the session protocol places no constraints on this choice.

The session layer preserves record boundaries and verifies end-to-end delivery, offering three different semantics: reliable in-order delivery, unreliable delivery, and reliable (but not necessarily ordered) delivery.[1]

4.1 API

We have implemented a preliminary version of the session protocol and created a new API that allows an application to create, use, and destroy a session. With this API, a session initiator (client) specifies which intermediate services it desires, permits, or prohibits to exist along the network path to the other endpoint. The client may also specify the order in which services appear. The service site can inspect and alter the client's list of services. No session is established without approval of both endpoints.

The API provides for clients, intermediaries, and servers. The server API is quite similar to the standard socket interface for servers; it allows servers to declare there existence, await incoming connection requests, and read and write data using methods very similar to common `read()` and `write()`.

The code below is an example client application that communicates with a chat server over L5. (The example is very slightly altered for clarity of presentation.) The client inserts a single intermediate service, a "filter server." The filter server drops any chat messages that contain a prohibited word; in this case, the word is "proscribed." This fanciful example is a strawman meant to suggest virus filtering, spam filter, or web site blocking.

```
char *badWord = "proscribed";

//  Create layer 5 connection record.
conn = new L5_Connection();
if (conn->Error() != ERRNO_NONE)
    crash("Creating connection: %s", conn->ErrorInfo.c_str();

//  Create record describing "word filter" intermediate service.
IntService filter = L5ConfigManager::getConfigManager()->GetServiceCode("INTSERV_FILTER");

//  Request word filter service in L5 connection.  Its location
//  (IP address, port) is unspecified, so a service instance will
//  be discovered dynamically.  TCP is the transport protocol.
conn->RequestService(filter, INTSERV_IP_UNSPECIFIED, 0, 0, INTSERV_TCP);

//  Tell word filter service to delete messages containing the bad word.
conn->SetServiceParameter(filter, badWord, strlen(badWord));

//  Connect to "chat server" via intermediate filter service.
char *serverName = "CHAT_SERVER";
int deliverySemantics = RELIABLE;
if (conn->Connect(serverName, deliverySemantics) != RES_OK)
    crash("Connecting to chat server: %s", conn->ErrorInfo.c_str();

//  Write into L5 connection
char *buffer = "Message sent by this client to chat server.";
```

[1] Reliable unordered delivery is useful for applications that perform their own sequencing.

```
int buflen = strlen(buffer);
int returnCode = conn->l5_write(buffer, buflen);
```

The code below shows the filter server. The key point of this example is that the intermediate service is easy to program. It reads an incoming message with `l5_service_read()`, peeks at the data, and either forwards the message (if it doesn't contain the word "proscribed") or sends an error notice back to the sender (if the message does contain the word). Either action is accomplished by calling `l5_service_write()` with proper argument. Since TCP is the transport protocol used to communicate with the filter, when it calls `l5_service_read()` the service sees the complete application payload rather than individual packets. Consequently, the service logic need not be concerned with loss, fragmentation, ICMP, or L3/L4 header formats. The endpoints could even share encryption keys with the intermediary, allowing it to inspect encrypted content.

```
//  Register intermediate filtering service by name "INTSERV_FILTER"
conn = new L5_Socket();
if (conn->l5_register_service(
                L5ConfigManager::getConfigManager()->GetServiceCode("INTSERV_FILTER"),
                argv[1]) != RES_OK)
    conn->PrintError();

//  Read incoming message
L5ServiceMsg *msg = conn->l5_service_read();
if (msg == NULL)
    crash("Connection error: read");

//  Create response record
L5ServiceResponse *resp = new L5ServiceResponse();

//  If bad word found in message: return error to sender
//  Otherwise: forward message to destination
resp->data = msg->data;
if (strstr(msg->data->data.getBuffer(), msg->parameter)) {
    resp->action = INTSERVICE_RESPONSE_RETURN;
    sprintf(resp->data->data.getBuffer(), "Bad word (%s) detected", msg->parameter);
    resp->data->data_size = strlen(resp->data->data.getBuffer()) + 1;
} else {
    resp->action = INTSERVICE_RESPONSE_FORWARD;
}
conn->l5_service_write(resp);
```

4.2 Potential Advantages

Adding an extra level of indirection between transport and application layers threatens to increase complexity and overhead. To make the concept worthwhile there should be significant potential advantages. This section briefly mentions some. There are disadvantages and challenges too; some of these are discussed starting in Section 4.3.

Potential advantage: A **cleaner programming model** for both applications and intermediate services:

Current intermediaries must either be user-configured with a host name or transport address (e.g., Web proxies), or implemented transparently. Using the session model, the application can

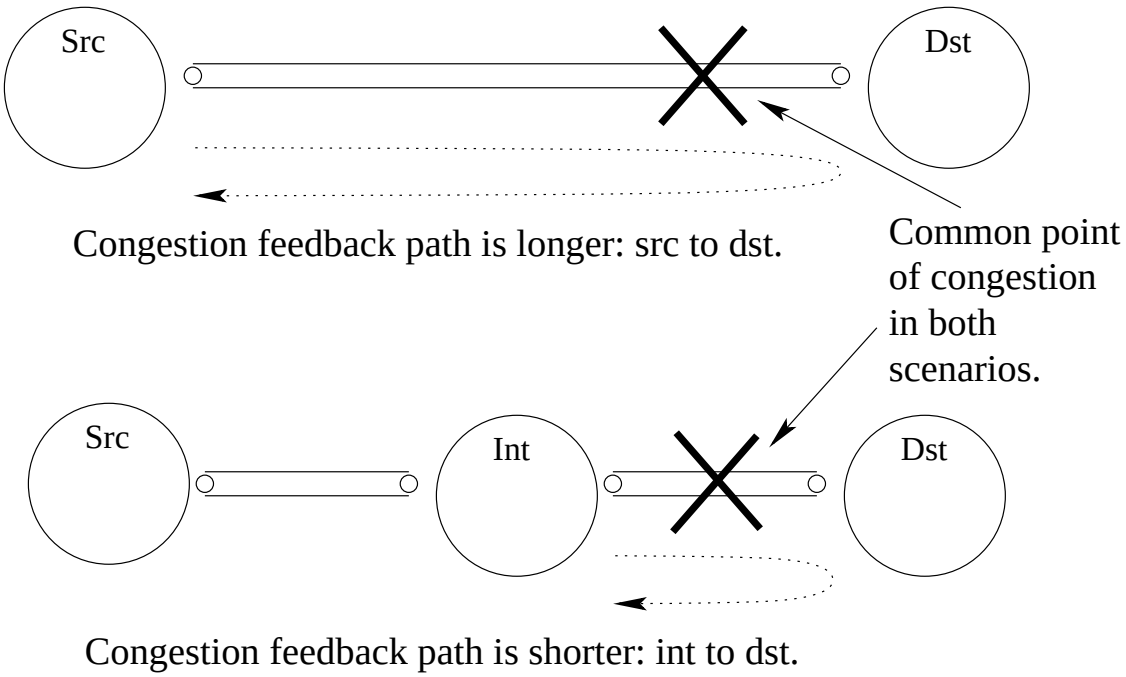


Figure 2: Shorter connections mean more rapid response to congestion.

specify an intermediate service generally, using a name. A service site along the communication path will be chosen for the application if the application prefers not to specify a location.

Currently, intermediaries must surmise when an "interaction" begins and ends. TCP makes this easy with its `SYN` and `FIN` bits, but this is a real problem with UDP-based services, enough so that some "middleboxes" (NATs and firewalls) do not allow UDP traffic to pass [27, sec 2.1]. The session architecture **clearly delineates the start and end of each interaction** through explicit setup and teardown signaling, regardless which transport protocols are used.

Potential advantage: **Improved congestion control**, both for individual transport connections and the Internet as a whole.

Implementing a session with several physically short transport connections instead of a single long one means each transport connection should have, on average, both smaller RTTs ($SRTT$) and less RTT variation ($RTTVAR$), and therefore timeouts should happen faster. Faster timeout provides both **faster recovery and faster congestion control**. Shorter connections should provide faster congestion control even when congestion is signaled explicitly (e.g., ECN [36]) rather than implicitly.

By performing congestion control over shorter paths that contain fewer routers, the response to congestion is more localized to the specific problem area. This should result in **better overall Internet capacity utilization**, since areas away from the bottleneck will not have their sending rates reduced unnecessarily. Also, there might be **less congestion in the overall Internet** if there were more points of congestion control, each one being more rapidly reactive.

We have conducted ns-2 simulations comparing congestion scenarios depicted in Figure 2. Provided that sufficient buffer capacity is available at the intermediary, response to congestion is up to 60% faster because of declines in both the round trip average and variance.

Potential advantage: **Improved connection throughput**.

There is a possibility of **improved throughput via pipeline parallelism**: if data flows from X to Y through Z, then two sender-receiver pairs (X-Z and Z-Y) operate over separate paths, simultaneously filling two receive buffers. With proper timing, it is possible that the two transport

connections to work in parallel for long enough to deliver data to Y faster than would a single transport connection between X and Y. Indeed, such a result has been reported for TCP, discovered by accident [37].

We have already investigated this possibility both through analysis [51] and ns-2 simulations and shown that, for reasonable loss rates and round trip times (RTTs), throughput can be up to 50% higher.

The multiplexing of many sessions over a fewer transport connections provides exactly the circumstances needed to realize pipeline parallelism at the transport level: a connection that is both long-lived and less bursty because of the statistical effects of multiplexing many L5 sessions over a single L4 connection. With transport connections no longer operating end-to-end, they can be left running permanently. At the limit, each pair of communicating nodes might send all traffic between themselves over a single long-lived transport connection.[2] There are well documented benefits to big flows [3]. Initialization overheads—such as Path MTU discovery, encryption key generation, etc.—can be more fully amortized. Also, the transport layer can use long-term aggregated information for congestion control (as suggested for Ensemble-TCP [21] and the Congestion Manager [4, 5]) rather than re-learning it for each (possibly brief) new transport connection.

Potential advantage: A **larger average MTU can be expected**—on average, over all transport connections—because a bottleneck link will limit only a single one of what might be several consecutive transport connections. For example, IP-over-ATM AAL5 sets the MTU at 9180 bytes [29], far in excess of the 1500-byte MTU common at the network edge. Therefore, ATM-based backbones are sending much smaller frames than they are equipped to. If transport connections carried the multiplexed traffic of many sessions, they would be better able to fill up large frames. This is significant many performance costs inside routers relate to packets handled rather than bytes transferred [29].

We are conducting ns-2 simulations of the situations similar to that pictured in Figure 3. The key issue in agglomerating smaller packets at intermediary *Int1* is scheduling: when *Int1* receives a packet on one transport connection, how long should it wait for a second packet (and further packets) to arrive on its other connections before forwarding to *Int2*?

Potential advantage: A 5-layer architecture can aid in **development and deployment of new ideas** at lower layers:

It has often been noted (e.g., [26]) that an architectural failing of IPv4 is that an IP address must serve as both the "endpoint identifier" (identifying a host) and as the "locator" (for routing). The overloading of IPv4 addresses complicates mobility models, encourages unsound security practices such as "authentication" based on source IP, and may have prevented good ideas from even being contemplated. Session layer connection IDs would act as endpoint identifiers, helping to **clean up TCP/IP** by allowing IPv4 addresses to be used purely for routing.

Because a session is named by an L5 identifier, it survive a change to the underlying transport parameters, including, for example, an endpoint changing its IP address, perhaps because it is mobile. In fact, we have implemented a prototype that maintains active sessions across an IP address change. In such a case, the last transport (TCP) connection—the one connecting an intermediary and an endpoint—is torn down and replaced. With the new TCP connection in place, the session resumes with no lost data. The application, which uses the L5 API, is unaware of the change.

Many papers, on disparate topics—examples include resource reservation [12] and various application-level session mechanisms such as HTTP cookies [28]—have introduced the concept of a "session" in order to provide a more intuitive discussion. The proposed work makes the session

[2]This is admittedly only an extreme example; however, it is not so farfetched. A recent paper that provides highly accurate measurements of Internet topology reports that 90% of all routers are connected to 5 or fewer other routers, while 99% are connected to 13 or fewer [46, sec. 7.2].

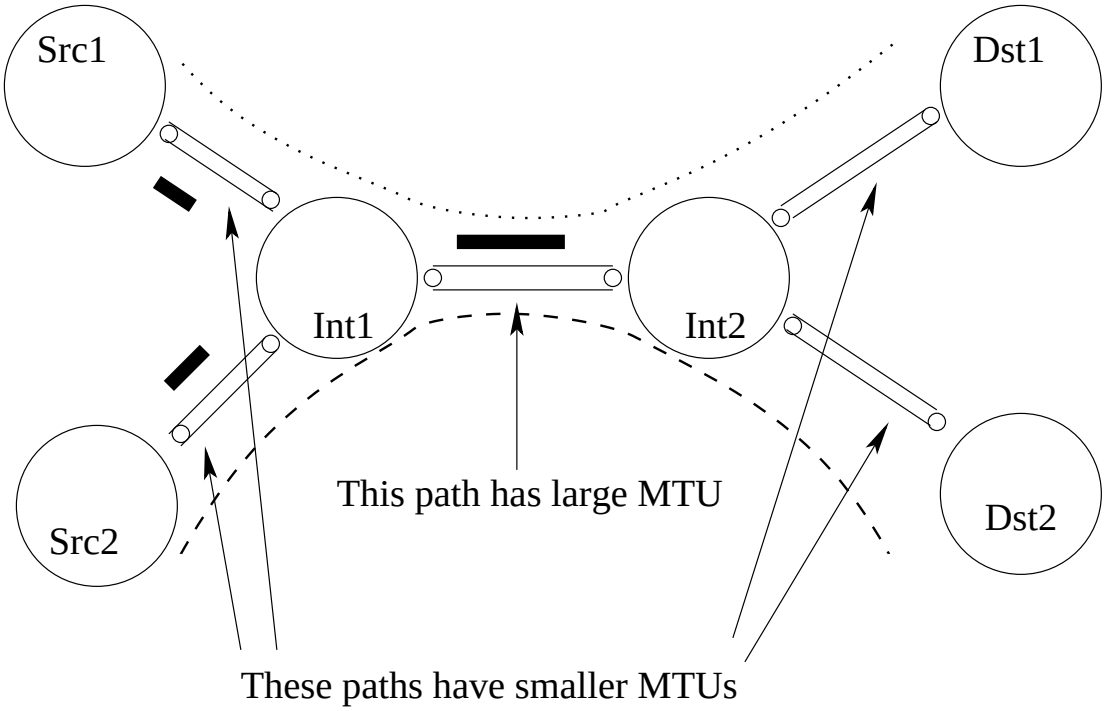


Figure 3: Small packets (in black) can be agglomerated at intermediaries and forwarded as fewer, larger packets in the Internet core, where the MTUs tend to be larger.

notion explicit, so certain **session-oriented ideas can be implemented naturally** or rendered unnecessary.

Relieving the transport protocol of the responsibility of verifying end-to-end delivery should permit **intellectually richer research on transport protocols.** A good deal of research on transport, and communication in general, starts with the straightjacket that layer 4 must verify delivery to the application; as a result, TCP has been over-engineered. In a 5-layer architecture, transport design can focus solely on data movement issues, engendering a philosophy of limited transport protocols that solve limited problems.

4.3 Discovering Intermediaries

One significant challenge is how to name, discover, and locate in-network services. (Our prototype currently fakes dynamic service discovery, reading available services from a small static list.)

An appealing idea is to use the proven existing mechanism of IPv4 *anycast* [54]. Anycast is a poorly documented "route configuration trick" that permits widely separated sites to share the same IP address. The sites advertise the same "host route" (actually, a /32 mask) to the shared IP address. Routing protocols (and BGP) spread the information, correctly counting the number of hops (or, in the case of BGP, domains). Any router that learns multiple paths to the shared IP address will prefer the most direct path; in different parts of the world, different service sites will prove closest to clients in their parts of the world. The effect is for clients to use the closest server. Currently the "F" root DNS server uses anycast to map 192.5.5.41 to 37 separate sites (see `http://f.root-servers.org`). Other root DNS servers plan also to start using anycast.

Anycast effectively provides automatic routing to the nearest instance of an IP address; therefore, an intermediate service could advertise itself in DNS in the usual way, and clients would use the IP address returned by DNS in the usual way to contact the nearest copy of the intermediate service. Because anycast is proven (though little-known) technology that promises to allow DNS-based lookup of intermediate service sites while requiring no new mechanisms, our plan is to

investigate anycast as the first option for service discovery.

4.4 Chaining Intermediaries

The two endpoints, and a variable number of intermediaries, are chained together using a synchronous two-phase setup protocol. A session initiator (client) specifies which intermediate services it desires, permits, or prohibits to exist along the network path to the other endpoint. The client may also specify the order in which services appear. A two-phase negotiation establishes a session. At the end of the first phase, the client's list of services is presented to the service site, which can alter the list. At the end of the second phase, the client has an indication of which services have been approved, in which order, along the path. It may then either accept or reject this result. This protocol protects the possibly-conflicting interests of both clients and servers. The session setup messages specify which transport protocol to use between consecutive sites. Connection-oriented transport connections (e.g., TCP) are started as layer-5 session setup messages hop back to the initiator during the second phase of setup.

The obvious drawback of a synchronous setup protocol is startup latency. There are two sorts of startup costs:

1. Setting up the L5 session itself. That is, contacting and initializing each intermediary and the opposite endpoint. (The session layer has much more work to do at startup than, say, TCP, and people already complain about TCP startup latency.)

2. Setting up L4 transport connections between pairs of sites along the path.

The second issue is easily solved: cache transport connections. This is desirable anyway because, as discussed above, there are considerable performance advantages to leaving transport connections running for a long time among intermediaries and between intermediaries and endpoints. Our current implementation caches transport connections and starts new transport connections only when there is no cached connection to the next site in the path.

The first issue is harder. We plan to investigate multicast to speed up L5 connection establishment. Since we expect there to be relatively few intermediate service sites (relative to the number of hosts), there should not be too many multicast trees, and the trees should tend to persist long-term. L5 setup might be very well suited to the characteristics of Internet multicast. On the other hand, we also plan for intermediaries to exist at anycast-ed addresses. Mixing anycast and multicast is uncharted territory, a research topic by itself.

4.5 Intermediaries As Bottlenecks

Intermediaries can be bottlenecks in several ways: they can be performance bottlenecks, can make applications less reliable by introducing additional points of failure, and they can provide additional targets for privacy and denial-of-service attacks. These problems are inherent to the concept of an intermediary—whether transparent or visible, managed or unmanaged—and are not made worse by this work; nevertheless, this work encourages the use of intermediaries and therefore should bear some of the responsibility of solving these problems.

To a degree, anycast ameliorates each of these concerns. With respect to reliability and denial-of-service, by spreading load across copies of an intermediate service, anycast ensures that failure of one of N intermediate service sites should not affect greatly more than 1/N of the active sessions. Anycast also affects performance, ensuring that each intermediate site will contact the closest copy of the next intermediate service.

Ideas for how to actually *survive* intermediate site failure include allowing applications to indicate during L5 setup whether they want intermediate state handled as (1) "soft," meaning the applications can tolerate loss of state, (2) hard cookies, meaning that the application allows the

intermediate service to pass its state on to them, or (3) stored in a general, replicated "state storage service." The state storage service would allow session state to be like doctor's records: when you change doctors, your records are transferred to the new physician.

5 Development and Evaluation Plan

We have already implemented a version of the protocol and its API for clients, intermediaries, and servers. We have implemented three intermediate services. In fact, for a usability study, we implemented all three services twice, once when endpoints use the session protocol to explicitly place the service in the L5 connection, and a second time when the endpoints use an ordinary TCP connection and the service spies on packets. Services are much easier to implement using the L5 interface, and clients and servers are no harder to implement.

The three services are word filtering (code shown above), web caching and annotation, and packet marking (e.g., [40, 42, 7]) to help with IP traceback. The first two services are meant to illustrate stateless and stateful operation, respectively.

We have also implemented transparent mobility whereby an endpoint can relocate to a new IP address without applications (that use the L5 API) noticing because L4 connections are torn down and replaced while L5 connections remain running.

Further work will continue in three directions:

1. Continued design and implementation to address outstanding architectural issues.
2. Simulations to evaluate the performance claims made above.
3. Investigation of how the presence of a protocol at layer 5 might suggest changes to the protocol stack generally and to the transport layer specifically.

Each is discussed in a subsection below.

5.1 Architectural Issues

Besides the issues mentioned in Sections 4.3 through 4.5 above, we will expand our prototype implementation in these ways:

- Give the ISP(s) as well as endpoints the ability to examine and alter the list of intermediaries inserted into a particular session. Just as the two endpoints might have divergent interests, so too the ISP might have interests divergent from the endpoints.
- Expand our capability, which is currently primitive, for securely reconfiguring intermediate sites while a session is running.
- Improve how intermediaries report errors, and how they react to failures.

There are a few reasons for implementing. One is to expose any overlooked issues or faulty assumptions; i.e., to ensure that the design is complete. Another reason is to produce a prototype that others can examine and use. A third reason is that many of the potential advantages of having a session layer are non-quantitative features such as convenience, usability, and elegance. It is not realistic to try to evaluate such qualities via simulation. We run our implementations both in the usual wired/wireless LAN settings and on PlanetLab, a worldwide distributed testbed [17].

5.2 Simulation

We will perform simulations to answer the performance-related questions that are best suited to simulation, either because they presume widespread deployment or because they require evaluating many different experimental variables such as topology and link parameters. As mentioned above, we have some preliminary simulation results for the congestion control and MTU studies.

Listed below are several studies that we will undertake; many more are possible, though left out because of space limitations. Each study is briefly summarized by an intuitive hypothesis; one or more experiments to test the hypothesis; and what we hope to learn from the experiment.

Pipeline Parallelism.
Hypothesis: Long-lived TCP connections with plenty of data (from multiplexed sessions) may be able to improve throughput, because of pipeline parallelism: there are several receive buffers to be filled simultaneously. Under some circumstances, end-to-end throughput may be greater over a session connection than over a single transport connection.
Experiment: Simulate the throughput achieved by running an infinite data stream over a sequence of TCP connections. Parameters to vary include the number of TCP connections in series, loss rate, the maximum window size, and the round-trip time. These are the parameters commonly understood to determine TCP's throughput [33]. If throughput improvement is seen for an infinite data stream, simulate successively smaller finite streams.
What will be learned: The circumstances, if any, under which layer-5 sessions can move data faster than individual transport connections.

There is preliminary evidence that such "split" or "cascaded" TCP connections can be faster than a single connection between the same endpoints. Spatscheck and Sibal discovered such an effect by accident in their study of *TPOT*, a TCP-level mechanism for redirecting TCP connections from a web site to a web cache. Also, we have completed a mathematical characterization of the performance of such "cascaded" TCP connections and found speedups of up to 50% [51].

More Rapid Recovery and Congestion Control of an Individual Session.
Hypothesis: Physically shorter TCP connections should have smaller RTT and RTTVAR, and hence should timeout and re-transmit more rapidly. More rapid re-transmission and response to congestion should be advantageous for both the individual connection and the entire network when there are losses due to congestion. **Experiment:** Simulate a congested set of TCP connections of various lengths. Determine throughput as a function of RTT and connection length. Determine RTTVAR as a function of connection length.
What will be learned: We should learn the relationship, if any, between the physical length of a TCP connection and the speed with which it responds to congestion. We should also learn to what extent more rapid response improves the efficiency of the network. (This type of study has not been undertaken before, because previous research has never had the luxury of considering the physical length of a TCP as a variable.)

Better Throughput and Congestion Control from Long-lived Connections.
Hypothesis: Data moves more efficiently over long-lived TCP connections than over short-lived connections. There is more history to draw from when sizing the congestion window, and a steady stream of acks to continue clocking data.
Experiment: Simulate streams of various length sent over TCP connections, and determine the relationship between throughput and stream length. Vary stream burstiness and the number of congestion-caused restarts. **What will be learned:** There is considerable prior work suggesting that the hypothesis is true: short, bursty TCP connections are a known performance problem, and both *Ensemble TCP* [21] and the *Congestion Manager* [4] have successfully pooled history from several previous TCP connections in order to more accurately control future connections to the same endpoints.

MTU Study.
Hypothesis 1: Portions of the Internet core could be carrying larger frames than they are currently

carrying. **Hypothesis 2:** Intermediate sites in the network that terminate an "edge" transport connection then re-send the data over a "core" transport connection with a higher MTU could increase overall Internet capacity utilization and efficiency, and end-to-end throughput.
Experiment 1: Determine MTU sizes on links throughout the Internet core. **Experiment 2:** Simulate the effects of terminating a small-MTU transport connection at an intermediate site in the core and then re-sending the data from that site over another large-MTU connection. Vary the MTUs using numbers drawn from the first experiment.
What will be learned: We hope to create an "MTU map" of the Internet—similar to a topographic map of a landscape—and quantify the untapped capacity of the Internet.

Better Congestion Control from More Endpoints.
Hypothesis: Breaking each physically long TCP connection into several short ones yields more TCP senders and hence more congestion controllers. As a result, it is possible that overall congestion will be reduced. More total congestion controllers potentially can back off from congestion faster. On the other hard, more total senders might result in more aggressive increases, and might cause *more* congestion. **Experiment:** Simulate a large number of TCP connections that intersect at a small number of routers, and measure the congestion they cause over a long time. Then simulate the same setting where each TCP connection is divided into N physically shorter connections. Once again measure the congestion. **What will be learned:** The relationship, if any, between the number of congestion control points and the incidence and severity of congestion in the Internet as a whole.

5.3 Interaction with the Transport Layer

In the current design of the Internet, the transport protocol is assigned three responsibilities: ensuring end-to-end delivery semantics, flow control, and congestion control. If delivery semantics is re-assigned to layer 5, it raises the question of what layer 4 (if it exists at all) should be doing. More generally—and perhaps granting that layer 3 will remain as some form of IP—how should the three functions be implemented? The best solutions may not include stacked architectures.

One possibility is for layer 3 to address hop-to-hop issues, layer 4 to address stage-to-stage issues, and layer 5 end-to-end. An intermediary would be a natural definition of a "stage," and the primary stage-to-stage issue is congestion since delivery semantics and flow control are naturally end-to-end issues.

One idea we will investigate in simulation is for routers to keep each other continually informed of the conditions of their queues. As a certain router becomes more congested, its upstream neighbors discover the fact and throttle themselves. (Throttling could consist of changing RED parameters.) If throttling fails to cure the downstream congestion, or if congestion results at any of the now-throttled upstream routers, then throttling continues upstream. Upstream throttling may continue until, at the limit, hosts are reached. However, congestion is initially assumed to be a "local" problem, with the problem becoming ever more global only as local remedies prove inadequate.

Unthrottling happens naturally because the information passed among routers is not a discrete simple binary signal ("I am congested now"), but rather a continuous and richer measure of conditions downstream. When downstream conditions tighten, upstream throttles. When downstream conditions ease, upstream un-throttles.

It is typical for core routers to be linked to very few others; Spring et al. report that approximately 90% are attached to 5 or fewer neighbors [46], and this figure includes highly connected PoP routers. Therefore, the amount of information exchanged should not be burdensome, while an indication of congestion can have wide effect after only a few levels of upstream notification.

Furthermore, adjacent core routers are set up by management policy, not by automatic neighbor discovery. Therefore, it is reasonable to suppose that adjacent core routers could perform a somewhat heavyweight initialization procedure that would set the parameters for a congestion control

protocol that they would run among themselves. This would allow different router implementations, with different queueing measures and policies, to interact. The central requirement of the inter-router congestion control protocol would be a syntax whereby any router could express to any other its *relative* conditions at the moment.

The information exchanged by routers could be rich enough to help cure the congestion. For example, if a router reported the condition of all its outgoing queues, upstream routers might be able to select different routes – routers in a common AS typically already share a common route map and/or set of MPLS LSPs for the AS.

6 Prior and Related Work

As the introduction mentions, the trend toward in-network services is strong and has been noticed by many. Although this proposal's approach to the situation is unique, there are many categories of related work.

One category is work that addresses Internet architecture. Clark and Blumenthal [18] discuss how social and political developments more so than technical developments are killing the original end-to-end Internet model. RFC 2775 [14] and a draft RFC by Carpenter and Austein [15] discuss the reduced end-to-end transparency of the modern Internet. The pros and (mostly) cons of a particular type of intermediary, NATs, are examined at length in RFC 2993 [23]. The limitations imposed by NATs are implicitly revealed in RFC 3235 [41], which explains how applications should be (re-)written to cope with NATs. RFC 3234 [16] provides a taxonomy of a wide variety of in-network devices (22 types) and evaluates whether and how each may be considered an intermediary, according to 8 different criteria.

Another category is work whose goal and/or mechanism would be subsumed by a session protocol at layer 5. This includes various proposals for TCP endpoint migration and persistent connections, mostly done in the context of mobility [19, 24, 57, 35, 31, 55, 43, 44]; work on "ensemble TCP" (methods for sharing path and/or congestion control information among several TCP endpoints at the same host) [3, 21, 53]; a mechanism for explicitly routing traffic to intermediaries [37]; and various *ad hoc* application-level "session" mechanisms [28, 45].

A third category of prior work is that which fits into the session-layer framework as a specific intermediate service. This includes NATs, firewalls, "performance-enhancing proxies" (PEPs) [9], transport relays, and possibly QoS reservation [12].

Overlay networks are an approach to application-specific routing that have lately drawn much attention. Overlays first drew interest as a method for running experimental protocols over various "backbones"—MBONE, 6BONE, ABONE. News coverage of peer-to-peer file sharing networks have since excited interest in doing nearly everything on overlays: expedited routing [1], limited QoS services [50], even capturing ongoing denial-of-service attacks [49]. The overlay concept is closely related to the proposed work, but bears an important difference: an overlay is purely a routing mechanism. Sites that communicate via overlays are all endpoints, often interacting in client-server fashion dubbed "peering" because every site is a server as well as client. Overlays do not provide concepts for binding together several sites into a single distribution computation. These concepts are present in the proposed work in the form of a session ID, explicit setup and teardown, an error reporting protocol, and reconfiguration.

The session protocol's explicit setup of the routing path calls to mind several other mechanisms that also setup source-routed "connections," albeit for different reasons. These include ATM virtual circuits, MPLS, and IP loose source routing. Also related are feature-rich protocols for telephony such as SCTP (which allows multi-homed connections) and SIP (which can manage a set of transport connections in order to provide telephone conferencing).

A final category is complementary work, specifically that on "middlebox management"—how endpoints might control intermediaries. This work includes earlier mechanisms for "opening pin-

holes" (SOCKS [30]), using HTTP as a transport [32], Realm-specific IP [10, 11], and the work of the IETF's MIDCOM working group [52, 47, 6, 38]. The proposed work does not conflict with and can co-exist with the MIDCOM effort, whose goal is to develop a control protocol that permits application intelligence to be located outside the network element that actually captures packets. The MIDCOM protocol, once developed, will allow communication and control between the packet capture element and a remotely located application-aware intermediary.

7 PI's Qualifications

The PI wrote the first paper on Mobile IP [25]. The method described in that paper is much simpler and arguably better and more practical than the outcome of the IETF's Mobile IP Working Group. That work was funded by an NSF grant, *Internetworking for Mobile Computers*. The work also produced a 50-page draft RFC that fully specified the protocol. The PI's students also wrote and distributed an implementation of Mobile IP and a device driver for NCR's *WaveLAN* wireless LAN—a pre-802.11 product that was popular among researchers in the early 1990s.

The PI was one of the founders of the field of mobility, starting in the late 1980s, and was program chair of the first Mobicom conference. The first phase of his research (funded by ONR Young Investigator grant *General Purpose Personalized Mobile Computing*) sought to implement services so that they *hid the effects of mobility*. For example, Mobile IP is IP that hides mobility. Other examples included file service and a method of relocating X11 windows onto new displays. The second phase of his research (funded by DARPA grant *Graceful Degradation Techniques for Mobile Computing in Heterogeneous Environments*) sought to *expose and manage* the unavoidable consequences of mobility. (The proposed work is similar in philosophy.) This work included the study of how to place and control proxies between a bandwidth-limited mobile host and the Internet so that the proxy could help to compensate for the limited bandwidth [56].

Recently, the PI has been working on novel ultra-low-energy routing protocols, especially for mobile situations. The basic idea is to break routing functions into pieces, and to permit the end system to elect which pieces should be executed and when they should be executed.

8 Potential Broader Impacts

The proposed session layer allows network intermediaries to be placed anywhere rather than only at network choke points. This capability decouples service provision from network ownership. Therefore, the session layer, if widely deployed, could enable a new industry of intermediate in-network services. This is the primary possible broader impact of this work.

To encourage wide experimentation with and dissemination of the session layer protocol, we have created a page on the popular `sourceforge` web site, where we maintain the current implementation. This page will be kept up to date as the work progresses.

9 Educational Plan

Prof. Duchamp already teaches two networking courses, *TCP/IP Networking* and its successor, *Advanced Internet Protocols*. The latter is project-oriented and students have used PlanetLab in the past in this course. Prof. Duchamp will also create and teach course on network performance evaluation. This course will introduce students to analytic modeling, simulation (using ns), and benchmarking (on PlanetLab). The proposed work will be a rich source of projects in both courses.

References

[1] D. Andersen et al. *Resilient Overlay Networks.* in Proc. 18th ACM Symp. on Operating Systems Principles (SOSP '01), pp. 131-145, October 2001.

[2] H. Balakrishnan et al. *TCP Performance Implications of Network Path Asymmetry.* RFC 3449. December 2002.

[3] H. Balakrishnan et al. *TCP Behavior of a Busy Internet Server: Analysis and Improvements.* in Proc. IEEE INFOCOM, pp. 252-262, March 1998.

[4] H. Balakrishnan, H.S. Rahul, and S. Seshan. *An Integrated Congestion Management Architecture for Internet Hosts.* in Proc. ACM SIGCOMM '99, pp. 175-187, August 1999.

[5] H. Balakrishnan and S. Seshan. *The Congestion Manager.* RFC 3124. June 2001.

[6] M. Barnes, ed. *Middlebox Communications (MIDCOM) Protocol Evaluation.* Work in progress. draft-ietf-midcom-protocol-eval-06.txt November 2002.

[7] S. Bellovin. *ICMP Traceback Messages.* Work in progress (expired). draft-bellovin-itrace-00.txt March 2000.

[8] S. Blake et al. *An Architecture for Differentiated Service.* RFC 2475. December 1998.

[9] J. Border et al. *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations.* RFC 3135. June 2001.

[10] M. Borella et al. *Realm Specific IP: Framework.* RFC 3102. October 2001.

[11] M. Borella et al. *Realm Specific IP: Protocol Specification.* RFC 3103. October 2001.

[12] R. Braden, ed. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification.* RFC 2205. September 1997.

[13] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: An Overiew.* RFC 1633. June 1994.

[14] B. Carpenter. *Internet Transparency.* RFC 2775. February 2000.

[15] B. Carpenter and R. Austein. *Recent Changes in the Architectural Principles of the Internet.* Work in progress (expired). draft-iab-arch-changes-00.txt February 2002.

[16] B. Carpenter and S. Brim. *Middleboxes: Taxonomy and Issues.* RFC 3234. February 2002.

[17] B. Chun et al. *PlanetLab: An Overlay Testbed for Broad-Coverage Services.* Submitted for publication. http://www.planet-lab.org/php/pdn.php

[18] D.D. Clark and M.J. Blumenthal. *Rethinking the Design of the Internet: The End to End Arguments vs. the Brave New World.* in Proc. 28th Telecommunications Policy Research Conference, September 2000. http://www.tprc.org/abstracts00/rethinking.pdf

[19] D. Crocker. *Transport Context Names (TCN): Concepts and Facilities.* Work in progress (expired). draft-crocker-tcn-00.txt February 1995.

[20] J. Day. *The (Un)Revised OSI Reference Model.* Computer Communication Review, 25(5):39-55, October 1995.

[21] L. Eggert, J. Heidemann, and J. Touch. *Effects of Ensemble TCP.* Computer Communication Review, 30(1):15-29, January 2000.

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

[23] T. Hain. *Architectural Implications of NAT.* RFC 2993. November 2000.

[24] C. Huitema. *Multi-homed TCP.* Work in progress (expired). draft-huitema-multi-homed-01.txt May 1995.

[25] J. Ioannidis, D. Duchamp, and G.Q. Maguire Jr. *IP-Based Protocols for Mobile Internetworking.* in Proc. ACM SIGCOMM '91, pp. 235-245, September 1991.

[26] M. Kaat. *Overview of 1999 IAB Network Layer Workshop.* RFC 2956. November 2000.

[27] E. Kohler, M. Handley, and S. Floyd. *Designing DCCP: Congestion Control Without Reliability.* Submitted for publication.

[28] D. Kristol and L. Montulli. *HTTP State Management Mechanism.* RFC 2965. October 2000.

[29] M. Laubach and J. Halpern. *Classical IP and ARP Over ATM.* RFC 2225. April 1998.

[30] M. Leech et al. *SOCKS Protocol Version 5.* RFC 1928. March 1996.

[31] D.A. Maltz and P. Bhagwat. *MSOCKS: An Architecture for Transport Layer Mobility.* in Proc. IEEE INFOCOM, pp. 1037-1045, March 1998.

[32] K. Moore. *On the Use of HTTP as a Substrate.* RFC 3205. February 2002.

[33] J. Padhye et al. *Modeling TCP Throughput: A Simple Model and its Empirical Validation.* in Proc. ACM SIGCOMM '98, pp. 303-314, September 1998.

[34] V. Paxson and M. Allman. *Computing TCP's Retransmission Timer.* RFC 2988. November 2000.

[35] X. Qu, J.X. Yu, and R.P. Brent. *A Mobile TCP Socket.* in Proc. Intl. Conf. on Software Engineering, November 1997.

[36] K. Ramakrishnan, S. Floyd, and D. Black. *The Addition of Explicit Congestion Notification (ECN) to IP.* RFC 3168. September 2001.

[37] P. Rodriguez, S. Sibal and O. Spatscheck. *TPOT: Translucent Proxying of TCP.* Computer Communication, 24(2):249-255, 2001.

[38] J. Rosenberg et al. *STUN – Simple Traversal of UDP Through Network Address Translators.* Work in progress. draft-ietf-midcom-stun-05.txt December 2002.

[39] J. Rosenberg et al. *SIP: Session Initiation Protocol.* RFC 3261. June 2002.

[40] S. Savage et al. *Practical Network Support for IP Traceback.* in Proc. ACM SIGCOMM 2000, pp. 295-306, August 2000.

[41] D. Senie. *Network Address Translator (NAT)-Friendly Application Design Guidelines.* RFC 3235. January 2002.

[42] A.C. Snoeren et al. *Hash-Based IP Traceback.* in Proc. ACM SIGCOMM 2001, pp. 3-14, August 2001.

[43] A.C. Snoeren and H. Balakrishnan. *An End-to-End Approach to Host Mobility.* in Proc. Mobicom 2000, pp. 155-166, August 2000.

[44] A.C. Snoeren, D.G. Andersen, and H. Balakrishnan. *Fine-Grained Failover Using Connection Migration.* in Proc. 3rd USENIX Symp. on Internet Technologies and Systems (USITS), pp. 221-232, August 2001.

[45] S. Spero. *Session Control Protocol (SCP).* http://www.w3.org/Protocols/HTTP-NG/http-ng-scp.html

[46] N. Spring, R. Mahajan, and D. Weatherall. *Measuring ISP Topologies with Rocketfuel.* in Proc. ACM SIGCOMM '02, pp. 133-145, August 2002.

[47] P. Srisuresh et al. *Middlebox Communication Architecture and Framework.* RFC 3304. August 2002.

[48] R. Stewart et al. *Stream Control Transmission Protocol.* RFC 2960. October 2000.

[49] R. Stone. *CenterTrack: An IP Overlay Network for Tracking DoS Floods.* in Proc. USENIX Security Symp. '00, pp. 199-212, July 2000.

[50] R. Subramanian et al. *OverQoS: Offering Internet QoS Using Overlays.* Computer Communication Review, 33(1):11-16, January 2003.

[51] A. Sundararaj and D. Duchamp. *Analytical Characterization of the Throughput of a Split TCP Connection.* Submitted for publication.

[52] R.P. Swale et al. *Middlebox Communications (midcom) Protocol Requirements.* RFC 3303. August 2002.

[53] J. Touch. *TCP Control Block Interdependence.* RFC 2140. April 1997.

[54] B. Woodcock. *Best Practices in IPv4 Anycast Routing.* http://www.pch.net/resources/tutorial/anycast

[55] V. Zandy and B. Miller. *Reliable Network Connections.* in Proc. Mobicom 2002, pp. 95-106, September 2002.

[56] B. Zenel and D. Duchamp. *A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment.* in Proc. Third Ann. ACM/IEEE Intl. Conf. on Mobile Computing and Networking, September 1997.

[57] Y. Zhang and S. Dao. *A "Persistent Connection" Model for Mobile and Distributed Systems.* in Proc. 4th Intl. Conf. on Computer Communication and Networks (ICCCN), September 1995.