Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives

Armando Fox Steven D. Gribble Yatin Chawathe
Eric A. Brewer
{fox,gribble,yatin,brewer}@cs.berkeley.edu

Today's Internet clients vary widely with respect to both hardware and software properties: screen size, color depth, effective bandwidth, processing power, and the ability to handle different data formats. The order-of-magnitude span of this variation is too large to hide at the network level, making application-level techniques necessary. We show that on-the-fly adaptation by transformational proxies *is a widely applicable, cost-effective, and flexible technique for addressing all these types of variation. To support this claim, we describe our experience with datatype-specific distillation (lossy compression) in a variety of applications. We also argue that placing adaptation machinery in the network infrastructure, rather than inserting it into end servers, enables incremental deployment and amortization of operating costs. To this end, we describe a programming model for large-scale interactive Internet services and a scalable cluster-based framework that has been in production use at UC Berkeley since April 1997. We present a detailed examination of TranSend, a scalable transformational Web proxy deployed on our cluster framework, and give descriptions of several handheld-device applications that demonstrate the wide applicability of the proxy-adaptation philosophy.*

1 Infrastructural On-the-Fly Adaptation Services

1.1 Heterogeneity: Thin Clients and Slow Networks

The current Internet infrastructure includes an extensive range and number of clients and servers. Clients vary along many axes, including screen size, color depth, effective bandwidth, processing power, and ability to handle specific data encodings, e.g., GIF, PostScript, or MPEG. As shown in tables 1 and 2, each type of variation often spans orders of magnitude. High-volume devices such as smart phones [12] and smart two-way pagers will soon constitute an increasing fraction of Internet clients, making the variation even more pronounced.

These conditions make it difficult for servers to provide a level of service that is appropriate for every client. Application-level adaptation is required to provide a *meaningful* Internet experience across the range of client capabilities. Despite continuing improvements in client computing power and connectivity, we expect the high end to advance roughly in parallel with the low end, effectively maintaining a gap between the two and therefore the need for application-level adaptation.

Platform	SPEC92/ Memory	Screen Size	Bits/ pixel
High-end PC	200/64M	1280x1024	24
Midrange PC	160/32M	1024x768	16
Typ. Laptop	110/16M	800x600	8
Typical PDA	low/2M	320x200	2

Table 1: **Physical variation among clients**

Network	Bandwidth (bits/s)	Round-Trip Time
Local Ethernet	10-100 M	0.5 - 2.0 ms
ISDN	128 K	10-20 ms
Wireline Modem	14.4 - 56 K	350 ms
Cellular/CDPD	9.6 - 19.2 K	0.1 - 0.5 s

Table 2: **Typical Network Variation**

1.2 Approach: Infrastructural Proxy Services

We argue for a *proxy-based approach* to adaptation, in which proxy agents placed between clients and servers perform aggressive computation and storage on behalf of clients. The proxy approach stands in contrast to the *client-based approach*, which attempts to bring all clients up to a least-common-denominator level of functionality (e.g. text-only, HTML-subset compatibility for thin-client Web browsers), and the *server-based approach*, which attempts to insert adaptation machinery at each end server. We believe the proxy approach directly confers three advantages over the client and server approaches.

- **Leveraging the installed infrastructure through incremental deployment.** The enormous installed infrastructure, and its attendant base of existing content, is too valuable to waste; yet some clients cannot handle certain data types effectively. A compelling solution to the problem of client and network heterogeneity should allow interoperability with existing servers, thus enabling incremental deployment while evolving content formats and protocols are tuned and standardized for different target platforms. A proxy-based approach lends itself naturally to transparent incremental deployment, since an application-level proxy appears as a server to existing clients and as a client to existing servers.

- **Rapid prototyping during turbulent standardization cycles.** Software development on "Internet time" does not allow for long deployment cycles. Proxy-based adaptation provides a smooth path for rapid prototyping of new services, formats, and protocols, which can be deployed to servers (or clients) later if the prototypes succeed.

- **Economy of scale.** Basic queueing theory shows that a large central (virtual) server is more efficient in both cost and utilization (though less predictable in per-request performance) than a collection of smaller servers; standalone desktop systems represent the degenerate case of one "server" per user. This supports the argument for Network Computers [28] and suggests that collocating proxy services with infrastructural elements such as Internet points-of-presence (POPs) is one way to achieve effective economies of scale.

Large-scale network services remain difficult to deploy because of three fundamental challenges: scalability, availability and cost effectiveness. By scalability, we mean that when the load offered to the service increases, an incremental and linear increase in hardware can maintain the same per-user level of service. By availability, we mean that the service as a whole must be available 24×7, despite transient partial hardware or software failures. By cost effectiveness, we mean that the service must be economical to administer and expand, even though it potentially comprises many workstation nodes operating as a centralized cluster or "server farm". In section 3 we describe how we have addressed these challenges in our cluster-based proxy application server architecture.

1.3 Contributions and Map of Paper

In section 2 we describe our measurements and experience with *datatype-specific distillation and refinement*, a mechanism that has been central to our proxy-based approach to network and client adaptation. In section 3 we introduce a generalized "building block" programming model for designing and implementing adaptive applications, describe our implemented cluster-based application server that instantiates the model, and present detailed measurements of a particular production application: TranSend, a transformational Web proxy service. In section 4 we present case studies of other services we have built using our programming model, some of which are in daily use by thousands of users, including the Top Gun Wingman graphical Web browser for the 3Com PalmPilot handheld device. We discuss related work in section 5, and attempt to draw some lessons from our experience and guidelines for future research in section 6.

2 Adaptation via Datatype Specific Distillation

We propose three design principles that we believe are fundamental for addressing client variation most effectively.

1. **Adapt to client variation via datatype-specific lossy compression.**

   Datatype-specific lossy compression mechanisms can achieve much better compression than "generic" compressors, because they can make intelligent decisions about what information to throw away based on the semantic type of the data. For example, lossy compression of an image requires discarding color information, high-frequency components, or pixel resolution. Lossy compression of video can additionally include frame rate reduction. Less obviously, lossy compression of formatted text requires discarding some formatting information but preserving the actual prose. In all cases, the goal is to preserve information that has the highest semantic value. We refer to this process generically as *distillation*. A distilled object allows the user to decide whether it is worth asking for a *refinement*: for instance, zooming in on a section of a graphic or video frame, or rendering a particular page containing PostScript text and figures without having to render the preceding pages.

2. **Perform adaptation on the fly.** To reap the maximum benefit from distillation and refinement, a distilled representation must target specific attributes of the client. The measurements reported in section 2.1 show that for typical images and rich-text, distillation time is small in practice, and end-to-end latency is reduced because of the much smaller number of bytes transmitted over low-bandwidth links. *On-demand distillation* provides an easy path for incorporating support for new clients, and also allows distillation aggressiveness to track (e.g.) significant changes in network bandwidth, as might occur in vertical handoffs between different wireless networks [39]. We have successfully implemented useful distillation "workers" that serve clients spanning an order of magnitude in each area of variation, and we have generalized our approach into a common framework, which we discuss in section 3.

3. **Move complexity away from both clients and servers.** Application partitioning arguments have long been used to keep clients simple [40]. However, adaptation through a shared infrastructural proxy enables incremental deployment and legacy client support, as we argued in section 1.2. Therefore, on-demand distillation and refinement should be done at an intermediate proxy that has access to substantial computing resources and is well-connected to the rest of the Internet.

Table 3 lists the "axes" of compression corresponding to three important datatypes: formatted text, images, and video streams. We have found that order-of-magnitude size reductions are often possible without destroying the semantic content of an object (e.g. without rendering an image unrecognizable to the user).

Semantic Type	Specific encodings	Distillation axes
Image	GIF, JPEG, PPM, Postscript	Resolution, color depth, color palette
Text	Plain, HTML, Postscript, PDF	Richness (heavily formatted vs. simple markup vs. plaintext)
Video	NV, H.261, VQ, MPEG	Resolution, frame rate, color depth, progression limit (for progressive encodings)

Table 3: **Three important types and the distillation axes corresponding to each.**

2.1 Performance of Distillation and Refinement On Demand

We now describe and evaluate datatype-specific distillers for images and rich-text.[1] The goal of this section is to support our claim that in the majority of cases, *end-to-end latency is reduced by distillation,* that is, the time to produce a useful distilled object on today's workstation hardware is small enough to be more than compensated by the savings in transmission time for the distilled object relative to the original.

2.1.1 Images

We have implemented an image distiller called *gifmunch*, which implements distillation and refinement for GIF [25] images, and consists largely of source code from the NetPBM Toolkit [35]. Figure 1 shows the result of running gifmunch on a large color GIF image of the Berkeley Computer Science Division's home building, Soda Hall. The image of Figure 1a measures 320×200 pixels—about 1/8 the total area of the original 880×610—and uses 16 grays, making it suitable for display on a typical handheld device.

Due to the degradation of quality, the writing on the building is unreadable, but the user can request a refinement of the subregion containing the writing, which can then be viewed at full resolution.

Image distillation can be used to address all three areas of client variation:

- **Network variation**: The graphs in figure 2

[1] A distiller for real-time network video streams is described separately, in [1].

Left (a) is a distilled image of Soda Hall, and above (b) illustrates refinement. (a) occupies 17 KB at 320x200 pixels in 16 grays, compared with the 492 KB, 880x600 pixel, 249 color original (not shown). The refinement (b) occupies 12 KB. Distillation took 6 seconds on a SPARCstation 20/71, and refinement took less than a second.

Figure 1: Distillation example

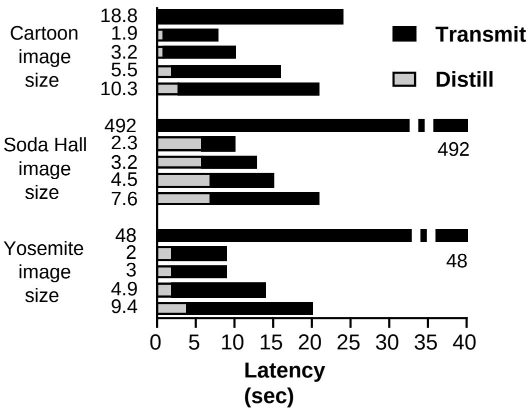

Figure 2: End-to-end latency for images with and without distillation. Each group of bars represents one image with 5 levels of distillation; the top bar represents no distillation at all. The y-axis number is the distilled size in kilobytes (so the top bar gives the original size). Note that two of the undistilled images are off the scale; the Soda Hall image is off by an order of magnitude.

depict end-to-end client latency for retrieving the original and each of four distilled versions of a selection of GIF images: the top set of bars is for a cartoon found on a popular Web page, the middle set corresponds to a large photographic image, and the bottom to a computer rendered image. The images were fetched using a 14.4Kb/s modem with standard compression (V.42bis and MNP-5) through the UC Berkeley PPP gateway, via a process that runs each image through gifmunch. [2] Each bar is segmented to show the distillation latency and transmission latency separately. Clearly, even though distillation adds latency at the proxy, it can result in greatly reduced end-to-end latency. This shows that on the fly distillation is not prohibitively expensive.

- **Hardware variation**: A "map to 16 grays" operation would be appropriate for PDA-class clients with shallow grayscale displays. We can identify this operation as an effective lossy compression technique precisely because we know we are operating on an image, regardless of the particular encoding, and the compression achieved is significantly better than the 2-4 times compression typically achieved by "generic" lossless compression (design principle #1).
- **Software variation**: Handheld devices such as the 3Com PalmPilot frequently have built-in support for proprietary image encodings only. The ability to convert to this format saves code space and decoding latency on the client (design principle #3).

2.1.2 Rich-Text

We have also implemented a rich-text distiller that performs lossy compression of PostScript-encoded

[2] The network and distillation latencies reflect significant overhead due to the naive implementation of *gifmunch* and the latency and slow-start effects of the PPP gateway, respectively. In section 3 we discuss how to overcome some of these problems, but it is worth noting that end-to-end latency is still substantially reduced even in this naive prototype implementation.

Feature	HTML	Rich Text	Post-Script
Different fonts	Y	Y	Y
Bold and Italics	Y	Y	Y
Preserves Font Size	head-ings	Y	Y
Preserves Paragraphs	Y	Y	Y
Preserves Layout	N	Y	Y
Handles Equations	N	some	Y
Preserves Tables	Y	Y	Y
Preserves Graphs	N	N	Y

Table 4: Features for postscript distillation

text using uses a third party postscript-to-text converter [34]. The distiller replaces PostScript formatting information with HTML markup tags or with a custom rich-text format that preserves the position information of the words. PostScript is an excellent target for a distiller because of its complexity and verbosity: both transmission over the network and rendering on the client are resource intensive. Table 4 compares the features available in each format. Figure 3 shows the advantage of rich-text over PostScript for screen viewing. As with image distillation, PostScript distillation yields advantages in all three categories of client variation:

- **Network variation**: Again, distillation reduces the required bandwidth and thus the end-to-end latency. We achieved an average size reduction of a factor of 5 when going from compressed PostScript to gzipped HTML. Second, the pages of a PostScript document are pipelined through the distiller, so that the second page is distilled while the user views the first page. In practice, users only experience the latency of the first page, so the difference in perceived latency is about a factor of 8 for a 28.8K modem. Distillation typically took about 5 seconds for the first page and about 2 seconds for subsequent pages.

- **Hardware variation:** Distillation reduces decoding time by delivering data in an easy-to-parse format, and results in better looking documents on clients with lower quality displays.

- **Software variation:** PostScript distillation allows clients that do not directly support PostScript, such as handhelds, to view these documents in HTML or our rich-text format. The rich-text viewer could be an external viewer similar to *ghostscript*, an applet for a Java-capable browser, or a browser plug-in rendering module.

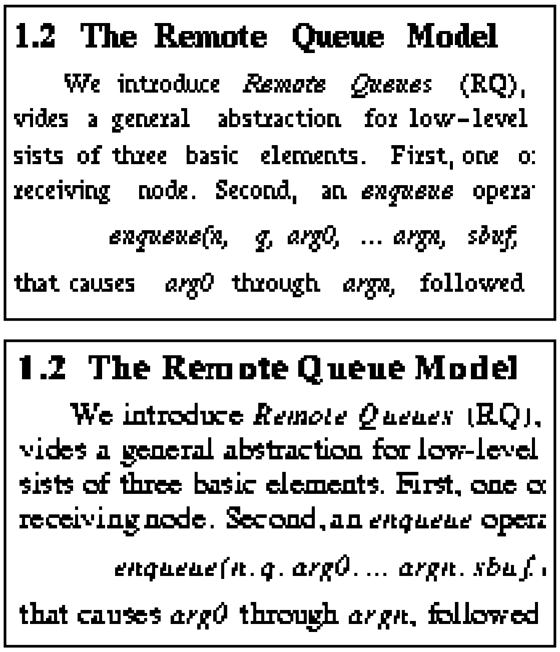

Figure 3: Screen snapshots of our rich-text (top) versus ghostview (bottom). The rich-text is easier to read because it uses screen fonts.

Overall, rich-text distillation reduces end-to-end latency, results in more readable presentation, and adds new abilities to low-end clients, such as PostScript viewing. The latency for the appearance of the first page was reduced on an average by a factor of 8 using the proxy and PostScript distiller. Both HTML and our rich-text format are significantly easier to read on screen than rendered PostScript, although they sacrifice some layout and graphics accuracy compared to the original PostScript.

2.2 Summary

High client variability is an area of increasing concern that existing servers do not handle well. We have proposed three design principles we believe are fundamental to addressing variation:

- Datatype-specific distillation and refinement achieve better compression than does lossless compression, while retaining useful semantic content and allowing network resources to be managed at the application level.

- When the proxy-to-client bandwidth is substantially smaller than the proxy-to-server bandwidth (as is the case, e.g., in wireless networks or with consumer wireline modems), on-demand distillation and refinement reduce end-to-end latency perceived by the client (sometimes by almost an order of magnitude), are more flexible than reliance on precomputed static representations, and give low-end clients new abilities such as PostScript viewing.

- Performing distillation and refinement in the network infrastructure rather than at the endpoints separates technical as well as economic concerns of clients and servers.

3 Scalable Internet Application Servers

In order to accommodate compute-intensive adaptation techniques by putting resources in the network infrastructure, we must address two important challenges:

1. Infrastructural resources are typically shared, and the sizes of user communities sharing resources such as Internet Points of Presence (POP's) is increasing exponentially. A shared infrastructural service must therefore *scale* gracefully to serve large numbers of users.

2. Infrastructural resources such as the IP routing infrastructure are expected to be reliable, with availability approaching 24×7 operation. If we place application-level computing resources such as distillation engines into this infrastructure, we should be prepared to meet comparable expectations.

In this section, we focus on the problem of deploying adaptation-based proxy services to large communities (tens of thousands of users, representative of the subscription size of a medium-sized Internet Service Provider). In particular, we discuss a cluster-friendly programming model for building interactive and adaptive Internet services, and measurements of our implemented prototype of a scalable, cluster-based server that instantiates the model. Our framework reflects the implementation of three real services in use today: TranSend, a scalable transformation proxy for the 25,000 UC Berkeley dialup users (connecting through a bank of 600 modems); Top Gun Wingman, the only graphical Web browser for the 3Com PalmPilot handheld device (commercialized by ProxiNet); and the Inktomi search engine (commercialized as HotBot), which performs over 10 million queries per day against a database of over 100 million web pages. Although HotBot does not demonstrate client adaptation, we use it to validate particular design decisions in the implementation of our server platform, since it pioneered many of the cluster-based scalability techniques generalized in our scalable server prototype.

We focus our detailed discussion and measurements on TranSend, a transformational proxy service that performs on-the-fly lossy image compression. TranSend applies the ideas explored in the preceding section to the World Wide Web.

3.1 TACC: A Programming Model for Internet Services

We focus on a particular subset of Internet services, based on *transformation* (distillation, filtering, format conversion, etc.), *aggregation* (collecting and collating data from various sources, as search engines do), *caching* (both original and transformed content), and *customization* (maintenance of a per-user preferences database that allows workers to tailor their output to the user's needs or device characteristics).

We refer to this model as TACC, from the initials of the four elements above. In the TACC model, applications are built from building blocks interconnected with simple API's. Each building block, or *worker*, specializes in a particular task, for example, scaling/dithering of images in a particular format, conversion between specific data formats, extracting "landmark" information from specific Web pages, etc. Complete applications are built by *composing* workers; roughly speaking, one worker can *chain* to another (similar to processes in a Unix pipeline), or a worker can *call* another as a subroutine or coroutine. This model of composition results in a very general programming model that subsumes transformation proxies [22], proxy filters [43], customized information aggregators, and search engines.

A *TACC server* is a platform that instantiates TACC workers, provides dispatch rules for routing network data traffic to and from them, and provides support for the inter-worker calling and chaining API's. Similar to a Unix shell, a TACC server provides the mechanisms that insulate workers from having to deal directly with low-level concerns such as data routing and exception handling, and gives workers a clean set of API's for communicating with each other, the caches, and the customization database (described below). We describe our prototype implementation of a scalable, commodity-PC cluster-based TACC server later in this section.

3.2 Cluster-Based TACC Server Architecture

We observe that clusters of workstations have some fundamental properties that can be exploited to meet the requirements of a large-scale network services (scalability, high availability, and cost effectiveness). Using commodity PCs as the unit of scaling allows the service to ride the leading edge of the cost/performance curve; the inherent redundancy of clusters can be used to mask transient failures; and "embarrassingly parallel" network service workloads map well onto networks of commodity workstations.

However, developing cluster software and administering a running cluster remain complex. A primary contribution of our work is the design, analysis, and implementation of a layered framework for building adaptive network services that addresses this complexity while realizing the sought-after economies of scale. New services can use this framework as an off-the-shelf solution to scalability, availability, and several other problems, and focus instead on the content of the service being developed.

We now describe our proposed system architecture and service-programming model for building scalable TACC servers using clusters of PC's. The architecture attempts to address the challenges of cluster computing (unwieldy administration, managing partial failures, and the lack of shared state across components) while exploiting the strengths of cluster computing (support for incremental scalability, high availability through redundancy, and the ability to use commodity building blocks). A more detailed discussion of the architecture can be found in [23].

The goal of our architecture is to separate the *content* of network services (i.e., what the services do) from their implementation, by encapsulating the "scalable network service" (SNS) requirements of high availability, scalability, and fault tolerance in a reusable layer with narrow interfaces. Application writers program to the TACC APIs alluded to in the previous section, without regard to the underlying TACC server implementation; the resulting TACC applications automatically receive the benefits of linear scaling, high availability, and failure management when run on our cluster-based TACC server.

The software-component block diagram of a scalable TACC server is shown in figure 4. Each physical workstation in a network of workstations (NOW [2]) supports one or more software components in the figure, but each component in the diagram is confined to one node. In general, the components whose tasks are naturally parallelizable are replicated for scalability, fault tolerance, or both.

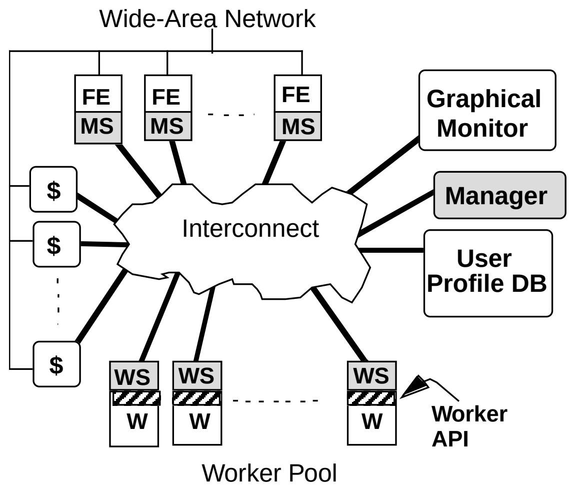

Figure 4: **Architecture of a cluster-based TACC server**: Components include front ends (FE), a pool of TACC workers (W) some of which may be caches ($), a user profile database, a graphical monitor, and a fault-tolerant load manager, whose functionality logically extends into the manager stubs (MS) and worker stubs (WS).

Front Ends provide the interface to the TACC server as seen by the outside world (e.g., HTTP server). They "shepherd" incoming requests by matching them up with the appropriate user profile from the customization database, and queueing them for service by one or more workers. Front ends maximize system throughput by maintaining state for many simultaneous outstanding requests, and can be replicated for both scalability and availability.

The **Worker Pool** consists of caches (currently Harvest [9]) and service-specific modules that implement the actual service (data transformation/filtering, content aggregation, etc.) Each type of module may be instantiated zero or more times, depending on offered load. The TACC API allows all cache workers to be managed as a single virtual cache by providing URL hashing, automatic failover, and dynamic growth of the cache pool.

The **Customization Database** stores user profiles that allow mass customization of request processing. The Manager balances load across workers and spawns additional workers as offered load fluctuates or faults occur. When necessary, it may assign work to machines in the overflow pool, a set of backup machines (perhaps on desktops) that can be harnessed to handle load bursts and provide a smooth transition during incremental growth.

The **Load Balancing/Fault Tolerance manager** keeps track of what workers are running where, autostarts new workers as needed, and balances load across workers. Its detailed operation is described in section 3.3, in the context of the TranSend implementation. Although it is a centralized agent, [23] describes the various mechanisms, including multicast heartbeat and process-peer fault tolerance, that keep this and other system components running and allow the system to survive transient component failures.

The **Graphical Monitor** for system management supports asynchronous error notification via email or pager, temporary disabling of system components for hot upgrades, and visualization of the system's behavior using Tcl/Tk [33]. The benefits of visualization are not just cosmetic: We can immediately detect by looking at the visualization panel what state the system as a whole is in, whether any component is currently causing a bottleneck (such as cache-miss time, distillation queueing delay, interconnect), what resources the system is using, and similar behaviors of interest.

The **Interconnect** provides a low-latency, high-bandwidth, scalable interconnect, such as switched 100-Mb/s Ethernet or Myrinet [32]. Its main goal is to prevent the interconnect from becoming the bottleneck as the system scales.

Components in our TACC server architecture may be replicated for fault tolerance or high availability, but we also use replication to achieve scalability. When the offered load to the system saturates the capacity of some component class, more instances of that component can be launched on incrementally added nodes. The duties of our replicated components are largely independent of each other (because of the nature of the Internet services' workload), which means the amount of additional resources required is a linear function of the increase in offered load.

3.3 Analysis of the TranSend Implementation

TranSend [22], a TACC reimplementation of our earlier prototype called Pythia [19], performs lossy Web image compression on the fly. Each TranSend worker handles compression or markup for a specific MIME type; objects of unsupported types are passed through to the user unaltered.[3]

We took measurements of TranSend using a cluster of 15 Sun SPARC Ultra-1 workstations connected by 100 Mb/s switched Ethernet and isolated from external load or network traffic. For measurements requiring Internet access, the access was via a 10 Mb/s switched Ethernet network connecting our workstation to the outside world. Many of the performance tests are based upon HTTP trace data from the 25,000 UC Berkeley dialup IP users [27], played back using a high-performance playback engine of our own design that can either generate requests at a constant rate or faithfully play back a trace according to the timestamps in the trace file.

In the following subsections we report on experiments that stress TranSend's fault tolerance, responsiveness, and scalability.

3.3.1 Self Tuning and Load Balancing

As mentioned previously, the load balancing and fault tolerance manager is charged with spawning and reaping workers and distributing internal load across them. The mechanisms by which this is accomplished, which include monitoring worker queue lengths and applying some simple hysteresis, are described in [23].

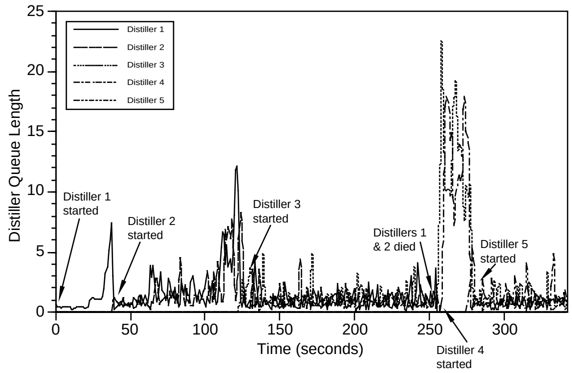

Figure 5: Worker queue lengths observed over time as the load presented to the system fluctuates, and as workers are manually brought down.

Figure 5 shows the variation in worker queue lengths over time. The system was bootstrapped with one front end and the manager, and a single demand-spawned worker. Continuously increasing

[3]The PostScript-to-richtext worker described in section 2 has not yet been added to TranSend.

the load caused the manager to spawn a second and later a third worker. We then manually killed the first two workers; the sudden load increase on the remaining worker caused the manager to spawn one and later another new worker, to stabilize the queue lengths.

3.3.2 Scalability

To demonstrate the scalability of the system, we performed the following experiment:

1. We began with a minimal instance of the system: one front end, one worker, the manager, and a fixed number of cache partitions. (Since for these experiments we repeatedly requested the same subset of images, the cache was effectively not tested.)

2. We increased the offered load until some system component saturated (e.g., worker queues growing too long, front ends no longer accepting additional connections, etc.).

3. We then added more resources to the system to eliminate this saturation (in many cases the system does this automatically, as when it recruits overflow nodes to run more workers), and we recorded the amount of resources added as a function of the increase in offered load, measured in requests per second.

4. We continued until the saturated resource could not be replenished (i.e., we ran out of hardware), or until adding more of the saturated resource no longer resulted in a linear or close-to-linear improvement in performance.

Req./ Second	**# FE's**	**# Wkrs.**	**Element that saturated**
0-24	1	1	workers
25-47	1	2	workers
48-72	1	3	workers
73-87	1	4	FE Ethernet
88-91	2	4	workers
92-112	2	5	workers
113-135	2	6	workers + FE Ethernet
136-159	3	7	workers

Table 5: Results of the scalability experiment. "FE" refers to front end.

Table 5 presents the results of this experiment. At 24 requests per second, as the offered load exceeded the capacity of the single available worker, the manager automatically spawned one additional worker, and then subsequent workers as necessary. (In addition to using faster hardware, the performance-engineering of the cluster-based server has caused a large reduction in the amortized cost of distillation for a typical image, compared to the values suggested by Figure 2.) At 87 requests per second, the Ethernet segment leading into the front end saturated, requiring a new front end to be spawned. We were unable to test the system at rates higher than 159 requests per second, as all of our cluster's machines were hosting workers, front ends, or playback engines. We did observe nearly perfectly linear growth of the system over the scaled range: a worker can handle approximately 23 requests per second, and a 100 Mb/s Ethernet segment into a front-end can handle approximately 70 requests per second. We were unable to saturate the front end, the cache partitions, or fully saturate the interior interconnect during this experiment. We draw two conclusions from this result:

- Even with a commodity 100 Mb/s interconnect, linear scaling is limited primarily by bandwidth into the system rather than bandwidth inside the system.

- Although we originally deployed TranSend on four SPARC 10's, a single Ultra-1 class machine would suffice to serve the entire dialup IP population of UC Berkeley (25,000 users officially, over 8000 of whom surfed during the trace).

4 Other TACC Applications

We now discuss several examples of new services in various stages of deployment, showing how each exploits the TACC model and discussing some of our experiences with the applications. Rather than providing detailed measurements as we did for TranSend in the previous section, the present goal is to demonstrate the flexibility of the TACC framework in accommodating an interesting range of applications, while providing consistent guidelines for approaching application partitioning decisions.

We restrict our discussion here to services that can be implemented using the proxy model, i.e., transparent interposition of computation between clients and servers. (Some of our services do not communicate via HTTP but are conceptually similar.) Also, although we have developed a wider range of applications using the TACC model as part of a graduate seminar [18], we concentrate on those applications that enable adaptation to network and client variation. These services share the following common characteristics, which make them amenable to implementation on our cluster-based framework:

- Compute-intensive transformation or aggregation
- Computation is parallelizable with granularity of a few CPU seconds
- Substantial value added by mass customization

4.1 TranSend as a TACC Application

TranSend is one of the simplest TACC applications we have produced. The dispatch rules simply match the MIME type of the object returned from the origin server to the list of known workers, which (as in all TACC applications) can be updated dynamically. In particular, TranSend does not exploit TACC's ability to compose workers by chaining them into a "pipeline" or having one worker call others as coroutines.

Transformed objects are stored in the cache with "fat URL's" that encode a subset of the transformation parameters, saving the work of re-transforming an original should another user ask for the same degraded version later. Each user can select a desired level of aggressiveness for the lossy compression and choose between HTML and Java-based interfaces for modifying their preferences.

The main difference between TranSend and commercial products based on its ideas (such as Intel's recently-announced QuickWeb [11]) is extensibility: adding support for new datatypes to TranSend is as simple as adding a new worker, and composing workers is as simple as modifying the dispatch rules (or modifying existing workers to hint to the TACC server that they should fall through to new workers). In fact, we have generalized TranSend into a "lazy fixations" system [20] in which users could select from among a variety of available formats for viewing an object; this was implemented by a "graph search" worker that treated all the transformation workers as edges in a directed graph and performed a shortest-paths search to determine what sequence of workers should be run to satisfy a particular request.

One of the goals of TACC is to exploit modularity and composition to make new services easy to prototype by reusing existing building blocks. TranSend's HTML and JPEG workers consist almost entirely of off-the-shelf code, and each took an afternoon to write. A pair of anecdotes illustrates the flexibility of the TACC API's in constructing responsive services. Our original HTML parser was a fast C-language implementation from the W3C. Debugging the pathological cases for this parser was spread out over a period of days—since our prototype TACC server masks transient faults by bypassing original content "around" the faulting worker, we could only deduce the existence of bugs by noticing (on the Graphical Monitor display) that the HTML worker had been restarted several times over a period of hours, although the service as a whole was continuously available.

We later wrote a much slower but more robust parser in Perl to handle proprietary HTML extensions such as inline JavaScript. All HTML pages are initially passed to the slower Perl parser, but if it believes (based on page length and tag density) that processing the page will introduce a delay longer than one or two seconds, it immediately throws an exception and indicates that the C parser should take over. Because the majority of long pages tend to be papers published in HTML rather than complex pages with weird tags, this scheme exploits TACC composition and dispatch to handle common cases well while keeping HTML processing latency barely noticeable.

4.2 Top Gun Wingman

Top Gun Wingman is the only graphical Web browser available for the 3Com PalmPilot, a typical "thin client" device. Based on file downloads, we estimate that 8000 to 10,000 users are using the client software and UC Berkeley's experimental cluster; ProxiNet, Inc. has since commercialized the program and deployed a production cluster to serve additional users. Figure 6 shows a screenshot of the browser.

Previous attempts to provide graphical Web browsing on such small devices have foundered on the severe limitations imposed by small screens, limited computing capability, and austere programming environments, and virtually all have fallen back to simple text-only browsing. Our adaptation approach, combined with the composable-workers model provided by TACC, allows us to approach this problem from a different perspective. The core of Top Gun Wingman consists of three TACC workers: HTML layout, image conversion, and intermediate-form layout to device-specific data format conversion. These three workers address the three areas of variation introduced in section 2:

- Hardware and software adaptation: We have

Figure 6: Screenshot of the Top Gun Wingman browser. This screenshot is taken from the "xcopilot" hardware-level Pilot simulator [15].

built TACC workers that output simplified binary markup and scaled-down images ready to be "spoon fed" to a thin-client device, given knowledge of the client's screen dimensions, image format, and font metrics. This greatly simplifies client-side code since no HTML parsing, layout, or image processing is necessary, and as a side benefit, the smaller and more efficient data representation reduces transmission time to the client. The image worker delivers 2-bit-per-pixel images, since that is what the PalmPilot hardware supports, and the HTML parsing and layout worker ensures that no page description larger than about 32KB is delivered to the client, since that is the approximate heap space limit imposed by the PalmPilot's programming environment. We have also added three "software upgrades" at the proxy since Wingman was first deployed: a worker that delivers data in AportisDoc [4] format (a popular PalmPilot e-book format), a worker that extracts and displays the contents of software archives for download directly to the PalmPilot, and an improved image-processing module contributed by a senior graphics hacker. In terms of code footprint, Wingman weighs in at 40KB of code (compared with 74KB and 109KB for HandWeb and Palmscape 5.0 respectively, neither of which currently support image viewing).

- Network protocol adaptation: In addition to delivering data in a more compact format and exploiting datatype-specific distillation, we have replaced HTTP with a simpler, datagram-oriented protocol based on Application Level Framing [10]. The combined effect of these optimizations makes Wingman two to four times faster than a desktop browser loading the same Web pages over the same bandwidth, and Wingman's performance on text-only pages often exceeds that of HTML/HTTP compliant browsers on the same platform, especially on slow ($<$ 56 Kb/s) links.

4.3 Top Gun Mediaboard

TopGun Mediaboard is an electronic shared whiteboard application for the PalmPilot. This is a derivation of the the desktop *mediaboard* application, which uses SRM (Scalable Reliable Multicast) as the underlying communication protocol. A reliable multicast proxy (RMX) TACC worker participates in the SRM session on behalf of the PDA clients, performing four main types of client adaptation:

- Transport protocol conversion: The PalmPilot's network stack does not support IP multicast. The RMX converts the SRM data into a unicast TCP stream that the client can handle.

- Application protocol adaptation: To keep the client implementation simple, all the complexities of the mediaboard command protocol are handled by the RMX. The protocol adapter transforms the entire sequence of mediaboard commands into a "pseudo-canvas" by executing each command and storing its result in the canvas, transmitting only a sequence of simple draw-ops to the client. The protocol and data format for transmitting the draw-ops is a direct extension of the Top Gun Wingman datagram protocol.

- On-demand distillation: The RMX converts specific data objects according to the client's needs. For example, it transforms the GIF and JPEG images that may be placed on the mediaboard into simpler image representations that the PalmPilot can understand, using the same worker that is part of Wingman. The client application can refine (zoom in on) specific portions of the canvas.

- Intelligent Rate Limiting: Since the proxy has complete knowledge of the client's state, the

RMX can perform intelligent forwarding of data from the mediaboard session to the client. By eliminating redundant draw-ops (for example, *create* followed by *delete* on the same object) before sending data to the client, the RMX reduces the number of bytes that must be sent over the low-bandwidth link. Moreover, although a whiteboard session can consist of a number of distinct pages, the RMX forwards only the data associated with the page currently being viewed on the client.

Top Gun Mediaboard is in prealpha use at UC Berkeley, and performs satisfactorily even over slow links such as the Metricom Ricochet wireless packet radio modem [31].

4.4 Charon: Indirect Authentication for Thin Clients

Although not yet rewritten as a TACC application, Charon [21] illustrates a similar use of adaptation by proxy, for performing indirect authentication. In particular, Charon mediates between thin clients and a Kerberos [38] infrastructure. Charon is necessary because, as we describe in [21], the computing resources required for a direct port of Kerberos to thin clients are forbidding. With Charon, Kerberos can be used both to authenticate clients to the proxy service, and to authenticate the proxied clients to Kerberized servers. Charon relieves the client of a significant amount of Kerberos protocol processing, while limiting the amount of trust that must be placed in the proxy; in particular, if the proxy is compromised, existing user sessions may be hijacked but no new sessions can be initiated, since new sessions require cooperation between the client and proxy. Our Charon prototype client for the Sony MagicLink [14], a once-popular PDA, had a client footprint of only 45KB, including stack and heap usage.

5 Related Work

At the network level, various approaches have been used to shield clients from the effects of poor (especially wireless) networks [5]. At the application level, data transformation by proxy interposition has become particularly popular for HTTP, whose proxy mechanism was originally intended for users behind security firewalls. The mechanism has been harnessed for anonymization [8], Kanji transcoding [36, 41], application-specific stream transformation [7], and personalized "associates" for Web browsing [6, 37]. Some projects provide an integrated solution with both network-level and application-level mechanisms [30, 17, 43], though none propose a uniform application-development model analogous to TACC.

Rover [16], Coda [29], and Wit [40] differ in their respective approaches to partitioning applications between a thin or poorly-connected client and more powerful server. In particular, Rover and Coda provide explicit support for disconnected operation, unlike our TACC work. We find that Rover's application-specific, toolkit-based approach is a particularly good complement to our own; although the TACC model provides a reasonable set of guidelines for thinking about partitioning (leave the client to do what it does well, and move as much as possible of the remaining functionality to the back end), we are working on integrating Rover into TACC to provide a rich abstraction for dealing with disconnection in TACC applications.

SmartClients [42] and SWEB++ [3] have exploited the extensibility of client browsers via Java and JavaScript to enhance scalability of network-based services by dividing labor between the client and server. We note that our system does not preclude, and in fact benefits from, exploiting intelligence and computational resources at the client; we discuss various approaches we have tried in [24].

6 Lessons and Conclusions

We proposed three design principles for adapting to network and client variation and delivering a meaningful Internet experience to impoverished clients: datatype-specific distillation and refinement, adaptation on demand, and moving complexity into the infrastructure. We also offered a high-level description of the TACC programming model (transformation, aggregation, caching, customization) that we have evolved for building adaptive applications, and presented measurements of our scalable, highly-available, cluster-based TACC server architecture, focusing on the TranSend web accelerator application. Finally, we described other applications we have built that are in daily use, including some that push the limits of client adaptation (such as Top Gun Wingman and Top Gun Mediaboard). In this section we try to draw some lessons from what we have learned from building these and similar applications and experimenting with our framework.

Aggressively pushing the adaptation-by-proxy model to its limits, as we have tried to do with Top Gun Wingman and Top Gun Mediaboard, has helped us validate the proxy-interposition approach for serving thin clients. Our variation on the theme of application partitioning has been to split the application between the client and the proxy, rather than between the client and the server. This has allowed our clients to access existing content with no server modifications. Our guideline for partitioning applications has been to allow the client to perform those tasks it does well in native code, and move as much as possible of the remaining work to the proxy. For example, since most thin clients support some form of toolkit for building graphical interfaces, sending HTML markup is too cumbersome for the client, but sending screen-sized bitmaps is unnecessarily cumbersome for the proxy.

A frequent objection raised against our partitioning approach is that it requires that the proxy service be available at all times, which is more difficult than simply maintaining the reliability of a bank of modems and routers. This observation motivated our work on the cluster-based scalable and highly-available server platform described in section 3, and in fact the TranSend and Wingman proxy services have been running for several months at UC Berkeley with high stability, except for a two-week period in February 1998 when the cluster was affected by an OS upgrade. Other than one part-time undergraduate assistant, the cluster manages itself, yet thousands of users have come to rely on its stability for using Top Gun Wingman, validating the efficacy of our cluster platform. This observation, combined with the current trends toward massive cluster-based applications such as HotBot [13], suggests to us that the adaptive proxy style of adaptation will be of major importance in serving convergent "smart phone"-like devices.

7 Acknowledgments

This project has benefited from the detailed and perceptive comments of countless anonymous reviewers, users, and collaborators. Ken Lutz and Eric Fraser configured and administered the test network on which the TranSend scaling experiments were performed. Cliff Frost of the UC Berkeley Data Communications and Networks Services group allowed us to collect traces on the Berkeley dialup IP network and has worked with us to deploy and promote TranSend within UC Berkeley. Undergraduate researchers Anthony Polito, Benjamin Ling, Andrew Huang, David Lee, and Tim Kimball helped implement various parts of TranSend and Top Gun Wingman. Ian Goldberg and David Wagner helped us debug TranSend, especially through their implementation of the Anonymous Rewebber [26], and Ian implemented major parts of the client side of Top Gun Wingman, especially the 2-bit-per-pixel hacks. Paul Haeberli of Silicon Graphics contributed image processing code for Top Gun Wingman. Murray Mazer at the Open Group Research Institute has provided much useful insight on the structure of Internet applications and future extensions of this work. We also thank the patient students of UCB Computer Science 294–6, *Internet Services*, Fall 97, for being the first real outside developers on our TACC platform and greatly improving the quality of the software and documentation.

We have received much valuable feedback from our UC Berkeley colleagues, especially David Culler, Eric Anderson, Trevor Pering, Hari Balakrishnan, Mark Stemm, and Randy Katz. This research is supported by DARPA contracts #DAAB07-95-C-D154 and #J-FBI-93-153, the California MICRO program, the UC Berkeley Chancellor's Opportunity Fellowship, the NSERC PGS-A fellowship, Hughes Aircraft Corp., and Metricom Inc.

References

[1] Elan Amir, Steve McCanne, and Hui Zhang. An application level video gateway. In *Proceedings ACM Multimedia 1995*, 1995.

[2] Thomas E. Anderson, David E. Culler, and David Patterson. A case for now (networks of workstations). *IEEE Micro*, 12(1):54–64, Feb 1995.

[3] D. Andresen, T. Yang, O. Egecioglu, O. H. Ibarra, and T. R. Smith. Scalability issues for high performance digital libraries on the world wide web. In *Proceedings of IEEE ADL '96, Forum on Research and Technology Advances in Digital Libraries*, Washington D.C., May 1996.

[4] Aportis Inc. AportisDoc Overview, 1998. `http://www.aportis.com/products/AportisDoc/benefits.html`.

[5] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving tcp performance over wireless links. In *Proceedings of the 1996 ACM SIGCOMM Conference*, Stanford, CA, USA, August 1996.

[6] Rob Barrett, Paul P. Maglio, and Daniel C. Kellem. How To Personalize the Web. In *Conference on Human Factors in Computing Systems (CHI 95)*, Denver, CO, May 1995. WBI, developed at IBM

Almaden; see `http://www.raleigh.ibm.com/wbi/wbisoft.htm`.

[7] Charles Brooks, Murray S. Mazer, Scott Meeks, and Jim Miller. Application-Specific Proxy Servers as HTTP Stream Transducers. In *Proceedings of the Fourth International World Wide Web Conference*, Dec 1995.

[8] C2net. Web anonymizer.

[9] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. In *Proceedings of the 1996 Usenix Annual Technical Conference*, pages 153–163, January 1996.

[10] D.D. Clark and D.L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *Computer Communication Review*, 20(4):200–208, Sep 1990.

[11] Intel Corp. QuickWeb Web Accelerator.

[12] Nokia Corp. and Geoworks Inc. Nokia 9000 Communicator. `http://www.geoworks.com/htmpages/9000.htm`.

[13] Inktomi Corporation. The hotbot search engine.

[14] Sony Corporation. The Sony MagicLink PDA. `http://www.sel.sony.com/SEL/Magic/`.

[15] Ivan Curtis. xcopilot Pilot simulator, 1998.

[16] Anthony D. Joseph et al. Rover: A toolkit for mobile information access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain Resort, CO, USA, Dec 1995.

[17] WAP Forum. Wireless application protocol (WAP) forum. http://www.wapforum.org.

[18] Armando Fox and Eric A. Brewer. CS 294-6: Internet services, class proceedings, fall 1997. http://www.cs.berkeley.edu/∼fox/cs294.

[19] Armando Fox and Eric A. Brewer. Reducing WWW Latency and Bandwidth Requirements via Real-Time Distillation. In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. World Wide Web Consortium.

[20] Armando Fox and Steven D. Gribble. DOLF: Digital objects with lazy fixations. Unpublished manuscript: CS 294-5 Digital Libraries Seminar, Spring 1996.

[21] Armando Fox and Steven D. Gribble. Security On the Move: Indirect Authentication Using Kerberos. In *Proc. Second International Conference on Wireless Networking and Mobile Computing (MobiCom '96)*, Rye, NY, November 1996.

[22] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric Brewer. TranSend Web Accelerator Proxy. Free service deployed by UC Berkeley. See `http://transend.cs.berkeley.edu`, 1997.

[23] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-Based Scalable Network Services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, St.-Malo, France, October 1997.

[24] Armando Fox, Steven D. Gribble, Yatin Chawathe, Anthony Polito, Benjamin Ling, Andrew C. Huang, and Eric A. Brewer. Orthogonal Extensions to the WWW User Interface Using Client-Side Technologies. In *User Interface Software and Technology (UIST) 97*, Banff, Canada, October 1997.

[25] Graphics interchange format version 89a (GIF). CompuServe Incorporated, Columbus, Ohio, July 1990.

[26] Ian Goldberg and David Wagner. Taz servers and the rewebber network: Enabling anonymous publishing on the world wide web. Unpublished manuscript available at http://www.cs.berkeley.edu/∼daw/cs268/, May 1997.

[27] Steven D. Gribble and Eric A. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Monterey, California, USA, December 1997.

[28] Tom R. Halfhill. Inside the web pc. *Byte Magazine*, pages 44–56, March 1996.

[29] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10:3–25, February 1992.

[30] Liljeberg, M., et al. Enhanced Services for World Wide Web in Mobile WAN Environment. Technical Report C-1996-28, University of Helsinki CS Department, April 1996.

[31] Metricom Corp. Ricochet Wireless Modem, 1998. `http://www.ricochet.net`.

[32] Myricom. Myrinet: A Gigabit Per Second Local Area Network. In *IEEE Micro*, February 1995.

[33] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[34] DEC SRC Personal Communications, Paul MacJones. Postscipt to text converter.

[35] Jef Poskanzer. Netpbm release 7. ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM, 1993.

[36] Y. Sato. DeleGate Server, March 1994. `http://wall.etl.go.jp/delegate/`.

[37] M.A. Schickler, M.S. Mazer, and C. Brooks. Pan-browser support for annotations and other meta-information on the world wide web. In *Proc. Fifth International World Wide Web Conference (WWW-5)*, May 1996.

[38] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings USENIX Winter Conference 1988*, pages 191–202, Dallas, Texas, USA, February 1988.

[39] Mark Stemm and Randy H. Katz. Vertical handoffs in wireless overlay network. *ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet*, Fall 1997.

[40] Terri Watson. Application Design for Wireless Computing. In *Mobile Computing Systems and Applications Workshop*, August 1994.

[41] Ka-Ping Yee. Shoduoka Mediator Service, 1995. `http://www.shoduoka.com`.

[42] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable services. In *Proc. Winter 1997 USENIX Technical Conference*, January 1997.

[43] Bruce Zenel. A Proxy Based Filtering Mechanism for the Mobile Environment. Thesis Proposal, Mar 1996.