To appear in the Proceedings of the 16th ACM Symposium on Operating System Principles

Agile Application-Aware Adaptation for Mobility

Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, Kevin R. Walker
School of Computer Science
Carnegie Mellon University

Abstract

In this paper we show that *application-aware adaptation*, a collaborative partnership between the operating system and applications, offers the most general and effective approach to mobile information access. We describe the design of *Odyssey*, a prototype implementing this approach, and show how it supports concurrent execution of diverse mobile applications. We identify *agility* as a key attribute of adaptive systems, and describe how to quantify and measure it. We present the results of our evaluation of Odyssey, indicating performance improvements up to a factor of 5 on a benchmark of three applications concurrently using remote services over a network with highly variable bandwidth.

1 Introduction

Adaptation is the key to mobility. Only through alertness and prompt reactions can a mobile client offer acceptable service in spite of the many problems that plague its existence. These include unpredictable variation in network quality, wide disparity in the availability of remote services, limitations on local resources imposed by weight and size constraints, concern for battery power consumption, and lowered trust and robustness resulting from exposure and motion [5, 15, 31].

Once the need for adaptation is recognized, many questions follow. What form should such adaptation take? Which system components should bear responsibility for adaptation? How does one characterize the adaptive capability of a mobile client? How does one compare alternative designs from the perspective of adaptation?

We present our answers to these and related questions in this paper. We describe the design and implementation of a software platform called *Odyssey*, and show how it provides effective support for concurrent execution of diverse mobile applications. We identify *agility* as a key attribute of adaptive systems, and describe how to quantify and measure it. Finally, we present the results of our evaluation of the Odyssey prototype to confirm the benefits of our approach. These results indicate performance improvements up to a factor of 5 on a benchmark of three applications concurrently using remote services over a network with highly variable bandwidth.

This research was supported by the Air Force Materiel Command (AFMC) under DARPA contract numbers F19628-93-C-0193 and F19628-96-C-0061. Additional support was provided by AT&T, Hughes, IBM, and Intel. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, DARPA, AT&T, Hughes, IBM, Intel, Carnegie Mellon University, or the U.S. Government.

2 Design Rationale

Odyssey is a set of extensions to the NetBSD operating system to support adaptation for a broad range of *mobile information access* applications. These applications execute on mobile clients but read or update remote data on servers. Our goal in building Odyssey is to enable mobile scenarios such as the following one.

2.1 Motivation

Consider a hypothetical scenario in which a tourist with a wearable computer running Odyssey is walking in an urban setting. A wireless overlay network [16] provides the computer with a variety of connection alternatives, which differ in bandwidth, coverage, cost, and reliability. The higher-bandwidth alternatives are more sensitive to fading and signal loss as the user moves in and out of the radio shadows of buildings.

As he walks, the user interacts with his computer through spoken commands; he receives output through a head-mounted display or synthesized speech. The speech software exploits remote compute servers when connected, but is capable of degraded interactions using a tiny vocabulary when disconnected. One application provides a video narration of local history, with content delivered from a remote server. Another application is a Web browser that can respond to queries about the local environment.

Odyssey monitors resources such as bandwidth, CPU cycles, and battery power, and interacts with each application to best exploit them. For example, when high-bandwidth connectivity is lost due to a radio shadow, Odyssey detects the change and notifies interested applications. The video application responds by skipping frames, thus displaying fewer frames per minute, while the Web application responds by displaying degraded versions of large images. When the user emerges from the radio shadow, Odyssey detects a substantial improvement in bandwidth and notifies applications. They then revert to their original behaviors.

Although the user is aware of changing application behavior during his walk, he does not have to initiate adaptation or be involved in its details. Rather, he can delegate these decisions to Odyssey, confident that reasonable tradeoffs will be made.

While mobile scenarios such as this motivated the design of Odyssey, its benefits may be valuable in a broader context. For example, large bandwidth variations can arise in wide-area networks with fluctuating loads. The adaptation supported by Odyssey can be valuable in coping with such variation.

2.2 Fidelity

As the example in the previous section illustrates, the data accessed by an Odyssey application may be stored in one or more general-purpose repositories such as file servers, SQL servers, or Web servers. Alternatively, it may be stored in more specialized repositories such as video libraries, query-by-image-content databases, or back ends of geographical information systems.

The constraints of mobility complicate data access from such servers. Ideally, a data item available on a mobile client should be indistinguishable from that available to the accessing application if it were to be executed on the server storing that item. But this correspondence may be difficult to preserve as resources become scarce; some form of degradation may be inevitable. We define *fidelity* as the degree to which data presented at a client matches the reference copy at the server.

Fidelity has many dimensions. One well-known, universal dimension is *consistency*. Systems such as Coda [18, 32], Ficus [28] and Bayou [38] expose potentially stale data to applications when network connectivity is poor or nonexistent. Other dimensions of fidelity depend on the type of data in question. For example, video data has at least two additional dimensions: frame rate and image quality of individual frames. Spatial data, such as topographical maps, has dimensions of minimum feature size or resolution. For telemetry data, appropriate dimensions include sampling rate and timeliness. The dimensions of fidelity are natural axes of adaptation for mobility. However, the adaptation cannot be solely determined by the type of data; it also depends on the application. Different applications using the same data may make different tradeoffs among dimensions of fidelity.

A key goal of Odyssey is to provide a framework within which diverse notions of fidelity can easily be incorporated. Our focus on mobile information access allows us to bound this diversity to manageable proportions. Specifically excluded from our purview are applications that involve stringent real-time constraints, such as video conferencing and multi-client interactive games.

Supporting multiple levels of fidelity complicates the task of evaluation. Since adaptive applications trade fidelity of data for performance, focusing solely on the latter can result in a misleading evaluation. For example, by forcing applications to operate at their lowest fidelity levels at all times, a system could ensure better performance than a competing system that strives to support higher fidelity levels when possible. Yet, this degenerate approach violates our intuitive notion of what constitutes a good system. Hence, the evaluation of an adaptive system must take into account both fidelity and performance.

2.3 Concurrency

The ability to execute multiple independent applications concurrently on a mobile client is vital. Although this ability is taken for granted on desktop operating systems, there continues to be skepticism about its value in mobile clients. This skepticism is fueled by the popularity of devices such as PDAs [1] and pocket organizers [39], which execute only one application at a time.

While such specialized mobile devices fill an important niche, we are convinced that many mobile users will find it valuable to run background applications in addition to the foreground application that dominates their attention. For example, an information filtering application may run in the background monitoring data such as stock prices or enemy movements, and alert the user as appropriate. As another example, an application used in emergency response situations may monitor physical location and motion, and prefetch damage-assessment information for the areas to be traversed shortly.

To effectively support concurrent applications, one must control their use of limited resources. In other words, there needs to be centralized monitoring and coordinated resource management on a mobile client. Operating systems, which have historically performed this role for CPU cycles and memory, must now manage a broader range of resources such as network bandwidth, disk cache space and battery power.

The need to coordinate resource management across applications mutes the effectiveness of many current approaches to mobile computing. For example, commercial applications such as Eudora [29] provide vertically integrated support for mobility, where each application assumes that it has full use of available network bandwidth. Even a more sophisticated toolkit approach such as Rover [13] only pays minimal attention to resource coordination.

2.4 Agility

Sound adaptation decisions require accurate and timely knowledge of resource availability. Ideally, a mobile client should always have perfect knowledge of current resource levels. In other words, there should be no time lag between a change in resource availability and its detection. Further, if this change is sufficient to warrant modification of client behavior, that too should be accomplished without delay.

Of course, no physical system can meet this ideal. The best we can hope for is to build close approximations through good design and engineering. Thus, a key property of an adaptive system is the speed and accuracy with which it detects and responds to changes in resource availability. We call this property the *agility* of the system. When changes are large and erratic, only a highly agile system can function effectively. In more stable environments, less agile systems may suffice. Agility is thus the property of a mobile system that determines the most turbulent environment in which it can function acceptably.

Agility is a complex property with many components. One source of complexity is differing sensitivity to different resources. For example, a system may be much more sensitive to changes in network bandwidth than to changes in battery power level. Another source of complexity is differing origins of changes in resource availability. The change may be caused by variation in the *supply* of a resource due to mobility, or by changed *demand* for it by concurrent applications. Since different mechanisms may be involved in detecting these two different types of changes, it may be necessary to distinguish these components of agility.

2.5 Minimalism

Rather than using a clean-sheet approach to designing Odyssey, we decided to extend an existing system. We chose NetBSD, a variant of the 4.4 BSD Unix operating system [23], as the starting point. NetBSD source code is publicly available without encumbrance, thus allowing free distribution of derivatives. The popularity of NetBSD also offers the possibility of attracting a substantial Odyssey user community.

We avoided changes to the NetBSD API and internal structure unless essential, and made the few necessary changes consistent with NetBSD idiom. Thus, Odyssey should be viewed as an exercise in minimalism: it represents the smallest set of interface and code extensions we believe necessary for agile adaptation in mobile environments.

3 Application-Aware Adaptation

3.1 Model of Adaptation

Odyssey's approach to adaptation is best characterized as *application-aware adaptation*. The essence of this model is a collaborative partnership between the system and individual applications. The system monitors resource levels, notifies applications of relevant changes, and enforces resource allocation decisions. Each application independently decides how best to adapt when notified.

This division of responsibility directly addresses the issues of application diversity and concurrency. Diversity is accommodated by allowing applications to determine the mapping of resource levels to fidelity levels. Concurrency is supported by allowing the system to retain control of resource monitoring and arbitration. The challenge is to design a system that can support this separation of concerns without compromising agility.

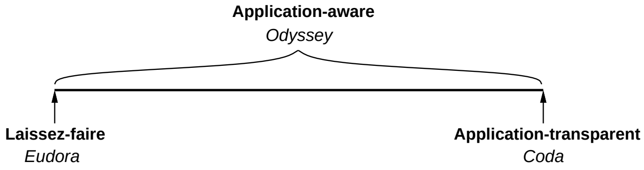


Figure 1: Models of Adaptation

Figure 1 places application-aware adaptation in context, spanning the range between two extremes. At one extreme, adaptation is entirely the responsibility of individual applications. This *laissez-faire* approach, used by commercial software packages such as Eudora, avoids the need for system support. But, as discussed in Section 2.3, it fails to address the issue of application concurrency. At the other extreme, *application-transparent adaptation*, the system bears full responsibility for both adaptation and resource management. This approach, exemplified by Coda, is especially attractive for legacy applications because they can run unmodified. Application concurrency is well supported, but application diversity is not, since control of fidelity is entirely in the hands of the system.

3.2 Realizing the Model

The obvious approach to implementing application-aware adaptation would be to directly reflect its separation of concerns in the Odyssey architecture. In such an architecture, system code would treat data generically; individual applications would be entirely responsible for differential handling of data types.

Unfortunately, the wide disparity in the physical and logical properties of various data types requires that some form of type-awareness be incorporated into the system for efficient resource usage. For example, the size distribution and consistency requirements of data from an NFS server differ substantially from those of relational database records. Image data may be highly compressible using one algorithm but not another. Video data can be efficiently shipped using a streaming protocol that drops rather than retransmits lost data; in contrast, only reliable transmissions are acceptable for file or database updates. It is impossible to optimize for such differences without some system-level knowledge of type.

These considerations lead to a more sophisticated architecture in which Odyssey has two responsibilities. It must be aware of shared access to remote data by concurrent applications so that it can properly manage resources. At the same time, it must have type-specific knowledge to allow effective resource management actions. Such knowledge is necessary, for example, to estimate the relative costs and benefits of compressing a cached item versus flushing it and refetching it later.

Odyssey incorporates type-awareness via specialized code components called *wardens*. A warden encapsulates the system-level support at a client necessary to effectively manage a data type. To fully support a new data type, an appropriate warden has to be written and incorporated into Odyssey at each client. The wardens are subordinate to a type-independent component called the *viceroy*, which is responsible for centralized resource management.

The collaborative relationship envisioned in application-aware adaptation is thus realized in two parts. The first, between the viceroy and its wardens, is *data-centric*: it defines the fidelity levels for each data type and factors them into resource management. The second, between applications and Odyssey, is *action-centric*: it provides applications with control over the selection of fidelity levels supported by the wardens.

4 Design and Implementation

An implementation of Odyssey must enable an application to

- operate on Odyssey objects,
- express resource expectations,
- be notified when expectations are no longer met, and
- respond by changing fidelity.

The Odyssey mechanisms supporting each of these requirements are described in the following sections.

4.1 Operating on Odyssey Objects

Consistent with our goal of minimalism, we have built upon NetBSD file system calls rather than defining a completely new interface. Thus, Odyssey is integrated into NetBSD as a new VFS file system [19]. In addition, we have found it necessary to add a few new system calls.

As shown in Figure 2, we have implemented the viceroy and wardens in user space rather than in the kernel. Operations on Odyssey objects are redirected to the viceroy by a small in-kernel *interceptor* module. All other system calls are handled directly by NetBSD.

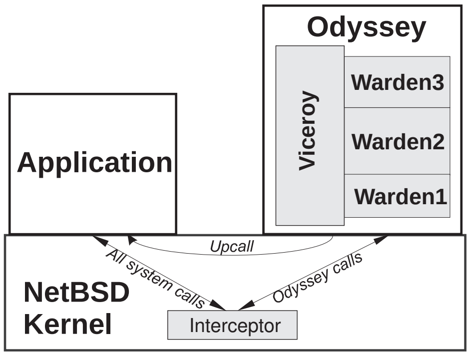


Figure 2: Odyssey Client Architecture

Wardens are statically linked with the viceroy, and the ensemble executes in a single address space with user-level threads. Communication between the viceroy and wardens is through procedure calls and shared data structures. The wardens are entirely responsible for communicating with servers and caching data from them when appropriate — applications never contact servers directly.

Our implementation of Odyssey outside the kernel is consistent with the philosophy of modern operating system designs such as Exokernel [6] and SPIN [3]. Since extensibility is the motivation for such systems, their use in the context of Odyssey should result in improved agility and easier installation of new wardens.

Integrating Odyssey with the file system yields several key advantages. It gives us a well-understood framework for integration as well as useful infrastructure to simplify implementation. Since file operations are performed on Odyssey objects by the appropriate warden, their semantics can be customized to provide a modicum of type-awareness without new system calls.

At the same time, our approach imposes certain limitations. Some data types do not naturally fit either the naming or access methods provided by the file system. Further, there are no standard file operations corresponding to fidelity changes. For naming, we incorporate extensions similar in spirit to *virtual directories* [9]. For access methods and fidelity changes, we rely on a general-purpose mechanism described in Section 4.4.

As Odyssey matures, we may discover that further improvements in functionality or agility are impossible unless we move the viceroy and wardens into the kernel. Until there is compelling evidence of this, however, we plan to retain the current architecture.

`request(in` path, `in` resource-descriptor, `out` request-id)
`cancel(in` request-id)

(a) Resource Negotiation Operations

resource-id
lower bound
upper bound
name of upcall handler

(b) Resource Descriptor Fields

Network Bandwidth	bytes/second
Network Latency	microseconds
Disk Cache Space	kilobytes
CPU	SPECint95
Battery Power	minutes
Money	cents

(c) Generic Resources in Odyssey

`handler(in` request-id, `in` resource-id, `in` resource-level)

(d) Upcall Handler

`tsop(in` path, `in` opcode, `in` insize, `in` inbuf,
`inout` outsize, `out` outbuf)

(e) Type-Specific Operations

This figure shows Odyssey's extensions to the NetBSD programming interface. Note that the `request` and `tsop` calls have variants that identify Odyssey objects by file descriptors rather than pathnames.

Figure 3: Odyssey API

4.2 Expressing Resource Expectations

Applications communicate resource expectations to Odyssey using the `request` system call shown in Figure 3(a). The call takes a resource descriptor identifying a resource and specifying a *window of tolerance* on its availability. This call expresses the application's desire to be told if the availability of the resource strays outside the window.

If, at the time of the `request`, the availability of the resource is within the window of tolerance, the viceroy registers the request and returns a unique identifier for it. This identifier can be used by the viceroy in notifying the application that the resource has left the requested bounds, or by the application in a future `cancel` system call to discard the registered request.

If the resource is currently outside the bounds of the tolerance window, an error code and the current available resource level are returned. The application is then expected to try again, with a new window of tolerance corresponding to a new fidelity level.

The fields of a resource descriptor are shown in Figure 3(b). Each resource is named by a unique *resource identifier*. Figure 3(c) lists the generic resources we plan to manage in Odyssey. At present the prototype only manages the most critical resource in mobile computing: network bandwidth. The window of tolerance is indicated by lower and upper bounds. A resource descriptor also specifies the name of a procedure that will be called to notify the application that the resource has left the window.

4.3 Notifying Applications

When the viceroy discovers that the availability of a resource has strayed outside a registered window of tolerance, it generates an *upcall* to the corresponding application. The application adjusts its fidelity according to its individual policy. It then issues another `request` call to register a revised window of tolerance appropriate to the new fidelity.

An upcall handler is invoked with three parameters, as shown in Figure 3(d). The first parameter identifies the `request` operation on whose behalf the upcall is being delivered. The second parameter identifies the resource whose availability has changed, and the third parameter gives the new availability.

Upcalls closely resemble Unix signals, but offer improved functionality. Like signals, upcalls can be sent to one or more processes, can be blocked or ignored, and have similar inheritance semantics on process fork. Unlike signals, upcalls offer exactly-once, in-order semantics for each receiver of a particular upcall. Further, upcalls allow parameters to be passed to target processes and results to be returned.

4.4 Changing Fidelity

Requests for fidelity changes do not map well to the NetBSD file system interface. Further, many data types have natural access methods that are not well supported by the untyped byte stream model. To address these shortcomings, we have included a general-purpose escape mechanism called `tsop`, or type-specific operation, shown in Figure 3(e). The arguments to `tsop` specify an Odyssey object and the opcode of a type-specific operation to be performed on it. Input and output parameters are specified as unstructured memory buffers, in the spirit of the `ioctl` system call.

5 Example Applications

To explore Odyssey's ability to support application diversity, we modified three very different applications to run on it. The first two applications are drawn from the domain of mobile information access: a video player whose source code we have access to, and a Web browser whose source code is not publicly available. The third application, a speech recognizer, was chosen to explore the effectiveness of Odyssey outside its original target domain.

Each application requires a different strategy for integration into Odyssey, and each has distinct notions of fidelity. Collectively, these applications serve as a vehicle to explore the generality and performance characteristics of Odyssey.

5.1 Video Player

Our support for video is based on *xanim*, a public-domain software package that can generate video animation from data stored in various formats in a local file. As shown in Figure 4, we split its monolithic implementation into a client and server, and wrote a warden to satisfy client requests and fetch data from the server.

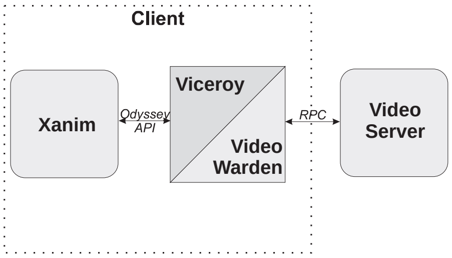


Figure 4: Video Player in Odyssey

Each movie is stored in multiple tracks at the server, one track per fidelity level. We have incorporated three levels of fidelity for Quicktime [11] video data: JPEG-compressed [42] color frames at qualities 99 and 50, and black-and-white frames. Storing all three tracks incurs only modest overhead, typically about 60% more than storing just the highest fidelity track.

The warden supports two tsops: to read a movie's meta-data, and to get a particular frame from a specified track. The warden performs read-ahead of frames to lower latency.

When the player opens a movie, it calculates the bandwidth requirements of each track from the movie meta-data. The player begins the movie at highest possible quality, and registers the corresponding window of tolerance with Odyssey. When it is notified of a significant change in bandwidth, the player determines a new fidelity level and switches to the corresponding track. If the player switches from a low fidelity track to a higher one, the warden discards the prefetched low-quality frames.

5.2 Web Browser

Netscape Navigator, or simply Netscape, is a widely-used tool for accessing the World-Wide Web. It is an especially interesting application for Odyssey because we do not have access to its source code. Since we cannot directly modify Netscape to take advantage of Odyssey, we exploit its *proxy* facility as shown in Figure 5.

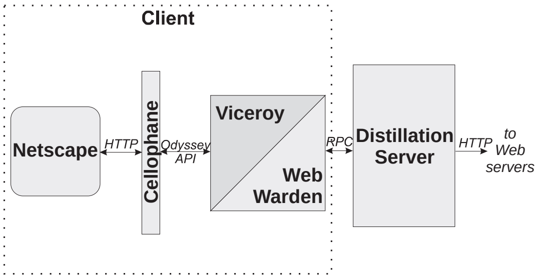


Figure 5: Extending Netscape for Odyssey

All of Netscape's requests are redirected to a client module called the *cellophane*. Together, Netscape and the cellophane constitute a single application from the viewpoint of Odyssey. The cellophane makes use of the Odyssey API and selects fidelity levels. Netscape passively benefits from the adaptation initiated by the cellophane.

The cellophane transforms HTTP requests from Netscape into file operations on Odyssey Web objects. The Web warden forwards these requests via the client's mobile network connection to a *distillation server*. The latter provides multiple levels of fidelity for images along the lines suggested by Fox et al [7]. Since images tend to be large and constitute a substantial fraction of HTTP traffic, focusing on them has a high payoff. At the highest fidelity, images are uncompressed. Progressively lower levels correspond to JPEG-compressed images of decreasing quality. The warden provides a tsop to set the fidelity level.

The distillation server fetches requested objects from the appropriate Web server, distills them to the requested fidelity level, and sends the results to the warden. The data is then passed to Netscape via the cellophane. These steps are completely transparent to both Netscape and the Web server; each perceives normal Web access.

Netscape exemplifies the unfortunate reality that source code is not publicly available for a growing number of important applications. Code interpositioning, the approach described above, represents only one way for such shrink-wrapped applications to benefit from Odyssey. Other possibilities include static binary rewriting of executables and dynamic modification of system calls.

5.3 Speech Recognizer

Speech recognition offers considerable potential as well as challenge for mobile computing. It is especially useful when mobile because it leaves the user's hands and eyes free for other activities such as driving [34]. However, the resource requirements for high-accuracy speech recognition are substantial, especially when mobile, since background noise is often high. Adding higher-level semantic processing, such as language translation, leads to even greater demands on computing resources. This combination of opportunity and challenge led us to implement speech recognition as an Odyssey application, even though it falls outside our primary domain of mobile information access.

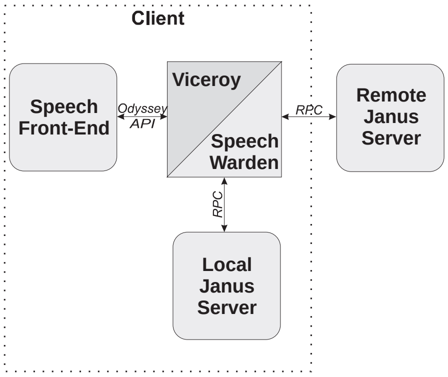


Figure 6: Speech Recognition in Odyssey

Figure 6 illustrates our support for speech recognition in Odyssey. The starting point for this implementation is a speech recognition system called *Janus* [41], whose source code is available to us. We have split this system into a client and server, and constructed a speech warden. The server accepts two forms of input: a raw utterance, or an utterance that has already been processed by the first of several phases of Janus. This pre-processing yields a compression ratio of approximately 5:1 at modest CPU cost.

The speech front-end captures a raw speech utterance and then writes it to an object in the Odyssey namespace. The warden, using the current bandwidth estimate, decides whether it is faster to perform the first pass of the recognition on the local, slower CPU, or to ship the larger, raw utterance to the server. In the extreme case of disconnection, the local Janus is capable of recognizing the utterance, but at a severe CPU and memory cost. When the utterance is recognized, the resulting text is made available to the front-end through a `read` operation. We are currently refining our implementation to support multiple levels of recognition fidelity.

6 Evaluation

Three central questions drove our evaluation of Odyssey:

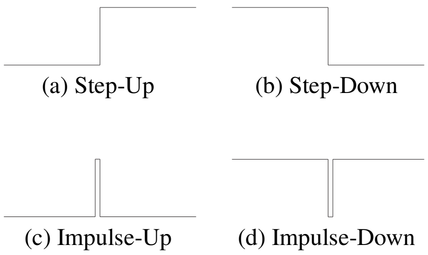
- How agile is Odyssey in the face of changing network bandwidth?
- How beneficial is it for applications to exploit the dynamic adaptation made possible by Odyssey?
- How important is centralized resource management for concurrent applications?

In posing these questions, two secondary questions come to light. First, how should the concept of agility be quantified? Second, what experimental methodology should we use to obtain agility metrics? We address these secondary questions first, in Section 6.1, and present our answers to the primary questions in Section 6.2.

6.1 Evaluation Strategy

6.1.1 Agility Metrics

Our approach to quantifying agility draws upon well-established principles for measuring dynamic response from the field of *control systems* [4, 30]. The accepted practice in that field is to characterize the adaptive ability of a system with respect to a particular output in terms of its responses to a set of input *reference waveforms*. Each reference waveform is conceptually simple, yet greatly stresses the adaptive ability of the system by varying the input in some sharp and substantial manner.



The Step-Up and Step-Down waveforms are each 60 seconds long, with a single, abrupt transition at the midpoint. The Impulse-Up and Impulse-Down waveforms are approximations to the ideal impulse, which is a spike of infinitesimal width and infinite height. We approximate the ideal with two-second wide excursions in the middle of a 60-second period. Our choice of parameters for these waveforms is based on our estimate of the basic network time constants of typical distributed systems today: 30 seconds should be long enough for a system to reach steady state after a bandwidth perturbation; a 2 second perturbation is large enough to be detectable by a sensitive system, yet small enough to be missed by an insensitive one.

Figure 7: Reference Waveforms for Agility Experiments

Figure 7 illustrates the reference waveforms used in our evaluation. Although these waveforms are idealized, approximations to them can occur in practice. The step waveforms can arise in overlay networks, where a mobile client may seamlessly switch between different network interfaces. Further, *virtual radios* such as SpectrumWare [37] may allow sharp bandwidth degradation. Impulse waveforms can arise as a result of frequent transitions in either of these situations, or in the presence of bursty background traffic.

6.1.2 Experimental Methodology

Generating Discontinuous Waveforms To subject Odyssey and its applications to these reference waveforms, we need to generate sharp discontinuities in network bandwidth. Accomplishing this in a repeatable and reliable manner is extremely difficult on any real network or combination of networks. We solve this problem through a technique called *trace modulation* [26, 33].

Trace modulation performs application-transparent emulation of a slower target network over a faster, wired LAN. It is implemented in two parts: a layer inserted in the protocol stack between the transport and network layers, and a user-level daemon. The added layer delays all traffic into and out of the modified host according to a simple linear model combining latency and bandwidth-induced delays. The daemon reads a list of model parameters, called a replay trace, from a file and feeds it to the delay layer. We have created synthetic replay traces to obtain the bandwidth variations corresponding to the reference waveforms.

Interpreting Results Since Odyssey applications trade fidelity for performance, interpreting the results of experiments requires some care. In comparing two strategies, one is strictly better than the other if it provides better fidelity with comparable performance, or better performance with comparable fidelity. In other cases, the comparison must take into account the application's goals.

These comparisons are clearly dependent on the choice of fidelity metrics. However, since only relative comparisons are made within a single application, the only requirement on fidelity is that it be strictly increasing as the quality of presented data increases.

6.1.3 Experimental Conditions

All of our experiments used identical hardware and software configurations: a single 90 MHz Pentium client with 32 MB of memory, and a collection of 200 MHz Pentium Pro servers with 64 MB of memory. These machines were running a NetBSD 1.2 kernel customized to include Odyssey and trace modulation extensions; modulation was performed at the client.

The bandwidth levels encoded in our modulation traces were chosen with two constraints in mind. First, they must be reasonably achieved on current wireless hardware. Second, they must provide for interesting tradeoffs when running our sample applications. We chose 120 KB/s (kilobytes per second) and 40 KB/s for the high and low bandwidth levels. The protocol round trip time measured on our setup was 21 ms for both bandwidths.

6.2 Experimental Results

6.2.1 How Agile is Odyssey?

In order to allow applications to make intelligent trade-offs between performance and quality, Odyssey must track changes in both the supply of and demand for network bandwidth. Because Odyssey may often be used in weakly-connected environments, we rely on purely passive observations rather than an active approach such as that suggested by Keshav [17]. These observations are logged by our user-level RPC mechanism [25] which is implemented on UDP. This mechanism combines a conventional RPC protocol for small exchanges with a sliding-window, selective-acknowledgement protocol for bulk data transfer. Each distinct endpoint has its own log, and observations for different endpoints are recorded independently.

Log entries are of two kinds: *round trip entries* that are recorded for small exchanges, and *throughput entries* that arise from bulk transfers. Each round trip entry records the time, T_{rtt}, to send a request to a server and receive a response, less server computation time. Each throughput entry records T_{win}, which is either the time for a receiver to request and receive a window's worth, D, of data, or for a sender to transmit that data and receive an acknowledgement. Round trip and throughput estimates are both smoothed by the viceroy using the following equation:

$$\text{new} = \alpha(\text{measured}) + (1 - \alpha)(\text{old}) \qquad (1)$$

Our implementation uses an α of 0.75 for round trip time, and 0.875 for throughput.

To obtain a bandwidth estimate, we observe that the transmission time for D bytes is T_{win} minus the time for transmitting the acknowledgement or the time to transmit the request. Assuming symmetrical data rates, both of these are $T_{rtt}/2$. This yields the following expression for bandwidth:

$$B = \frac{D}{T_{win} - (T_{rtt}/2)} \qquad (2)$$

Noise in round trip estimates can severely impact bandwidth estimates; to discount anomalous increases in round trip time, we cap the percentage rise possible at each estimate. This has the effect of erring on the side of caution and underestimating bandwidth in certain situations, but eliminates anomalies introduced by our user-level implementation.

The viceroy collects information from all logs to estimate the total bandwidth available to the client. It then estimates the fraction of this bandwidth likely to be available to each connection. A connection estimate is composed of two parts: a *competed-for* part proportional to recent use, and a *fair-share* part reflecting an expected lower bound.

Varying Supply To measure agility with respect to bandwidth supply, we ran a synthetic Odyssey application, *bitstream*, that consumed data as fast as possible through a streaming warden over a single connection from a server. During data transfer, we varied network bandwidth in accordance with the reference waveforms of Figure 7. To ensure that the system was in steady state, we primed it for thirty seconds prior to observation. The bandwidth estimated by Odyssey for each waveform is shown in Figure 8.

Figure 8(a) shows that Odyssey demonstrates excellent agility on the Step-Up waveform by detecting its bandwidth increase almost instantaneously. The second graph, Figure 8(b), shows that agility on the Step-Down waveform is not quite as good as on Step-Up. The *settling time* for this waveform — the time required to reach and stay within the nominal bandwidth range — is 2.0 seconds. The slower downward transition is caused by the fact that we generate a throughput estimate only at the end of a window of data. If bandwidth falls abruptly while a large window of data is being transmitted, the drop is not recorded until the last packet of the window arrives.

Figures 8(c) and 8(d) show agility for the Impulse-Up and Impulse-Down waveforms. The leading edge of the upward impulse is accurately traced, but the trailing edge has a noticeable settling time. The downward impulse is too short for estimation to settle accurately, and there is again a noticeable settling time on the trailing edge of the impulse.

Varying Demand We next examine agility with respect to bandwidth demand. We began these experiments with a single bitstream application running on a client. As before, we primed the system for thirty seconds to ensure that it was in steady state before beginning observation. After thirty seconds of observation, we introduced a second, identical bitstream client. To study sensitivity of the results to offered load, we repeated the experiments with each application attempting to consume 10%, 45%, and 100% of the nominal throughput. All experiments were conducted at the higher of our two modulated bandwidths.

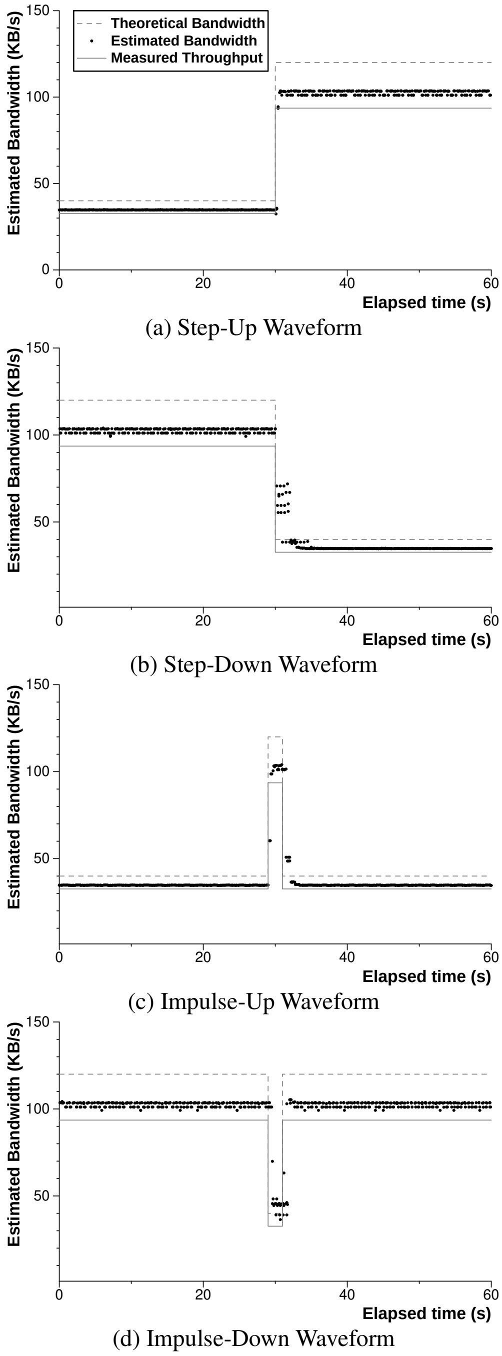
Figures 9(a)–9(c) show the viceroy's estimation of the bandwidth available to the two competing streams. In all experiments, the second stream causes some transient effects when it is started. This transient is much more pronounced in the two higher-load experiments; in the full-utilization case it takes almost 5 seconds to settle back to the nominal value.

At low utilization, the second stream reaches its nominal value almost immediately; in the other two cases, this takes longer. Higher rates of consumption by the first stream give it more weight compared to the startup of the second stream. Hence, the first stream is given more of the competed-for bandwidth until the second stream has established itself.

6.2.2 How Beneficial is Adaptation?

We next compare the performance and fidelity of adaptive applications using Odyssey with versions of these applications using static policies. In this comparison, we represent fidelity as a number between zero and one that indicates, in an application-specific manner, the quality of data delivered.

In our experiments, each application executed the same workload using an adaptive strategy as well as one fixed strategy per fidelity level. Each execution was repeated for all the reference waveforms. As before, we primed the experiment with a thirty second period of constant bandwidth to eliminate startup transients.



(a) Step-Up Waveform

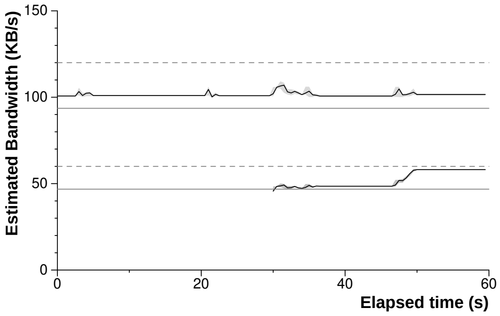
(b) Step-Down Waveform

(c) Impulse-Up Waveform

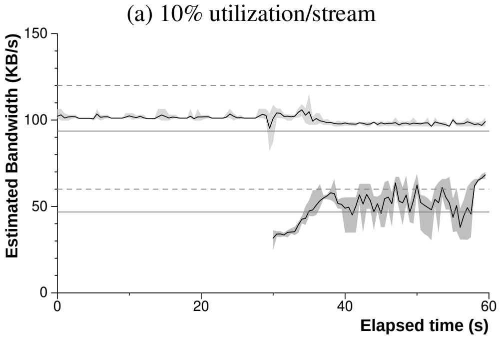
(d) Impulse-Down Waveform

This figure shows the agility of bandwidth estimation in the face of varying supply. Each graph merges the results from five trials, and each bandwidth observation is represented by a single dot on the graph. The dashed lines represent the theoretical bandwidth of the emulated network, as specified by the synthetic traces used for emulation. The dotted lines are the measured, instantaneous throughputs obtained using a large bulk transfer between client and server. Ideally, all samples would lie between the dashed and dotted lines.

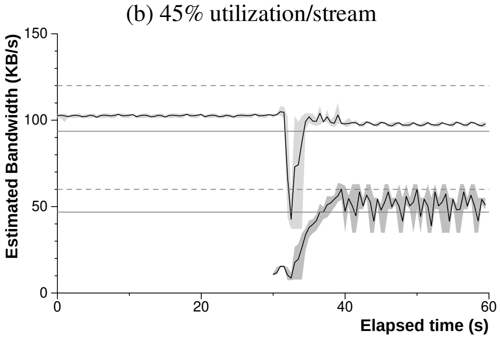
Figure 8: Supply Estimation Agility



(a) 10% utilization/stream



(b) 45% utilization/stream



(c) 100% (attempted) utilization/stream

This figure shows the agility of bandwidth estimation; note that we measure availability, not consumption. The upper curve is the total estimated bandwidth; the lower is the bandwidth available to the second stream, which starts after 30 seconds. The pairs of straight lines show the nominal ranges for each curve; a perfectly agile system would always show the upper and lower curves within their respective pairs. Each graph depicts the results of five trials. The solid lines show averages, and gray regions show the spread between observed maximum and minimum values.

Figure 9: Demand Estimation Agility

Video Player We compare the adaptive behavior of the Odyssey video player to three fixed policies: always JPEG(99), always JPEG(50), and always black-and-white. The higher bandwidth is sufficient to fetch JPEG(99) frames. At the low bandwidth, JPEG(50) frames can be fetched without loss. All movie tracks are encoded at ten frames per second, with 600 frames to display during each trial.

JPEG(99) frames are assigned a fidelity of 1.0, JPEG(50) frames have a fidelity of 0.5, and black-and-white frames a fidelity of 0.01. The fidelity for a single execution of xanim is the average fidelity of frames displayed. Thus a movie with half of its frames displayed from each of the two best tracks would have a fidelity of 0.75. The performance metric is frames dropped. Xanim's adaptation goal is to play the highest quality possible without dropping frames. Other applications might choose, instead, to preserve frame quality but reduce the frame rate.

Figure 10 summarizes the results of the xanim benchmark. In both Step waveforms, roughly half the frames displayed by Odyssey are JPEG(50) frames, and the other half JPEG(99) frames. For Impulse-Up, Odyssey shows only JPEG(50) frames, and for Impulse-Down almost all JPEG(99) frames. Thus, the adaptive xanim nearly always displays the optimal quality frame.

For all waveforms other than Impulse-Up, Odyssey achieves a much better fidelity than JPEG(50), with only a modest increase in dropped frames. For all waveforms other than Impulse-Down, Odyssey drops many fewer frames than the JPEG(99) strategy by reverting to JPEG(50) frames at low bandwidth. For Impulse-Down, Odyssey is indistinguishable from the JPEG(99) strategy.

In the adaptive strategy, dropped frames occur primarily during the downward bandwidth transitions. It takes at least one data transfer for Odyssey to notice the drop in bandwidth, and that transfer is fetching high-quality frames, which are destined to be late.

Web Browser For the Web browser experiments, we repeatedly fetched a 22KB image as fast as possible. To preserve experimental control, the image was stored on a Web server on the test network, with a distillation server interposed between the client and the Web server. The cellophane could choose one of four levels of fidelity: original quality or JPEG compression at quality levels 50, 25, or 5. The fidelity of each of these levels is 1.0, 0.5, 0.25, and 0.05 respectively. The fidelity for an experiment is the average fidelity of all images fetched in that experiment.

The performance metric is the average time to fetch and display an image during an execution. For the baseline against which to compare, we executed the trace on an unmodified, private Ethernet. Our Web client's adaptation goal is to display the best quality image that can be fetched within twice the Ethernet time, in this case 0.4 seconds. With this goal, full-quality images can be fetched at the high bandwidth. At low bandwidth JPEG(50) is the best possible.

Figure 11 summarizes the results of the Web benchmark. The static strategy of fetching full-quality images only meets our performance goals in the Impulse-Down case. This is not surprising, as most of that trace provides sufficient bandwidth for full-quality images. In contrast, Odyssey meets our performance goal in all cases, and does so at better quality than any of the sufficiently fast static strategies. In the Impulse-Up case, Odyssey is fooled into fetching better quality images for a brief period by the impulse's transient increase in bandwidth.

Speech Recognizer For the speech experiments, we recognized a single, short phrase, repeating the recognition as quickly as possible. Since the quality of recognition does not vary, the only interesting metric is the speed with which recognitions take place. Figure 12 gives the recognition times for the three possible strategies: always hybrid, always remote, and adaptive.

At the bandwidths in our reference traces, hybrid translation is always the correct strategy when speech is the sole application. As Figure 12 shows, Odyssey duplicates the always-hybrid strategy. We have confirmed, through experiments not reported here, that at higher bandwidths an adaptive strategy has benefits.

6.2.3 How Important is Centralized Resource Management?

Finally we examine the usefulness of Odyssey's centralized resource management. We do this by comparing Odyssey with two forms of uncoordinated resource management, with all three applications concurrently running on the much longer synthetic waveform shown in Figure 13.

As a basis of comparison we modified the viceroy to support *laissez-faire* resource management; rather than combining information from all logs as in Section 6.2.1, each log is examined in isolation. This reflects what applications would discover on their own: information is less accurate than that globally obtained but with similar delays in discovery.

Waveform	B/W Fidelity = 0.01 Drops	JPEG(50) Fidelity = 0.5 Drops	JPEG(99) Fidelity = 1.0 Drops	Odyssey Drops	Odyssey Fidelity
Step-Up	0 (0.0)	3 (1.8)	169 (0.8)	7 (2.2)	0.73 (0.01)
Step-Down	0 (0.0)	5 (11.2)	169 (2.4)	25 (8.9)	0.76 (0.01)
Impulse-Up	0 (0.0)	3 (0.7)	325 (4.3)	23 (7.4)	0.50 (0.01)
Impulse-Down	0 (0.0)	0 (0.0)	12 (5.7)	14 (6.5)	0.98 (0.01)

This table gives the fidelity and number of frames dropped by xanim under various strategies for each of the four reference waveforms. Note that larger fidelity values represent better quality, while fewer dropped frames indicates better performance. Each observation is the mean of five trials, with standard deviations given in parentheses. Notice that Odyssey achieves fidelity as good as or better than the JPEG(50) strategy in all cases, but performs as well or better than JPEG(99) within experimental error.

Figure 10: Video Player Performance and Fidelity

Waveform	JPEG(5) Fidelity = 0.05 Time (s)	JPEG(25) Fidelity = 0.25 Time (s)	JPEG(50) Fidelity = 0.5 Time (s)	Full Quality Fidelity = 1.0 Time (s)	Odyssey Time (s)	Odyssey Fidelity
Ethernet	—	—	—	0.20 (0.00)	—	—
Step-Up	0.25 (0.01)	0.30 (0.01)	0.29 (0.01)	0.46 (0.01)	0.35 (0.05)	0.78 (0.08)
Step-Down	0.25 (0.01)	0.30 (0.01)	0.29 (0.01)	0.46 (0.00)	0.35 (0.03)	0.77 (0.04)
Impulse-Up	0.27 (0.01)	0.33 (0.01)	0.34 (0.00)	0.71 (0.00)	0.42 (0.06)	0.63 (0.08)
Impulse-Down	0.24 (0.01)	0.27 (0.02)	0.29 (0.01)	0.34 (0.01)	0.36 (0.02)	0.99 (0.01)

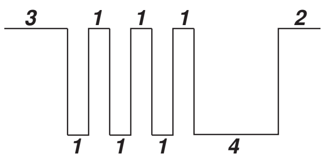
This table gives the fidelity and average time for Netscape to fetch and display our test image under various strategies for each of the four reference waveforms. Note that larger fidelity numbers represent better quality, while smaller times indicate better performance. Each observation is the mean of five trials; standard deviations are given in parentheses. Notice that Odyssey achieves a better fidelity than JPEG(50) in all cases and, unlike the full-quality strategy, meets our performance goal within experimental error for all cases.

Figure 11: Web Browser Performance and Fidelity

	Recognition Time (sec.)		
Waveform	Always Hybrid	Always Remote	Odyssey
Step-Up	0.80 (0.00)	0.91 (0.00)	0.80 (0.00)
Step-Down	0.80 (0.00)	0.90 (0.00)	0.80 (0.00)
Impulse-Up	0.85 (0.00)	1.11 (0.00)	0.85 (0.00)
Impulse-Down	0.76 (0.00)	0.77 (0.00)	0.76 (0.01)

This table gives the average time, in seconds, for a recognition by the speech application for each of the two static strategies as well as the adaptive strategy for each of the four reference waveforms. Each observation is the mean of five trials. Standard deviations are shown in parentheses. Note that Odyssey correctly reproduces the always-hybrid case, which is optimal at our reference bandwidth levels.

Figure 12: Speech Recognizer Performance



This 15-minute synthetic trace models the bandwidth variation seen by a user walking through a hypothetical urban setting. Each number indicates the time duration of the corresponding segment in minutes. The high and low bandwidths are as indicated in Section 6.1.3. The user begins well-connected but soon enters a region of intermittent quality. She then enters the radio shadow caused by a large building, and finally returns to good connectivity.

Figure 13: Bandwidth Variation in Urban Scenario

One could also imagine the networking layer of an operating system immediately notifying applications when switching between networking technologies. We have implemented this strategy, which we call *blind-optimism*, by passing the theoretical bandwidth to the viceroy at each transition via an upcall. The viceroy then notifies any interested applications. This information is less accurate because it does not reflect the impact of any other applications, but is delivered without the delay of bandwidth discovery.

Figure 14 presents the results of this experiment. The fidelity and performance metrics as well as the application goals for this experiment are the same as in Section 6.2.2. The message of Figure 14 is that Odyssey's centralized resource estimation provides significant benefits over both *laissez-faire* and blind-optimism. By correctly accounting for bandwidth competition, the Odyssey Web browser and video player fetch data at lower fidelity, thus enabling all applications to come much closer to their performance goals. Odyssey drops a factor of 2 to 5 fewer frames than the other strategies, and Web pages are loaded and displayed roughly twice as fast. The resulting decrease in network utilization improves speech recognition time as well.

7 Related Work

To the best of our knowledge, Odyssey is the first system to simultaneously address the problems of adaptation for mobility, application diversity, and application concurrency. It is the first effort to propose and implement an architecture for application-aware adaptation that pays careful attention to the needs of mobile computing. The identification of agility as a key attribute for mobile systems, and the first approach to evaluating it, have both occurred in the context of Odyssey.

At the same time, Odyssey has benefited considerably from much

	Video		Web		Speech
	Drops	Fidelity	Seconds	Fidelity	Seconds
Odyssey	1018 (48.6)	0.25 (0.00)	0.54 (0.02)	0.47 (0.01)	1.00 (0.01)
Laissez-Faire	2249 (80.2)	0.39 (0.01)	0.95 (0.03)	0.93 (0.01)	1.21 (0.01)
Blind-Optimism	5320 (23.3)	0.80 (0.00)	1.20 (0.00)	1.00 (0.00)	1.26 (0.02)

This table demonstrates the benefit of Odyssey's centralized resource management by comparing it to two implementations of uncoordinated resource management. The fidelity and performance metrics for this experiment are the same as in Figures 10–12. Notice that by degrading the fidelity of fetched video and web data, Odyssey comes closer to each application's performance goals by factors of 2-5. Such a trade-off is made possible by Odyssey's more accurate estimation of bandwidth available to each application. Each observation in this table is the mean of five trials, with standard deviations given in parentheses.

Figure 14: Performance and Fidelity of Concurrent Applications

previous work. A substantial debt is owed to Coda, which first demonstrated that client resources could be effectively used to insulate users and applications from the vagaries of mobile information access. The strategy of trading lowered consistency for improved availability was shown to be effective and usable by Coda and related systems such as Ficus and Bayou. It was the recognition that consistency represented a particular dimension of the broader concept of fidelity that led to the design of Odyssey. Many aspects of the Odyssey prototype, such as its implementation in user space and its use of a log-based mechanism for monitoring network quality, were based on positive experience with similar strategies in Coda.

Many systems together served as a backdrop to our thinking on fidelity: the Rover toolkit; mobile Web software such as Mobisaic [40] and W4 [2]; software embodying concepts such as dynamic documents [14] and distillation [7]; commercial email packages such as Eudora; and numerous PDAs and pocket organizers. Examination of these systems also helped us identify an essential missing ingredient in all of their designs: effective management of scarce resources across multiple applications. These systems, in conjunction with Coda, helped us formulate our taxonomy of adaptation strategies — *laissez-faire*, application-aware, and application-transparent.

The issues of resource reservation and guarantees lie at the heart of real-time systems [24], and have become important in high performance networking [8]. These communities have recently applied reservation techniques, with two changes, to mobile clients [20, 27]. First, rather than reserving a particular quantity of a resource, they reserve a range; the underlying system transparently adapts within the range. Second, if the range is exceeded or the client moves, a renegotiation involving some or all of the end-to-end path is initiated.

In contrast to these systems, Odyssey abandons the reservation model entirely; either the reserved bounds would be so wide as to degenerate to application-transparent adaptation, or costly renegotiations on behalf of a mobile host would be too frequent. Framed as an end-to-end consideration, ultimate responsibility for coping with changes in resource levels resides with applications. Odyssey's role is only to improve efficiency, agility and fairness by insulating applications from insignificant variations in resource levels, and by providing a focal point for resource monitoring and allocation.

Recent adaptive systems, such as McCanne's RLM [22] and Inouye's video player [12], employ feedback-driven adaptation rather than Odyssey's measurement-based approach. Such systems scale back quality, and hence resource consumption, when application performance is poor, and they attempt to discover additional resources by optimistically scaling up usage. Using application-specific feedback relieves such systems of the need to calibrate to individual resources, but this feedback is per-application. As this paper has shown, this kind of *laissez-faire* approach does not provide for application concurrency, even though it works well for individual applications. Further, attempts to increase resource usage amount to active rather than passive resource monitoring, a questionable strategy when bandwidth is scarce.

Finally, the installation of pieces of code at low levels of the system to encapsulate specialized knowledge about different data types is a common practice in databases [10]. The primary purpose of such code is to improve disk management. The use of wardens in Odyssey resembles this practice, but differs in that wardens support multiple fidelity levels.

8 Future Work

We see many short-, medium-, and long-term tasks ahead of us. The prototype improvements already alluded to will constitute our short-term tasks. Specifically, we intend to incorporate adaptation for objects other than images in the Web application of Section 5.2. We also plan to add support for multiple levels of fidelity in the speech application of Section 5.3.

In the medium term, we plan to enhance the prototype and gain experience with it in real use. First, we will broaden support for resource management to the full range of resources in Figure 3(c), and correspondingly expand the scope of our evaluation. This will enable Odyssey to support a broader class of applications, making it attractive as a platform for serious use. We then expect to deploy Odyssey to a small user community, and to gain empirical feedback to complement our evaluation through controlled experiments.

Our long-term plans are more speculative. Currently, we expect to work in three broad areas:

- The speech application of Section 5.3 suggests the importance of being able to dynamically decide whether to ship data or computation. This capability is currently provided in an *ad hoc* manner by the speech warden. Extending Odyssey to provide full support for deciding between *dynamic function or data shipping* would enable us to more thoroughly explore this tradeoff in mobile computing.
- *Search* of distributed repositories performs poorly when mobile because it lacks the temporal locality needed for caching to be effective in compensating for poor bandwidth. We plan to explore a solution that uses *dynamic sets* [35, 36] in conjunction with support for dynamic function versus data shipping.
- The design of adaptive mobile systems is currently a black art. Developing systematic principles for their design, as well as techniques for analyzing their agility and stability before they are built, would be valuable.

9 Conclusion

The need for adaptation in mobile information access is now widely accepted. In this paper, we put forth the view that application-aware adaptation offers the most general and effective approach to addressing this need. The essence of our approach is a collaborative partnership between applications and the system, with a clear separation of concerns. We argue that previous approaches to adaptation represent limiting cases of this general approach.

The Odyssey architecture supports application-aware adaptation while paying careful attention to a variety of practical considerations. Our prototype confirms the feasibility of realizing this architecture, and its ability to support a wide range of applications. Our evaluation identifies agility as a key enabling attribute of an adaptive system, describes how to measure it, and reports on the agility of the Odyssey prototype. The evaluation confirms that the prototype does a good job of balancing performance and fidelity, and confirms the importance of centralized resource management. At the same time, it suggests avenues for further improvement. Overall, Odyssey promises to be a versatile and effective platform for further research in mobile computing.

Acknowledgements

Markus Baur and Jie Yang helped us considerably in understanding and using Janus. Our use of xanim was made possible by Mark Podlipec. Bob Baron provided valuable assistance with NetBSD on innumerable occasions. Geoff Kuenning, David Maltz, Lily Mummert, Wayne Sawdon, and our SOSP shepherd, Jeff Mogul, provided insightful comments that substantially improved this paper. Finally, the careful and detailed feedback from our anonymous reviewers led to significant improvements in content and presentation.

References

[1] Apple Computer Inc., Cupertino, CA. *Newton Message Pad Handbook*, 1993.

[2] J. F. Bartlett. W4 — The Wireless World Wide Web. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[3] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility, Safety and Performance in the SPIN Operating System. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain, CO, December 1995.

[4] R. C. Dorf, editor. *The Electrical Engineering Handbook*, chapter 93: Control Systems. CRC Press, 1993.

[5] D. Duchamp. Issues in Wireless Mobile Computing. In *Proceedings of the Third IEEE Workshop on Workstation Operating Systems*, Key Biscayne, FL, April 1992.

[6] D. R. Engler, M. F. Kaashoek, and J. O'Toole Jr. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain, CO, December 1995.

[7] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International ACM Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, October 1996.

[8] L. Georgiadis, R. Guerin, V. Peris, and R. Rajan. Efficient Support of Delay and Rate Guarantees in an Internet. In *Proceedings of the ACM SIGCOMM'96 Conference*, Stanford, CA, August 1996.

[9] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole Jr. Semantic File Systems. In *Proceedings of the 13th ACM Symposium on Operating System Principles*, 1991.

[10] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[11] Apple Computer Inc. *Inside Macintosh: Quicktime*. Addison-Wesley Publishing Co., Reading, MA, 1993.

[12] J. Inouye, S. Cen, C. Pu, and J. Walpole. System Support for Mobile Multimedia Applications. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, St. Louis, MO, May 1997.

[13] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain, CO, December 1995.

[14] M. F. Kaashoek, T. Pinckney, and J. A. Tauber. Dynamic Documents: Mobile Wireless Access to the WWW. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[15] R. H. Katz. Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications*, 1(1), 1996.

[16] R. H. Katz and E. A. Brewer. The Case for Wireless Overlay Networks. In *SPIE Multimedia and Networking Conference*, January 1996.

[17] S. Keshav. Packet-Pair Flow Control. To appear in IEEE/ACM Transactions on Networking.

[18] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1), February 1992.

[19] S.R. Kleiman. Vnodes: An Architecture for Multiple File System Types in Sun UNIX. In *Summer Usenix Conference Proceedings*, 1986.

[20] S. Lu, K.-W. Lee, and V. Bharghavan. Adaptive Service in Mobile Computing Environments. In *Fifth IFIP International Workshop on Quality of Service*, New York, NY, May 1997.

[21] A. Luotonen and K. Altis. World-Wide Web Proxies. *Computer Networks and ISDN Systems*, 27, September 1994.

[22] S. McCanne, V. Jacobsen, and M. Vetterli. Receiver-driven Layered Multicast. In *Proceedings of the ACM SIGCOMM'96 Conference*, Stanford, CA, August 1996.

[23] M. K. McKusick, K. Bostic, M. J. Karels, and J. S. Quarterman. *The Design and Implementation of the 4.4BSD Operating System*. Addison Wesley, 1996.

[24] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[25] L. B. Mummert. *Exploiting Weak Connectivity in a Distributed File System*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1996. CMU-CS-96-195.

[26] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz. Trace-Based Mobile Network Emulation. In *Proceedings of ACM SIGCOMM'97 Conference*, Cannes, France, September 1997.

[27] S. Pope and P. Webster. QoS Support for Mobile Computing. In *Fifth IFIP International Workshop on Quality of Service*, New York, NY, May 1997.

[28] G. J. Popek, R. G. Guy, T. W. Page, and J. S. Heidemann. Replication in Ficus Distributed File Systems. In *Proceedings of the IEEE Workshop on Management of Replicated Data*, Houston, TX, November 1990.

[29] Qualcomm Inc., San Diego, CA. *Eudora Macintosh User Manual*, 1993.

[30] O. Rubin. *The Design of Automatic Control Systems*. Artech House, Norwood, MA, 1986.

[31] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), February 1996.

[32] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4), April 1990.

[33] M. Satyanarayanan and B. Noble. The Role of Trace Modulation in Building Mobile Computing Systems. In *Proceedings of the Sixth IEEE Workshop on Hot Topics in Operating Systems*, Chatham, MA, May 1997.

[34] A. Smailagic and D. P. Siewiorek. Modalities of Interaction with CMU Wearable Computers. *IEEE Personal Communications*, 3(1), February 1996.

[35] D. C. Steere. Exploiting Non-Determinism in Set Iterators to Reduce I/O Latency. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, Saint Malo, France, October 1997.

[36] D. C. Steere. *Using Dynamic Sets to Reduce the Aggregate Latency of Data Access*. PhD thesis, Carnegie Mellon University, School of Computer Science, 1997. CMU-CS-96-194.

[37] D. L. Tennenhouse, T. Turletti, and V. G. Bose. The SpectrumWare Testbed for ATM-based Software Radios. In *IEEE International Conference on Universal Personal Communications*, September 1996.

[38] D. B. Terry, M. M. Theimer, K. Petersen, and A. J. Demers. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain, CO, December 1995.

[39] U.S. Robotics, Inc., Los Altos, CA. *Pilot Handbook*, 1996. Part Number 423-0001.

[40] G. M. Voelker and B. N. Bershad. Mobisaic: An Information System for a Mobile Wireless Computing Environment. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[41] A. Waibel. Interactive Translation of Conversational Speech. *IEEE Computer*, 29(7), July 1996.

[42] G. K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4), April 1991.