Abstract

Imagine that, while on a business trip to Paris, you decide to take a few extra days to sample the city's museums. When you buy your museum pass, you are given a virtual tour guide device — a small PDA that can deliver information about specific pieces in the museum system as well as general information about the city. This device communicates via wireless networks. In or near a museum, the device has access to a high-speed micro-cellular network; in the rest of the city, it makes use of the GSM infrastructure. The user can ask this device to elaborate on specific sites or pieces, display related information, or perform geographically-based queries. These requests are satisfied by applications such as a customized Web browser and a video playback application. When within range of a museum's high-quality network, the displayed information is of excellent quality: images are at high resolution and color, and video is delivered in full motion. However, when the user strays away from the high-bandwidth network, each application degrades the quality of the data it delivers so that it arrives in a timely fashion.

System Support for Mobile, Adaptive Applications

Brian Noble, University of Michigan

The prospect of anytime, anywhere network access presents both opportunity and challenge to application writers. Providing connectivity to those on the move enables entirely new services while expanding the reach of current applications. Unfortunately, taking advantage of pervasive networking is difficult. The quality of network connectivity afforded mobile users is extremely *turbulent*: it changes frequently, dramatically, and without warning. Applications must somehow cope with these changes.

There are many reasons for this dynamic behavior. An individual wireless channel is subject to path loss, fading, and environmental interference [1]. Together, these can have a significant impact on the performance of the channel. Further, overlay networks arrange wireless coverage as a set of overlapping technologies, each providing a different tradeoff between bandwidth, coverage, cost, and reliability [2].

How can applications deal with such variation? Ideally, they would take advantage of high-quality connectivity when available, but behave reasonably over networks with poor performance. In the past, systems have dealt with variation by trading plentiful resources for those that are scarce. For example, Web caches [3] attempt to insulate clients from the vagaries of wide-area network performance by spending disk space. Unfortunately, such techniques are not able to hide the orders-of-magnitude changes in performance that are all too common in wireless networks.

Rather than rely on the system to manage resources transparently, applications must themselves *adapt* to prevailing network characteristics. This article presents an overview of our experience with *Odyssey*, a platform for adaptive mobile data access. We have developed several applications for Odyssey, including a Web browser, a video player, and a speech recognition system. Each of these applications is described in more detail in a forthcoming paper [4].

Odyssey's approach to adaptation is to adjust the quality of accessed data to match available resources. For example, when faced with a sharp decrease in bandwidth, a Web browser might ask for more highly-compressed images; a video player may reduce frame rate or frame quality of the stream; and a map viewer may filter out small or irrelevant features. In order to trade data quality for resource consumption and performance, one must first have a notion of quality. We introduce *fidelity*, Odyssey's notion of data quality, and present a few simple examples of its use.

We present Odyssey's division of labor between applications and the operating system for making adaptive decisions. In this division, the system provides the mechanisms enabling adaptation, leaving applications free to set adaptive policy. We present the client architecture used to provide this collaboration, and discuss the programming model induced by it.

In order to adapt to turbulent environments, a system must react to significant changes as fast as they occur. In other words, an adaptive system must be as *agile* as possible. We discuss the importance of agility, and describe how it can be evaluated. This section concludes with a description of the bandwidth estimation mechanism, which proves to be the limit on agility.

Unfortunately, maximizing agility can come at the expense of stability. Often, this tradeoff is preferable; optimizing for agility provides the best possible fidelity given the resources available. But, what should one do when the pursuit of agility leads to instability that is disconcerting for the user? We present this problem, and outline our plan to address it.

Context for this Work

Odyssey was designed to support a broad range of *mobile information access* applications. Such applications reside on a mobile client, but access or update data stored on a remote server. We assume that these servers are more capable and trustworthy than clients; they need not be mobile and are administered by some central authority. Our target client platform has been laptops running a variant of UNIX, typically NetBSD or Linux.

Given this model, we have focused on the client as the provider of adaptive services in the system. The client is best positioned to understand what resources are available to it and what its current priorities are in using those resources. Therefore, adaptive decisions are driven entirely by the client. For their part, servers are able to support these adaptive decisions by providing data at various qualities, but do not take an active role in quality selection or resource monitoring.

Fidelity: A Measure of Data Quality

In order to define a notion of data quality, one must first have a standard against which to compare: a *reference copy*. Any data item accessed by a client has such a reference copy, which is the most complete, current, and detailed version of that item available. When resources are plentiful, a mobile client will access and manipulate the reference copy. However, when resources become scarce, the mobile client may choose to access or manipulate an item that has been degraded, consuming fewer resources. We define *fidelity* as the degree to which a data item used by a mobile client matches the reference copy. Note that lossless compression is not a fidelity-changing operation, as the compressed object is indistinguishable from the original.

Fidelity is a property of potentially many dimensions. One such dimension, consistency, is shared by every data item regardless of type. For example, suppose a mobile client has cached some data element that has been updated by some other host. If network bandwidth to that host is plentiful, the client might fetch the new version immediately. However, if bandwidth is scarce, the client might choose to defer retrieval, instead working with the stale copy.

While the consistency of any data item may be weakened regardless of type, other dimensions of fidelity require knowledge of the item's structure. For example, one might degrade video streams in several ways: reduce the frame rate, reduce the quality of individual frames, or reduce the size of individual frames. Any of these operations requires significant knowledge about the video's representation. Likewise, a map might be degraded by omitting features below a certain size, or by showing only elevations and rivers, but not roads.

It is important to remember that fidelity is per type, not per application. A video editor and a video player have the same dimensions of fidelity at their disposal, though they may choose to manipulate them differently.

Figure 1 illustrates the notion of fidelity as a multi-dimensional property. Figure 1a shows the reference copy of a picture of the University of Michigan's EECS building. In Fig. 1b, the size of the image is the same as in the reference copy, but the image quality is substantially degraded; in Fig. 1c the converse is true. Figure 1d shows the image both at reduced size and image quality.

In order to effectively trade fidelity for performance, one must quantify it. As we are still exploring the space of possible adaptations, we have focused on a single dimension per data type. The fidelity metrics for each possible fidelity are assigned as a scalar value between 0.0 and 1.0. The latter is assigned to the reference copy, while the former would be assigned to a representation with no information content. While the assigned metrics do provide a total ordering on available fidelities, they do not provide a relative measure; a fidelity of 0.5 cannot be said to be "half as good" as a fidelity of 1.0.

Clearly, a relative weighting would be useful. However, such a definition of fidelity depends critically on the perceived quality of each version. For some data types, such as images [5] and video [6], perceptual quality is relatively well understood. For others, such as maps, the problem has not been fully examined. Furthermore, the relative importance of different fidelity dimensions may depend on the use to which the data is being put. For example, video data from a teleconference may more usefully preserve motion while sacrificing detail. In contrast, video of a lecture might need to preserve detail so that the chalkboard is readable.

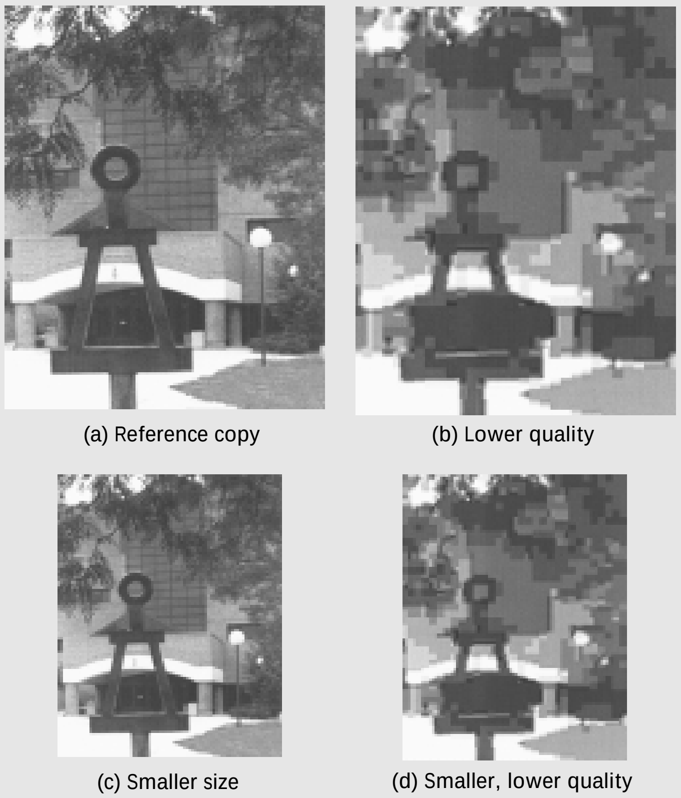

Figure 1. *Fidelity dimensions for images.*

Application-Aware Adaptation: Shared Responsibility

Which entity on the client is responsible for making adaptation decisions: the operating system, or individual applications? Odyssey takes the position that decisions must be made jointly by the two parties. The operating system, as the arbiter of shared resources, is in the best position to determine resource availability. It provides the mechanisms for adaptation and a common point of resource control. However, the application is the only entity that can properly decide how to adapt to a given situation, and must be free to specify adaptive policy. We call this collaborative model of responsibility *application-aware adaptation*.

Why a Collaboration?

To see why such a collaboration is required, consider the alternatives. In the first, called *laissez-faire adaptation*, concurrent applications compete for resources, but each is solely responsible for its own adaptation. In this model, each application must infer the resource consumption of the others to make the best possible adaptation decisions. However, without a common point of resource control, they cannot have accurate knowledge of one another, and will tend to adapt at cross purposes. Correctly accounting for competing applications through a central point of resource control is critical to providing good adaptive behavior. Experiments with a set of representative applications show significant improvement in quality within specified performance bounds when compared to *laissez-faire* adaptation [7].

At the other extreme, consider the case where the operating system on the mobile node is wholly responsible for making adaptation decisions for applications; we call this model *application-transparent adaptation*. In it, the competition for scarce resources is correctly accounted for, but individual

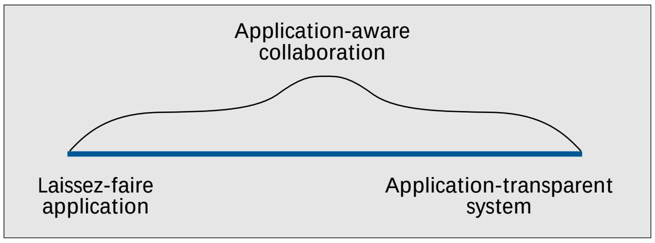
Application-aware collaboration
Laissez-faire application
Application-transparent system

Figure 2. *Spectrum of adaptation models.*

applications cannot choose to make their own adaptive decisions. For example, the teleconference user and the student viewing a remote lecture would each be forced to use the same adaptive policy. It is hard to imagine a single policy that could satisfy both users' needs.

Of course, systems providing application-aware adaptation might decide to divide functionality between the operating system and applications in any number of ways. In other words, application-aware adaptation describes a spectrum of approaches between *laissez-faire* and application-transparent adaptation, as illustrated in Fig. 2. Odyssey explores a single point in this space.

Architecture of an Adaptive Client

The architecture through which Odyssey provides application-aware adaptation is shown in Fig. 3. Odyssey can be most properly thought of as a set of operating system extensions supporting adaptive applications. It was implemented as a kernel component in user space for simplicity, but could have also been implemented directly in the operating system kernel or as a middleware layer.

Odyssey provides access to objects that it manages through the client file system. The *interceptor* module in the kernel redirects file system operations on these objects to Odyssey proper, which comprises two kinds of components: the *viceroy* and a set of *wardens*.

The viceroy is responsible for all type-independent tasks on the client. The most important of these is monitoring resource usage and notifying applications of significant changes in resource availability. The viceroy acts as the single point of resource control, enabling support for concurrent applications.

Because adaptive policy is the province of applications, they must be able to specify which changes in resource availability are important to them. Since this information must flow across the application/kernel boundary, providing it is potentially expensive. However, applications need not be informed of every small change to a resource. To see why, consider a video conferencing application that adapts to changes in network bandwidth. At a given bandwidth, there is some lower bound below which the application will be forced to degrade the video stream. Likewise, there is an upper bound above which the application might increase video fidelity. Together, these bounds form a *window of tolerance* on bandwidth.

When an application chooses a fidelity, it informs Odyssey of the windows of tolerance for any resources of interest. It does so through a new system call, the *resource request*, naming the window of tolerance for one resource along with a function to call if that window is violated. This request is passed on to the viceroy. Thereafter, as the viceroy updates its estimates of resource availability, it will notify an application of any estimate that lies outside of its declared window through an *upcall* [8]. This carries with it the new resource value so that the application might properly react. Details of these and other Odyssey extensions to the application programming interface can be found elsewhere [9].

Fidelity, and hence adaptation, is a type-specific notion. The wardens are the code components in Odyssey that provide all type-specific functionality, one per type. Wardens

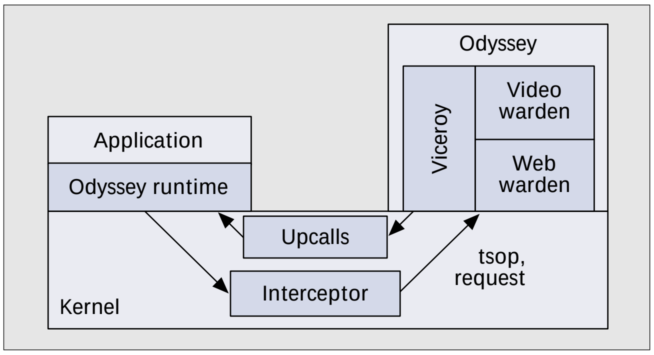

Figure 3. *Odyssey client architecture.*

manage all communication between the client and various servers, and offer a menu of fidelities from which applications can pick. Their inclusion in the operating system allows one to incorporate type-specific knowledge into resource usage decisions. For example, the wardens can provide customized buffer cache replacement policies when the expected data access patterns warrant.

To avoid explicitly encoding each fidelity-changing operation in the API, Odyssey instead provides a general mechanism, the *type-specific operation*, or *tsop*. The `tsop()` call is not interpreted by any code component other than the warden; it is similar in spirit to the `ioctl()` system call found in most UNIX-like systems. In addition to fidelity-changing operations, `tsop()` also enables the provision of type-specific access methods. Applications can then use abstractions more suited to the data at hand than the low-level read/write interface provided for untyped files. For example, video applications can speak in terms of frames of video, rather than ranges of bytes. This can simplify applications significantly.

Each Odyssey application is linked with the *Odyssey runtime library*. This library hides the details of the `tsop()` call behind more meaningful abstractions. It also provides the required user-level support for upcalls used in resource notification.

Programming Model

The programming model induced by Odyssey's API is one of an adaptive decision loop, illustrated in Fig. 4. This control loop first selects a fidelity, then places resource requests with the viceroy as appropriate. The viceroy monitors resource availability and, when a significant change is detected, informs each interested application. The applications then select a new fidelity, repeating the process.

In each of our sample applications, this adaptive decision loop is outside of the main body of control; it is invoked asynchronously by the upcall mechanism. This has important implications for adaptive applications, namely, that they be equipped to handle data of differing fidelities as it is presented.

While this may seem a significant restriction, it has not proven to be so. This is because data types amenable to sophisticated fidelity degradations are self-descriptive, and the applications using them are already capable of handling many different representations. For example, a Web browser is able to decode a number of different image formats, and images of a single format are all decoded the same way, regardless of the degree of compression. The same is true for most other data types of interest.

To take the best advantage of Odyssey's services, applications must be modified to use its API. However, there are ways to support *shrink-wrapped* applications, those for which source code is unavailable. For example, we have incorporated Netscape's Web browser into Odyssey by interposing a proxy between it and Odyssey. This proxy acts as the adaptive application. By intercepting file and network system calls, one could achieve similar results for other legacy applications.

Agility: The Limit on Adaptation

In order to provide support in the broadest possible set of environments, an adaptive system must be as agile as possible. That is, it must be able to react to true changes in the environment quickly. A system that cannot adapt as fast as changes occur will be forced to lower its expectations to the worst case, or suffer unacceptable performance.

The agility of Odyssey is determined by the adaptive decision loop in Fig. 4. The first step of this process, fidelity selection, is the province of the application and server. However, Odyssey is responsible for the other three steps; they determine the upper bound on agility for any application.

Measuring the costs of placing requests and notifying applications is straightforward. Micro-benchmarks show that the time for each of these operations is on the order of a millisecond [6], approximately the cost of a null IPC between two processes on that same machine. These costs could be made substantially smaller with a better-tuned IPC system or an in-kernel implementation.

Measuring Agility

While requests and notifications are easy to measure, it is more challenging to measure the system's ability to detect changes in network performance. It is tempting to run the system over a live, wireless network and record the results. However, it is not clear how one should interpret those results. Wireless networks exhibit performance that is both rapidly changing and impossible to duplicate precisely. The former leads to uncertainty as to how the system should perform in a given trial, while the latter makes aggregating multiple trials difficult.

Instead, we use a technique called *network trace modulation*. Trace modulation is an extension to a client's operating system that delays or drops all network packets according to a simple performance model. The parameters to this model can be varied over time by providing a list of parameters, and the duration for which each parameter set should be applied; this is called a *replay trace*. This technique allows us to approximate arbitrary network performance, much as a sequence of short, straight lines approximates a curve.

Trace modulation allows live software to be run over a simulated networking environment in real time. The description of this environment – the replay trace – can be generated either *empirically* or *synthetically*. Empirical traces are generated by first measuring the performance of a wireless network from a mobile client's perspective over some period of time. These performance observations can then be *distilled* to a replay trace that, when run on a client capable of trace modulation, faithfully recreates the performance seen by the measuring host. Using this technique, one can gain insight as to how a system will perform in the wireless environment, but at a complexity comparable to performing experiments in a wired networking testbed. A detailed description and validation of empirical trace replay can be found elsewhere [11].

Empirical trace replay has been a valuable tool in our evaluation of mobile systems. However, it falls short in assessing the agility with which one can detect changes in networking performance. This is because the network performance recreated by an empirical trace is still too complicated to yield to tractable analysis.

Instead, we subject Odyssey to synthetic traces that provide instantaneous, ideal changes in network bandwidth. This process is very similar to the control-theoretic technique of *impulse response analysis*. By characterizing real client performance in the context of such ideal circumstances, it is much simpler to see what limits to agility are placed by the network estimator.

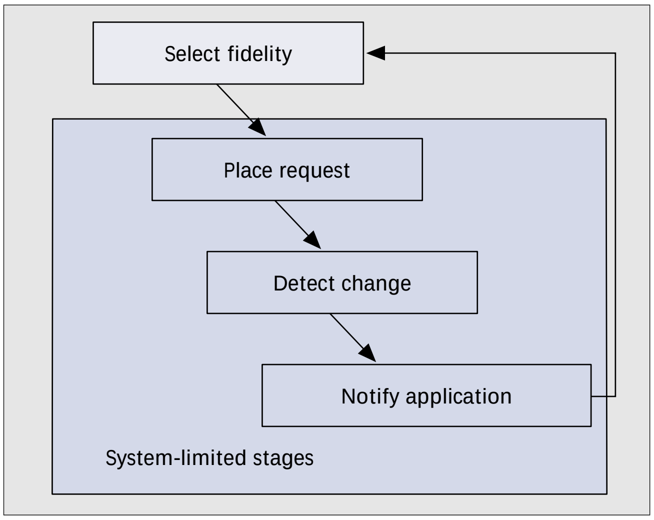

Figure 4. *Adaptive decision loop.*

Network Estimation in Odyssey

In order to provide adaptive services, an Odyssey client must estimate the quality of the network paths used by various applications. It does so through passive observation of the traffic generated by these applications, all of which passes through the viceroy. This traffic is composed of both short interactions with servers through remote procedure call, or RPC, as well as bulk transfer of large objects [12]. The latter provides sliding-window, selective-acknowledgement flow control.

The communications layer in Odyssey records the time required for each RPC, minus the server-side computation time. This gives an estimate of round-trip time, and hence latency, to the server in question. By observing the time needed to complete each window of a bulk transfer, the client can estimate throughput. Given both throughput and latency, one can estimate the available bandwidth to the server. Odyssey updates its estimates of latency and bandwidth using a simple low-pass filter, updating once every half second. This filter biases heavily toward the most recent observations to provide the best agility possible.

Note that this scheme assumes the path between the server and client is *symmetric*; that is, that the performance from server to client is identical to that from client to server. In practice this is not true, though often the differences in performance are small. To account for asymmetry, one would need reliably synchronized clocks at each end of the path in question.

Detecting Decreases: Limit on Agility

To determine Odyssey's agility in detecting changes in network bandwidth, we subjected it to a set of *reference waveforms*. These waveforms are synthetically generated replay traces, used by the trace modulation tool to vary the bandwidth available to the client. Because Odyssey must rely on passive observations to produce estimates, we run a synthetic application that streams traffic between a server and an Odyssey client as fast as the underlying network will allow. This removes any application-dependent limits on agility, allowing us to focus on those limits placed by Odyssey proper.

The results for two of these waveforms are shown in Fig. 5 and Fig. 6. Each waveform lasts one minute. The first waveform, called *step-up*, instantaneously increases the available bandwidth from 40 kbytes/s to 120 kbytes/s halfway through the trace. The second waveform, called *step-down*, decreases bandwidth from 120 kbytes/s to 40 kbytes/s. The *x* axis is time in seconds; the *y* axis is estimated bandwidth, in bytes per second.

Each graph depicts the waveform, as well as all estimates made by the viceroy over five trials. The waveform itself is

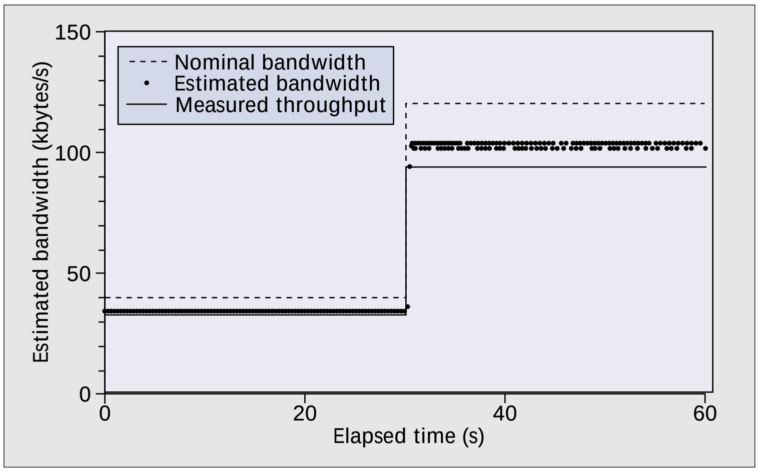

Figure 5. *Detecting increased bandwidth.*

shown as two gray curves. The uppermost, dashed curve shows the nominal bandwidth of the waveform. The lower, dotted curve shows the long-term throughput achieved by the synthetic applications over a modulated network of that bandwidth. Each observation is represented as a single dot. A perfectly agile estimator would produce observations entirely within these two curves.

As Fig. 5 shows, increases in bandwidth are easy to detect; the estimates reach the new value within the sampling period. However, Fig. 6 paints a more pessimistic picture for decreases in bandwidth. In each experiment, several estimates fall in between the old and new bandwidth values. This is because Odyssey is limited to observing completed windows of data. When network performance is good, these windows are allowed to grow large. When performance later drops precipitously, the in-flight window is elongated in time, resulting in estimates that are off the mark. This problem is exacerbated by the protocol's efforts to perform its own adaptations, further delaying timely performance updates.

Stability: Improving the User Experience

The agility of a system is an important metric. It defines the most turbulent environment in which a system can operate. A more agile system will provide better fidelity and performance to users than a less agile one would. However, pursuing agility while completely sacrificing stability can be counterproductive.

To see why, consider a video player with three available fidelities. Each of the fidelities provides the same frame rate, but a different frame quality: high-quality color, low-quality color, and grayscale. To the user, the difference between the first two fidelities is small; one has to look closely to see the artifacts introduced in the low-quality color frames. However, the perceived difference between the latter two fidelities is large. Rapidly switching between them is likely to result in an unpleasant experience for the user.

We believe there is a general principle underlying this observation: users are tolerant of frequent changes between fidelities with small perceptual differences, but are intolerant of frequent, perceptually large changes. One might suspect that users would prefer to remain at a lower fidelity than be exposed to such changes, much as users prefer systems with predictable response times to those with variable latency, even if the average response time is somewhat longer [13].

Tolerance of change, and hence the need for stability, depends on both the data type at hand and the application making use of it. The former is easy to see, as stability is required only when perceptual differences are pronounced. However, one can imagine applications that would prefer to increase or decrease stability independent of perceptual quality. We are currently exploring ways to include stability in Odyssey applications. The remainder of this section outlines our plans for doing so.

Incorporating stability in the core system – the viceroy – would unnecessarily limit its applicability. Because the viceroy imposes a fundamental limit on the agility of all applications, decreasing its agility would hamper those applications requiring more. Instead, stability is properly incorporated by individual applications, as their circumstances warrant.

For example, consider again the video player with three fidelities. When playing at the middle fidelity, it must quickly degrade to the lowest one when bandwidth decreases, else it might suffer a loss of service. However, when playing at the lowest fidelity, the video player might choose to be *skeptical* of an increase in bandwidth, and delay the corresponding increase in fidelity until it is clear that the change in performance is a persistent one. This notion of skepticism is quite general. Each proposed increase in fidelity might be met with some degree of skepticism, proportional to the perceptual distance between old and new fidelities.

While the viceroy should not provide stability itself, it can assist applications in the task of incorporating skepticism. When notifying an application about a change in resource availability, the viceroy could also include information about the expected variance in that estimate. When variance is low, the application can place more faith in the new estimate, and reduce its skepticism accordingly. Conversely, when variance is high, the application must increase its skepticism to avoid disconcerting oscillations in fidelity choice. While the case for including uncertainty with estimation is clear for adaptive applications, there are many other domains in which it might be useful [14].

Future Work

While we have gained substantial experience in providing adaptive services to applications, there is much left to be done. We are currently extending our work along three fronts:

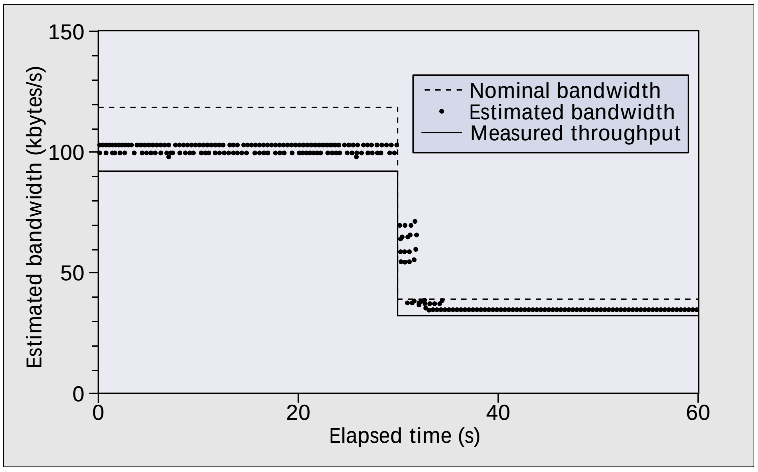

Figure 6. *Detecting decreased bandwidth.*

improving our notion of fidelity, supporting peer-to-peer, collaborative services, and improving the quality of information passed across the software/radio interface.

Quantifying fidelity in terms of perceptual quality will allow us to make more direct comparison between different adaptive schemes. Together with the exposure of variance in resource estimates, it also enables an exploration of how one might properly balance the competing concerns of agility and stability.

We are also exploring the impact that *interactive, collaborative services* place on Odyssey's model of adaptation. Supporting such services is clearly important; doing so will allow us to directly support peer-to-peer systems. When compared to stored information, the timeliness constraints of interactive services reduces the degree to which buffering can mask network variations. Further, one often has at best an approximate estimate of how on-line fidelity degradation will affect the size and resource requirements of data. Thus, one needs to cope with unforeseen variations in the *demand* for resources, as well as the supply of them.

Finally, we are exploring the currently inviolable interface between wireless devices and mobile software. Most software assumes only that wireless networks forward and deliver packets as best they can. Likewise, devices treat software only as equal-opportunity consumers and producers of packets. At best, each layer hides valuable information from the other; at worst the layers adapt at cross-purposes. In the Mackinac project, we are exploring the use of *translucent interfaces* to exchange the right information across these two layers without unduly complicating either of them. There are many potential applications of such information. For example, it might allow us to more rapidly detect certain classes of network constriction events, such as that in Fig. 6.

Conclusion

We have been exploring application-aware adaptation through Odyssey for the past several years. Structuring an adaptive, mobile system as a collaboration between the operating system, which provides the mechanisms for adaptation, and the applications, which supply adaptive policy, has proven to be powerful. Applications can take their own goals into account when choosing how to adapt to mobile environments, but can do so safely in the presence of other applications competing for the same set of scarce resources.

Acknowledgments

Although this article has a single author, it describes work with many contributors at both the University of Michigan and Carnegie Mellon University. Much of the work on Odyssey was performed at CMU under the author's doctoral advisor, M. Satyanarayanan, and with the help of many colleagues, including Jason Flinn, Dushyanth Narayanan, Morgan Price, Eric Tilton, and Kip Walker. Work continues on the problem of mobile data access at Michigan with the help of David Barkovic, Mark Corner, Ben Fleis, Minkyong Kim, Jim Zajkowski, and the author's colleague, Kimberly Wasserman.

This research was supported by the Air Force Materiel Command (AFMC) under DARPA contract numbers F19628-93-C-0193 and F19628-96-C-0061, and the Space and Naval Warfare Systems Center (SPAWAR) under DARPA contract number N66001-96-C-8505. Additional support was provided by AT&T, IBM, and Novell. The views and conclusions contained here are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, SPAWAR, DARPA, AT&T, IBM, Novell, The University of Michigan, Carnegie Mellon University, or the Governments of the United States or the State of Michigan.

References

[1] T. S. Rappaport, *Wireless Communications: Principles and Practice*, Upper Saddle River, NJ: Prentice Hall PTR. 1996.
[2] R. H. Katz and E. A. Brewer, "The Case for Wireless Overlay Networks," *Multimedia Computing and Networking 1996*, San Jose, CA, Jan 1996, pp. 77–88.
[3] A. Luotonen and K. Altis, "World-Wide Web proxies," *Computer Networks and ISDN Sys.*, vol. 27, no. 2, Nov. 1994, pp. 147–54.
[4] B. D. Noble and M. Satyanarayanan, "Experience with Adaptive, Mobile Applications in Odyssey," *Mobile Networks and Apps.*, to appear.
[5] A. B. Watson, Ed., *Digital Images and Human Vision*, Cambridge, MA: MIT Press, 1993.
[6] A. A. Webster *et al.*, "An Objective Video Quality Assessment System Based on Human Perception," *Human Vision, Visual Processing, and Digital Display IV*, Feb. 1993, San Jose, CA, pp. 15–26.
[7] B. D. Noble *et al.*, "Agile Application-Aware Adaptation for Mobility," *Proc. 16th ACM Symp. Op. Sys. Principles*, St. Malo, France, Oct. 1997, pp. 276–87.
[8] D. D. Clark, "The Structuring of Systems Using Upcalls," *Proc. 10th ACM Symp. Op. Sys. Principles*, Orcas Island, WA, Dec. 1985, pp. 171–80.
[9] B. D. Noble, M. Price, and M. Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing," *Comp. Sys.*, vol. 8, no. 4, 1995, pp. 345–63.
[10] B. D. Noble, "Mobile Data Access," Carnegie Mellon University, Ph.D. thesis, CMU-CS-98-118, 1998.
[11] B. D. Noble *et al.*, "Trace-Based Mobile Network Emulation," *ACM SIGCOMM '97 Conf. Apps., Technologies, Architectures, and Protocols for Comp. Commun.*, Cannes, France, Sept. 1997, pp. 51–62.
[12] M. Satyanarayanan, "RPC2 User Guide and Reference Manual," School of Comp. Sci., Carnegie Mellon Univ., Oct. 1991.
[13] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3rd ed., Reading, MA: Addison-Wesley. 1998.
[14] B. D. Noble, L. Li, and A. Prakash, "The Case for Better Throughput Estimation," *Proc. 7th Wksp. Hot Topics Op. Sys.*, Rio Rico, AZ, Mar. 1999.

Biography

BRIAN NOBLE (bnoble@umich.edu) is an assistant professor in the Electrical Engineering and Computer Science Department at the University of Michigan. His research centers on the software supporting mobile computing systems and wide area information access, including networking, infrastructure, and end-system concerns. He received the Ph.D. in computer science from Carnegie Mellon University in 1998.