Chapter 1

Mobile Adaptive Computing

1.1 What Is Mobile Computing?

What is mobile computing? What are its different aspects?. Basically, mobile computing systems are distributed systems with a network to communicate between different machines. Wireless communication is needed to enable mobility of communicating devices. Several types of wireless networks are in use currently, and many books have been devoted to mobile communication. In this book we cover only the basic aspects of mobile communication. However, this book is not devoted to wireless communication and networks. Readers not familiar with wireless communication and networking can refer to Appendix A for a short description. In this book, we are concerned mostly with the logical aspects of mobile communication.

What is the difference between mobile computing and mobile communication? Communication is necessary for distributed computing. Many mobile computing tasks require mobile communication. But that is not the end of it. Mobile communication does not solve all the problems. As we will see in this book, many issues need to be resolved from a higher-level perspective than just being able to exchange signals/packets.

What interests us the most about mobile computing is what we can do with this new facility of mobile communication. What new applications can be enabled? Think about the applications we currently use. Many of these applications do not adapt very well to our needs. For example, suppose that we are browsing the World Wide Web and are interested in going to an Italian restaurant for lunch. What information will we get when we do a search for “Italian restaurant” through search engines such as Google? Usually we get a lot of information that we do not care. The search engines of today do not care about who is invoking the search or the location from where the search is being invoked. You can type “Italian restaurant in Tempe” or “Italian restaurant near Arizona State University.” You can make your query as specific as possible. But wouldn’t it be nice if the applications were aware of your location or, more generally, your context and if the response from the search engine were ordered according to your context? Surely, such context-aware applications will make us more productive.

Let us consider another application, such as video streaming over the Internet. Suppose that you are on the move, watching a movie. Wireless communication is

different from wired communication. When you have a wire, you usually have a fixed bandwidth available to you and your application. Once the application is started and the movie starts, you can watch the movie at a good *quality of service* (QoS). (Currently, this is not really true if streaming is done across the Internet. Nevertheless, it is expected that this will become a reality in the near future.) In wireless communication, however, the bandwidth is mostly shared among several users in a dynamic fashion. This is to say that no dedicated bandwidth is available. Even if your application can reserve certain wireless bandwidth, the usable bandwidth fluctuates owing to the nature of the wireless medium. There are both short- and long-term fluctuations. The question is, "How should the application respond to these fluctuations?" One way that the application can respond is in a uniform manner, irrespective of what you are watching. The other approach is to respond based on the type of movie you are watching. For example, suppose that you are watching an action movie. The application can reduce the bandwidth requirement by switching from full-color video to black and white or by reducing the resolution. On the other hand, if you are watching an interview of Bill Gates, the application may switch simply to audio streaming. There are several important points to note here. These decisions are based on the content of the video, and the decision making may involve both the client and the server (the client needs to inform the server that it no longer wants the video frames, only the audio).

In this chapter we will reexamine computer (software) system design to see how it needs to be changed to accommodate mobility. As we will see, many of the changes are concerned with providing mechanisms for adapting to changing environmental and system conditions, such as the location and the available resources. Mobile computing is about providing information anytime anywhere or, more generally, computing anytime and anywhere.

Mobile computing is also about dealing with limitations of mobile computing devices. For example, *personal digital assistants* (PDAs) and laptops have small interfaces and are powered by batteries. One of the major issues is how to do computation in an energy-efficient manner. Battery technology is not advancing at the same pace as processor technology, so it is not expected that battery capacity will double at a fixed rate, as is true about processor speeds, according to Moore's law. One does not have to deal with these issues when designing systems for stand-alone or distributed systems. One may have to deal with issues of fault tolerance in distributed systems, such as server crashes or network link failures. However, energy usually is not an issue. In mobile systems, energy becomes a resource like processing time or memory space. Therefore, one now has to design resource management techniques for energy, just as traditional operating systems deal with process and memory management.

Different computers in a mobile computing environment may have different capabilities. Any collaborative activity between these devices needs an underlying software entity to deal with the heterogeneity of these devices. This software entity is called a *middleware layer*. A middleware layer also may allow a mobile device and a

wired device to interact. When mobile clients move from one administrative domain to another, they may want to know what new services are available.

How about security or privacy? Wireless communication takes place over an "open" wire and is relatively easy to tap. It may seem that traditional techniques of cryptography can be used to secure wireless communication. However, the main problem is that these secure techniques are designed for wired networks and are computation- and communication-intensive. Attempts to reduce these overheads lead to security schemes that are relatively easy to break. We will look into security schemes for mobile networks in later chapters.

1.2 Adaptability—The Key to Mobile Computing

Humans have evolved to be on the top of the food chain. It is probably undeniable that we have done so by being able to adapt quickly and effectively to varying situations. It is only because of our ability to adapt that we can be found from Arctic regions to the Sahara Desert. What are the techniques by which we are able to adapt to such diverse environments? Can we incorporate some of these techniques into our computing systems? Anyone who has used computers seriously would like them to be more resilient and adaptive to our needs and circumstances. Computing systems and applications fail for various reasons. What is most frustrating is when they fail for no apparent reason. You install a new application, and some other apparently unrelated application stops functioning. And sometimes we just want our computers to learn from our past actions and act proactively and appropriately. Making systems resilient and adaptive is not a trivial task. It took us millions of years to rise to the top of the food chain. Computers as we know them today are not even a hundred years old.

The vision of mobile computing is to be able to roam seamlessly with your computing devices while continuing to perform computing and communication tasks uninterrupted. Many technological advances at various fronts such as security, privacy, resource allocation, charging, and billing have to be made to make this feasible. A quintessential characteristic of any solution to mobile computing problems is its ability to adapt to dynamic changes in computing and communication environments. A system's agility to react to changes in the computing environment and continue its computing tasks uninterrupted is a new measure of performance in mobile computing environments.

Consider a scenario in which you move from one coverage area of an access point to another while a video streaming application is running on your computer. To continue receiving the video stream uninterrupted and without deterioration in video quality, the video stream packets now should be routed automatically through the new access point. In an Internet Protocol (IP)–based network this may involve the mobile client obtaining a

new IP address in the new access point's IP network and informing the server of the new address so that it can now send the packets to the new address. Many more sophisticated solutions to this problem have been developed. The point here is that the underlying system has to take many actions automatically to ensure continued connectivity, in this case uninterrupted viewing of the video stream. In essence, the system has to adapt to the changes in the environment, such as the network configuration and the availability of communication and computation resources and services. However, is this enough? More specifically, the preceding adaptation scheme does not take into account the applications requirements, and the applications themselves did not play any part in the adaptation. Does this application-transparent way of adapting suffice to meet the goals of mobile computing?

1.2.1 Transparency

Transparency is the ability of a system to hide some characteristics of its underlying implementation from users. Much of the research effort in distributed computing has been devoted to developing mechanisms for providing various forms of transparency. Examples of these include the following:

- *Access transparency* is the ability of a system to hide the differences in data representation on various machines and the mode of access of a particular resource.
- *Location transparency* is the ability of a system to conceal the location of a resource. Related to location transparency are *name transparency* (which ensures that the name of a resource does not reveal any hints as to the physical location of the resource) and *user mobility* (which ensures that no matter which machine a user is logged onto, she should be able to access resources with the same name).
- *Failure transparency* is the ability of the system to hide failure and recovery of a system component.

Mobile computing systems can be viewed as a form of distributed system, and attempts can be made to provide "mobility transparency," which would encompass the transparencies just mentioned. This would, in essence, support application-transparent adaptation. But is this an achievable, or even desirable, goal for building mobile computing systems and applications? Let us look closely at the characteristics of the mobile computing environment and their implications in these regards.

1.2.2 Constraints of Mobile Computing Environments

Mobile computing has many constraints that distinguish a *mobile computing environment* (MCE) from the traditional desktop workstation/PC–based distributed computing environment. Notable among these are the following (Satyanarayanan, 1996):

1. *Mobile computers can be expected to be more resource-poor than their static counterparts*. With the continued rapid improvement of hardware technology, in accordance with Moore's law, it is almost certain that a laptop computer purchased today is more powerful that the desktop computer purchased just a year or even a few months ago. However, mobile computers require a source of electrical energy, which is usually provided by battery packs. Since batteries store a finite amount of energy, they need to be replaced or recharged. The first option costs money, and the second option, although cheaper in terms of money expended, requires plugging in the computer for recharging, restricting mobility. This has an impact on the design of mobile computers—all the hardware and software components in mobile computers are designed to reduce energy consumption and to increase the lifetime of the batteries. For example, processors on mobile computers are designed to consume less energy and, consequently, achieve a lower computation performance.

2. *Mobile computers are less secure and reliable*. Since mobile computers accompany their users everywhere, they are much more likely to be lost or stolen. Furthermore, they are more likely to be subjected to hostile or unfriendly use, e.g., a child throwing his daddy's PDA in a tantrum.

3. *Mobile connectivity can be highly variable in terms of it's performance (bandwidth and latency) and reliability*. Disconnections, both voluntary and involuntary, are common. The link bandwidth can vary by orders of magnitude over time and space.

Thus, in general, resource availability and quality vary dynamically.

The preceding characteristics of a mobile computing environment require rethinking about how mobile applications and systems should be designed. Resource paucity and the lower reliability of mobile devices point toward designing systems in such a manner that more reliance is placed on the static infrastructure. On the other hand, the possibility of disconnections and poor connectivity point toward making systems less reliant on the static infrastructure. Further, as mobile devices are moved around (or even if they are not), their situations keep changing over time. Mobile devices should change their behavior to be either more or less reliant on static infrastructure depending on current conditions. Figure 1.1 illustrates this need for dynamic adaptation in a mobile computing environment.

Insert Figure 1.1

1.2.3 Application-Aware Adaptation

Who should be responsible for adaptation—the application, the system, or both? There are two extreme approaches to designing adaptive systems: application-transparent (the system is fully responsible for adaptation) and laissez-faire (the system provides no support at all) (Satyanarayanan, 1996). Obviously, the laissez-faire approach is not desirable because it puts too much burden on the application developer. Further, no support from the underlying system restricts the types of adaptations that can be performed. However, as the following example points out, the application-transparent approach is not sufficient either. Consider two different multimedia applications. In one you are videoconferencing using a mobile device, and in the other you are watching a live video stream from a remote server on your mobile device. Now consider the following scenarios. In one, you move from an area with sufficient bandwidth for your application to an area where the amount of bandwidth is less than that needed by your application. In the other, your laptop's battery power level drops considerably. Both scenarios deal with changes in availability of resources. How would you like your system/application to behave under each scenario?

In the application (user)–transparent approach, the system/application may behave the same irrespective of the application running. However, different responses may be suitable depending on the type of application that is running. For example, in the first scenario, a nonadaptive system may just do nothing and let the audio/video quality drop. In the second scenario, the system may just give a warning to the user without any assistance on how to deal with the situation. In an adaptive system, various behaviors can be envisioned. For example, the system may try to do its best in both situations. However, the system's adaptation does not take into account the kind of application that is running. For example, in the first scenario, the system may try to adapt by requesting that the server or other peers start to send lower-quality video, in effect requiring lower bandwidth. In the second scenario, the system may try to conserve energy by reducing the intensity of the backlight of the display (besides warning the user of the lower battery power level). A still more adaptive approach is possible in which the system interacts with the user/application in deciding how to adapt.

In the application-transparent approach, the responsibility of adaptation lies solely with the underlying system. On the other hand, in the application-aware approach, the application collaborates with the underlying system software. Here the system provides status information about the available resources. The application uses this information to make decisions on how to adapt to changes in the resource availability. Each application can adapt in its own way. Figure 1.2 illustrates the spectrum of adaptation strategies that are possible (Satyanarayanan, 1996).

Insert Figure 1.2

1.3 Mechanisms for Adaptation

What can be adapted? As we will see in this section, both the functionality of various components in the mobile application and the data that are delivered to the application can be adapted. The next question is, “How to adapt?” In the context of the *client-server* (CS) model, functionality can be adapted by varying the partition of duties between the client and the server; e.g., during disconnection, a mobile client works autonomously, whereas during periods of strong connectivity, the client depends heavily on the fixed network, sparing its scarce local resources. Next we look at these approaches in more detail.

1.3.1 Adapting Functionality

The first approach is to change dynamically the functionality of the computational entities involved in response to the change in the operating conditions. An example of this approach is the extended CS model (Satyanarayanan, 1996). The CS paradigm is the most widely used architecture for distributed computing. In the standard CS model, the roles of the client and server are defined usually at design time, and these remain fixed during operation of the system (run time). Typically, a small number of servers provide some services such as access to databases, Web pages, allocation of temporary IP addresses, and name translation to a usually larger group of clients. A client, or the underlying system—middleware—may select dynamically the server from which to request the service. A server may or may not maintain information, or state, about the clients to which it is providing service. The state information may be maintained as soft or hard. Soft state information, once installed, has to be updated periodically to avoid automatic deletion by the state maintainer (in our case, a server), whereas hard state information, once installed, requires explicit deletion. Soft state is useful in systems with very dynamic configurations, such as mobile systems. This is so because soft state requires no explicit action to make the state information consistent with dynamic changes in the system. Soft state is used in various protocols, such as the Resource Reservation Protocol (RSVP) and the Internet Group Management Protocol (IGMP) to adapt gracefully to dynamic changes in the system state. Specifically, in the case of data servers (such as file servers), the CS model (as implemented by Coda) has the following characteristics (Satyanarayanan, 1996):

- A few trusted servers act as the permanent safe haven of the data.
- A large number of un-trusted clients can efficiently and securely access the data.
- Good performance is achieved by using techniques such as caching and prefetching.
- Security of data is ensured by employing end-to-end authentication and encrypted transmission.

  Developers of the Coda file system point to the following advantages of the CS

model: It provides good scalability, performance, and availability. “The CS model decomposes a large distributed system into a small nucleus that changes relatively slowly, and a much larger and more dynamic periphery of clients. From the perspective of security and system administration, the scale of the system appears to be that of the nucleus. From the perspective of performance and availability, a client receives service comparable to stand-alone service” (Satyanarayanan, 1996).

Impact of Mobility on the CS Model

The CS model permits a resource-poor mobile client to request a resource-rich server to perform expensive computations on its behalf. For example, the client can send a request to the server, go to sleep to conserve energy, and later wake up to obtain the result from the server. For the sake of improved performance and availability, the boundary between the clients and servers may have to be adjusted dynamically. This results in an *extended CS model*. In order to cope with the resource limitations of clients, certain operations that normally are performed at the client may have to be performed by resource-rich servers. Such lean clients with minimal functionality are termed as *thin clients*. Conversely, the need to cope with uncertain connectivity requires the clients sometimes to emulate the functions of the servers. This results in a short-term deviation from the classic CS model. However, from the long-term perspective of system administration and security, the roles of servers and clients remain unchanged.

1.3.2 Adapting Data

Another way to adapt to resource availability is by varying the quality of data (fidelity) that is made available to the application running on the mobile client. *Fidelity* is defined as the “degree to which a copy of data presented for use at the client matches the reference copy at the server” (Noble, 1997). This kind of adaptation is extremely useful in mobile information access applications. The QoS requirements for such applications are

- *Information quality.* “Ideally, a data item being accessed on a mobile client should be indistinguishable from that available to the application if it were to execute on the server storing the data.” (Noble, 1997).
- *Performance*
    - *From the mobile client’s perspective*. Latency of data access should be within tolerable limits.
    - *From the system’s perspective*. Throughput of the system should be maximized.

In general, it is difficult to provide both high-performance and highest-quality information in a mobile computing environment. In some cases, information quality can be traded for increased performance. The basic idea behind data adaptation is as follows: Assume that any data item accessed by a mobile client has a *reference copy* which is maintained at a remote server. The reference copy of a data item is assumed to be complete and up-to-date. At times when resources are plentiful, the mobile client directly accesses and manipulates the reference copy. However, whenever resources are scarce, the mobile client may choose to access or manipulate a data item of lower fidelity, and consequently, consuming fewer resources.

Fidelity and Agility

Data fidelity has many dimensions depending on its type. Consistency is a dimension which is shared by all data types. The other dimensions are type-dependent (Noble, 1997):

- *Video data*—frame rate and image quality
- *Spatial data such as topographic maps*—minimum feature size
- *Telemetry data*—sampling rate and timeliness

Various dimensions of data fidelity can be exploited for adapting to mobility. For example, a mobile client can choose to use the locally cached stale copy when it is disconnected from the server and a possibly more current copy when the server is accessible. Fidelity of data can be changed in several ways. This requires knowledge of data representation. For example, a video stream can be degraded by reducing the frame rate, reducing the quality of individual frames, or reducing the size of individual frames. Another point to note is that different applications using the same data may exploit different tradeoffs among dimensions of fidelity. For example, a video editor may choose to slow the frame rate, whereas a video player may choose to drop frames. When developing different strategies for trading off data fidelity dimensions against performance, an issue that arises is how to determine which strategy is better. Developers of the Odyssey system (a middleware for application-aware adaptation developed at Carnegie Mellon University) have evaluated their system using agility as a metric.

Agility is defined as the speed and accuracy with which an adaptive application detects and responds to changes in its computing environment, e.g., change in resource availability (Noble, 1997). The larger the change, the more important agility is. For example, an adaptive system that tries to adapt to the availability of connection bandwidth can try to determine how well the system reacts to sudden changes in bandwidth. One issue is how to model changes in the environment. The developers of Odyssey have used reference waveforms—step-up, step-down, impulse-up, and impulse-down—to model variation in wireless bandwidth availability. These waveforms were

generated using a trace modulation technique that emulates a slower target network over a faster wired local area network (LAN)—in this case, a wireless LAN over a faster wired LAN. In general, the results obtained from such studies should be interpreted by keeping in mind that an adaptation strategy is strictly better than another if it provides better fidelity with comparable performance or better performance with comparable fidelity. Further, the comparison must take into account the application's goals. The interested reader should refer to Noble (1997) for a detailed performance study of an adaptive system.

1.4 How to Develop or Incorporate Adaptations in Applications?

In general, it is difficult to enumerate all the mechanisms that can be employed to construct adaptive programs. However, it should be clear intuitively that all adaptive programs must adapt to some detectable *change* in their *environment.* Either a program can implement its own mechanisms to detect the changes, or mechanisms may be provided by some other entity—a middleware layer or operating system—to make the program aware of these external changes. In general, we can view these entities as software sensors (as opposed to hardware sensors, which we will come across later in this book). For example, a Transmission Control Protocol (TCP) client adapts its transmission window size by indirectly monitoring the congestion level in the network. Conceptually, it maintains a software timer for each packet sent, and as long as it receives an acknowledgment for a packet before its timer expires, it keeps increasing the size of its transmission window up to a maximum allowable window size. However, in the event of a loss—timeout or receipt of triple acknowledgment for a packet—it assumes that the loss is due to a buildup of congestion in the network, so it backs off by reducing its transmission window size. As a side note, this behavior is not suitable for wireless networks because the packet loss may be due to the high error rate in a wireless link on the delivery path or may be because an endpoint has moved. In such cases, the TCP client should not back off but instead should continue trying to push the packets through the network. Many techniques have been developed to "adapt" TCP for wireless networks, e.g., TCP-snoop (Balakrishnan, 1995).

In the state-based approach, changes in mobile computing are viewed as state transitions. Irrespective of how the state of the environment is sensed, the adaptation of function and/or data can be performed when a state transition occurs. Logically, each system state corresponds to an *environmental state.* Each system state is associated with some appropriate functionality. As long as the environment remains in a particular state, the system behaves according to the functionality associated with that state. When the environment's state changes, the system may have to perform some bookkeeping functions associated with state transition before assuming the functionality associated with the new state. In order to perform these operations, the system may have some

additional states.

For example, consider the functionality adaptation in the Coda (Continued data availability) distributed file system developed at Carnegie Mellon University. Coda is designed to maximize the availability of data at the expense of possible access to stale data. Each Coda client (called *Venus*) maintains a local cache. Venus adapts its functionality based on the state of the connectivity between the client and the server. Venus uses the following four states:

- *Hoarding.* Venus is in the hoarding state when it has *strong connectivity* with the server. In this state, the client aggressively prefetches files from the server to store locally. Files to be prefetched are decided on the basis of user preference and access pattern.
- *Emulating*. Venus is in the emulating state when it is *disconnected* from the server. In this state, the client emulates the server by optimistically allowing both read and write access to local files. To later update the primary copy of the files on the server and to detect any conflicting updates, the client maintains a log of all file operations.
- *Write-disconnected*. Venus is in the write-disconnected state when the client has *weak connectivity* to the server. In this state, a Coda client decides whether to fetch files from the server or to allow local access.
- *Reintegration.* Venus enters this state when the connectivity improves to *strong connectivity*. In this state, Venus resynchronizes its cache with the accessible servers. The log of operations is used for this purpose. If any conflict is detected, then user assistance may be required. On completion of resynchronization, Venus enters the hoarding state.

Note that the first three states correspond to some environmental state but that the last state corresponds to an environmental state transition. The state-transition diagram for Venus is shown in Fig. 1.3.

Insert Figure 1.3

1.4.1 Where Can Adaptations Be Performed?

In a distributed application, in particular, a CS application, the adaptation can be performed at the client, at the server, or at both the client and the server. Further, there are additional possibilities. The adaptation also can be performed within the network, say, at an intermediate software entity called a *proxy*. For example, consider a typical CS application. Several different adaptations may be performed on different components located at different points in the data and control path between the client and the server:

- *Adapting to the hardware/software capabilities of the mobile device*—in the proxy and/or at the server
- *Adapting to the connectivity of the mobile device*—at the server and/or the client
- *Adapting to the resource availability at the mobile device*—at the client

Let us look at some concrete examples to get a better understanding of incorporating adaptations in mobile applications.

Proxies

Proxies have been used by many applications to perform various tasks, such as filtering data and connections (e.g., security firewalls) and modifying control data [e.g., *network address translators* (NATs) change the IP fields]. Of particular interest to data adaptation are transcoding proxies. Transcoding is the process of converting data objects from one representation to another. Transcoding proxies can be used to adapt to various situations dynamically, such as the availability of bandwidth and capabilities of the end device. For example, if the end device is not capable of handling full-motion video, a transcoding proxy may convert it to a form that can be displayed on the end device (Han, 1998).

Conceptually, a transcoding proxy may be viewed as consisting of three modules: (1) an *adaptation-policy module* (2) a *data (content) analysis module*, and (3) a *content-transformation module*. Figure 1.4 shows the architecture of the transcoding proxy developed at the IBM T. J. Watson Research Center (Han, 1998). The adaptation-policy module can take as input information such as the server-to-proxy bandwidth (B_{sp}), proxy-to-client bandwidth (B_{pc}), client device capabilities (e.g., video display capabilities), user preferences, and content characteristics (provided by content-analysis module). Based on these inputs, it can decide on whether and how to modify the content. The content-transformation module performs the actual modification.

Figure 1.5 presents an example of logic that can be implemented in a transcoding policy module. The *transcoding threshold* is the document size beyond which transcoding becomes beneficial. To get an understanding of this parameter, consider the following simple cost-benefit analysis. Assume that the proxy uses the store-and-forward mechanism for delivering documents to the client. This is to say, the request from the client first is submitted to the proxy, the proxy then obtains the entire document from the server, and finally, it forwards the document to the client. Further, assume that the goal is to minimize the latency of document retrieval. Following the analysis in Han (1998), the total delay for document retrieval without the proxy (D_{sc}) and with the proxy (D_{spc}) is

$$D_{sc} = 2 \times \mathrm{RTT}_{\mathrm{pc}} + 2 \times \mathrm{RTT}_{\mathrm{sp}} + S/\min(B_{pc}, B_{sp})$$

$$D_{spc} = 2 \times \mathrm{RTT}_{\mathrm{pc}} + 2 \times \mathrm{RTT}_{\mathrm{sp}} + D_p(S) + S/B_{sp} + S_p(S)/B_{pc}$$

where RTT_{pc} is the round-trip delay between the proxy and the client, RTT_{sp} is the round-trip delay between the server and the proxy, S is the document size, $D_p(S)$ is a proxy delay function that relates the proxy processing delay to the document size, and $S_p(S)$ is an output size function that relates the transcoded document size to the input document size. Obviously, using transcoding is better when $D_{spc} < D_{sc}$. From this, a lower bound (*transcoding threshold*) on the input document size can be obtained to be

$$S > [D_p(S) + S_p(S)/B_{pc}]/(1/B_{pc} - 1/B_{sp}) = \text{transcoding threshold}$$

Note that the server-to-proxy bandwidth and the proxy-to-client bandwidth correspond with the bottleneck bandwidth along the data path from the server to the proxy and the proxy to the client, respectively. These bandwidths can vary dramatically as the user moves and with fluctuations in the available wireless bandwidth. Equation (1.1) can be used to adapt dynamically to changes in available bandwidth. However, for this to work, the transcoding proxy would need a good bandwidth estimator of the bandwidth that might be available in the near future.

Transcoding proxies also can work in the streamed mode. In this mode, the data stream is modified and passed on to the client as it is obtained from the server. For an analysis of when adaptive-streamed transcoding is beneficial and other details about transcoding proxies, see Han (1998).

WebExpress: An Example of Adaptation Using Proxies

Browsing over wireless networks can be expensive and slow owing to pay-per-minute charging (in cellular networks) and the characteristics of wireless communication. Additionally, the Hyper-Text Transport Protocol (HTTP) was not designed for wireless networks and suffers from various inefficiencies—connection overhead, redundant transmission of capabilities, and verbosity. WebExpress (Housel, 1998), developed by researchers at IBM, significantly reduces user cost and response time in wireless communication by intercepting the HTTP data stream and performing various optimizations on it. It is aimed at enabling routine commercial applications on mobile computers.

WebExpress uses proxies called *intercepts* that allow WebExpress to be used with any Web browser and any Web server. They enable WebExpress to intercept and control communications over the wireless link for the purpose of reducing traffic volume and optimizing the communication protocol, reducing data transmission. As shown in Fig. 1.6, the WebExpress architecture consists of two components that are inserted into the data path between the Web client and the Web server—*client-side intercept* (CSI), also known as *client-side proxy*, and *server-side intercept* (SSI), also known as *server-side proxy*. CSI is a process that runs in the end-user client mobile device, whereas SSI is

a process that runs within the wireline network. One of the features of this client-proxy-server model, also called the *intercept model*, is that the proxies are transparent to both Web browsers and servers. This makes this adaptation technique insensitive to the evolution of technology. This is a very important advantage because HTML/HTTP technology was (and still is) maturing rapidly when WebExpress was developed. Another advantage is the highly effective data reduction and protocol optimization without limiting browser functionality or interoperability. WebExpress employs several optimization techniques such as *caching*, *differencing*, *protocol reduction*, and *header reduction*.

- *Caching*. WebExpress supports both client and server caching using the *least recently used* (LRU) algorithm. Cached objects persist across browser sessions. Caching reduces the volume of application data transmitted over the wireless link through cross-browser sessions. We will look at the details of this in Chap. 3.

- *Differencing*. Caching techniques do not help in common graphic interface (CGI) processing, where each request returns a different result, e.g., a stock-quote server. However, different replies from the same program (application server) are usually very similar. For example, replies from a stock-quote server contain lots of unchanging data such as graphics. For each dynamic response from a CGI (HTML file) cached at the SSI, the SSI computes a base object for the page before sending it to the CSI. If the SSI receives a response from the CGI server and the cyclic redundancy check (CRC) received does not match the CRC of the base object, the SSI returns both the difference stream and the base object. This is called a *basing operation* in WebExpress parlance. *Rebasing* is carried out in the same fashion when the SSI detects that the difference stream has grown beyond a certain threshold.

- *Protocol reduction*. Repeated TCP/IP connections and redundant header transmissions present additional overhead. The WebExpress system uses two techniques to reduce this overhead and optimize browsing in a wireless environment:

    - *Reduction of TCP/IP connection overhead using virtual sockets.* WebExpress establishes a single TCP/IP connection between the CSI and the SSI. The CSI sends requests over this connection. The SSI establishes a connection with the destination server and forwards the request. Thus overhead is incurred between the SSI and the Web server but not over the wireless link. *Virtual sockets* are used to provide multiplexing support. Virtual sockets are implemented in the following manner: Data sent are prefixed by a virtual socket ID, command byte, and a length field. At the CSI, the virtual socket ID is associated with a real socket at the browser. At the SSI, the virtual socket ID is mapped to a socket connection at an HTTP server. This mechanism permits efficient transport of HTTP requests and responses while maintaining protocol transparency.

- *Reduction of HTTP headers.* HTTP request headers containing lists of Multipurpose Internet Mail Extensions (MIME) content types can be hundreds of bytes in length. The CSI allows this information to flow in the first request and then saves the list. On subsequent requests, the CSI compares the received list with the saved one. If the two lists match, the MIME content-type list is deleted from request. The SSI inserts the saved one if none is present.

The transparent proxy–based architecture of WebExpress allows the operation of commercial Web applications on wireless networks. Differencing and virtual sockets offer the most critical optimizations in the WebExpress system.

1.5 Support for Building Adaptive Mobile Applications

Adaptations should be customized to the needs of individual applications. We have argued that applications are in a better position to perform application-specific adaptations than the operating system alone. However, what does this mean with regard to where adaptations can be performed? Applications not only make local adjustments, but they also should collaborate with other adaptation technologies that are available in other components of the system. For example, in the CS scenario, both the client and the server may need to adapt. The advantage of application-aware adaptation is that the application writer knows best how the application should adapt. However, does the application writer know the underlying system as well as the application? Furthermore, application-aware adaptation tends to work only at the client or perhaps at the server. If this is the only mechanism for adaptation, without any monitoring on resource usage by each application, this may result in selfish behavior by the applications. In the following sub-sections we look at details of some current efforts to developing adaptive applications. Here, we focus only on the adaptation mechanisms of these systems. We will re-examine some of these systems later in this book from the perspective of middleware services they provide to mobile applications.

1.5.1 Odyssey

Odyssey (Noble, 1997) aims to provide high fidelity and to support concurrent mobile applications with agility. It emphasizes collaboration between applications and the operating system in performing adaptation to handle constraints of the mobile computing environment, especially those imposed by the presence of wireless links.

The following case is a motivating example from the Odyssey group. Imagine a user with a lightweight/wearable mobile computer with ubiquitous wireless access to remote services, an unobtrusive heads-up display, a microphone and earphones, and

speech recognition for computer interaction with online language translation. The user has ubiquitous connectivity, for example, owing to an overlay network, but the quality varies as he moves and as different networks are accessed. The user simultaneously gets voice, video, and other data sent to him. When the user moves to a relatively shadowed area and the network bandwidth drops dramatically, Odyssey informs the video, audio, and other applications of these changes, allowing them to make the proper adaptations in their network usage and their behavior. Why use application-aware adaptation here? The presumption is that for this environment, only the application knows what to do because the operating system does not have the application-level knowledge. If the operating system makes the decision, it may do the wrong thing. However, the operating system must be involved to ensure fairness among competing applications. In essence, the basic Odyssey adaptation model is as follows: The operating system support on the portable machine monitors network conditions. Each application interacts with the operating system tools to negotiate services. When network conditions change, the operating system notifies the applications of what has happened.

As shown in Fig. 1.7, the Odyssey architecture consists of two main components—*viceroy* and *warden*. Viceroy performs centralized resource management and monitors the availability of resources, notifying applications of changes. Wardens provide data-type-specific operations, namely, `tsop()` functions, to change the fidelity. A `tsop()` function is similar to the transcoding proxies we saw earlier but specific to a data type. Wardens are also responsible for communicating with servers and for caching data.

In the Odyssey application-adaptation model, applications do not interact directly with their remote servers. Applications talk to their wardens, and wardens talk to the servers. Applications tend to have limited roles in actually adapting transmissions. They may know about different formats and tolerances and accept data in their different adapted versions.

Applications interact with Odyssey to adapt their behavior. All data to and from the server flow through Odyssey. Applications must register their preferences and needs with Odyssey in the form of requests. A request specifies that an application needs a particular resource within certain limits, e.g., between 100 kbps and 1 Mbps of bandwidth. If the request can be satisfied currently, it is. If things change later, the application is notified using an `upcall` to the applications. In response, the application can adjust itself and make another request. Note that `upcalls` can occur because resource availability became either worse or better.

The following example illustrates a typical interaction between an application and Odyssey. A video application requests enough bandwidth to receive a color video stream at 20 frames per second (fps). Odyssey responds, “No way, try again.” The application retries and requests bandwidth sufficient for 10 fps in black and white, specifying the minimum and the maximum needed for this application so that it can improve the quality if it is worthwhile. The channel subsequently gets noisy, and the bandwidth drops.

Odyssey performs an `upcall` informing the application that the bandwidth is outside the specified limits. The application requests a lower bandwidth suitable for a lower fps rate.

Odyssey wardens mediate server-application interaction. A warden is a data-type-specific module capable of performing various adaptations on that data type. Additionally, wardens do caching and prefetching. If Odyssey needs to handle a new data type, not only does the application need to be altered, but a new warden also has to be written. The better the warden understands the data type, the better is the potential adaptivity. The Odyssey viceroy is the central controlling facility that handles sharing of resources. The viceroy notices changes in resource conditions. If these changes exceed preset limits, the viceroy informs the affected applications using the `upcall` mechanism. Developers of Odyssey have evaluated their system to answer the following questions:

How agile is Odyssey in the face of the 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?

Interested readers should refer to Noble (1997) for details.

1.5.2 Rover

Rover is an object-based software toolkit for developing both *mobility-aware* and *mobility-transparent* CS distributed applications (Joseph, 1995, 1997). It provides application developers with two programming and communication abstractions specifically designed for assisting applications in harsh network environments such as mobile computing—*relocatable dynamic objects* (RDOs) and *queued remote procedure calls* (QRPCs). RDOs can be used to reduce interaction between two weakly connected entities, such as a client on the mobile device and a server in the wireline network. Rover RDOs are objects with well-defined interfaces and are loadable dynamically from the server to the client. This, in essence, moves objects to the client machine, avoiding the client having to communicate with the object at the server. QRPCs can be used to handle disconnections. Rover QRPCs are essentially nonblocking *remote procedure calls* (RPCs) that support *split-phase operations*. That is, they allow an application to make an RPC without worrying about whether the destination is currently reachable. If the destination of the RPC is not reachable at the time of the call, the call is queued. On reconnection to the RPC's destination, the RPC is performed. The result of the RPC is delivered asynchronously to the application. Figure 1.8 illustrates the various components in the distributed CS object system of the Rover toolkit and the control flow within the toolkit.

Insert Figure 1.8

- How does one use Rover? By writing and perhaps rewriting the application using the toolkit—invoking these tools when network connectivity is poor. However, this requires a good understanding of network programming and user mobility. To use RDOs, applications—usually clients—import RDOs from the server. They then invoke methods on imported RDOs. When finished, RDOs are exported back to the server. If needed, Rover also can cache copies of objects. This allows clients to use the cached copy instead of fetching the original from the server. Updates to objects are handled by an optimistic CS replication method. RPCs are queued only at the client, where they are stored until the client can handle them. When the appropriate level of connectivity is established, Rover clears the queue intelligently using an RPC prioritization mechanism. It also can batch related requests. If the client is not available when the response comes back, the server drops the response. The rationale is that queued requests at the client eventually will be replayed. This may cause some inefficiency, but it simplifies application design. A mobile application can employ QRPCs in various situations dynamically to optimize cost to the user or the performance of the application (Reiher, 1998):*Optimize use of expensive links*. Consider a situation in which the mobile user pays for wireless connectivity based on the duration of usage, e.g., pay-per-minute plans for cellular phones. To optimize the monetary cost to the mobile user, QRPC can batch several requests and then disconnect from the network after invoking all the batched QRPC calls in a single connection.

- *Make use of asymmetric links*. The queued RPC requests are not associated with a particular network interface. Thus responses can be obtained over any network device. This permits, for example, an application to launch requests over expensive links and to receive responses over cheaper links. This is beneficial for situations where the request size is much smaller than the expected response size (e.g., in Web browsing) and when the response can be used incrementally, as it arrives.

- *Stage messages near their destination*. RPC queuing can be arranged to occur just before a “bad” link. If the link quality improves, the RPC queue will then be cleared out. Meanwhile, the transmitter does not get blocked on the bad link.

.

1.6 Summary

In this chapter we discussed the limitations of mobile computing environments. In order to cope with these limitations, mobile applications have to be adaptive. Adaptations can

be performed either by changing functionality or by changing the data provided to the application. The kinds of applications we considered in this chapter mostly fall under the domain of mobile information access. Many of these applications involve downloading different kinds of information. For example, Web browsing involves downloading HTML multimedia documents, database access involves accessing different kinds of databases, and video streaming applications involve downloading of continuous video and audio streams. We saw what kind of adaptations can be performed to enable these applications to continue working in mobile computing environments. Furthermore, many of these applications are based on the CS model. In order to enable adaptation, this model is extended.

Once we established the need for adaptation and the mechanisms that can be used for performing adaptations, we looked at who should be responsible for it—the system or the application itself. Approaches to developing adaptive applications can be categorized as completely internal to the applications, layered outside the application, using special operating system features and libraries, and interacting with other mechanisms such as intelligent use of proxies. We studied various approaches to incorporating adaptation in an application, such as using ad hoc methods to achieve the needs of the application, using tools specific to applications (e.g., Web browser proxy method), and using general adaptation tools such as toolkits and operating system features. However, in the current state of the art, most mechanisms only work at the client side, some work at the server, and a few help with anything in between. The main question for future work in this direction is, "How can an adaptive application obtain finer control over the entire path between its distributed components?" (Reiher, 1998).

References

Balakrishnan, H., S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks,"*ACM Wireless Networks* 1(4):469, 1995.

Han, R., P. Bhagwat, R. LaMaire, et al., "Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing Personal Communications," *IEEE* Personal Communications 5(6):8, 1998 (see also *IEEE Wireless Communications*).

Housel, B., G. Samaras, and D. Lindquist, "WebExpress: A Client/Intercept Based System for Optimizing Web Browsing in a Wireless Environment," *Mobile Networks and Applications* 3:419, 1998.

Joseph, A., A. F. deLapinasse, J. A. Tauber, et al., "Rover: A Toolkit for Mobile Information Access," Proceedings of the fifteenth ACM symposium on Operating Systems Principles SOSP'95, Cooper Mountain, Colorado, December 1995.

Joseph, A., A. F. deLapinasse, J. A. Tauber, et al., “Mobile Computing with Rover Toolkit,” *IEEE Transactions on Computers* 46(3):337, 1997.

Noble, B., M. Satyanarayanan, D. Narayanan, et al., “Agile Application-Aware Adaptation for Mobility,” Proceedings of the sixteenth ACM symposium on Operating Systems Principles SOSP’97, December 1997.

Reiher, P.., Lecture notes CS239 Hot Topics in Operating Ssytems, UCLA, 1998.

Satyanarayanan, M., “Fundamental Challenges in Mobile Computing,” *PODC* XX:XX, 1996.

Satyanarayanan, M., “Mobile Information Access,” *IEEE Personal Communications* 3(1):XX, 1996.

Gouda, M. G., and T. Herman, “Adaptive Programming,” *IEEE Transactions on Software Engineering* 17:911, 1991.

Noble, B., M. Price, and M. Satyanarayanan, “A Programming Interface for Application-Aware Adaptation in Mobile Computing,” in *Proceedings of the 1995 USENIX Symposium on Mobile and Location-Independent Computing.* Ann Arbor, MI, April 10-11, 1995.

Pitoura, E., and G. Samara, *Data Mangement for Mobile Computing.* New York: Kluwer

Academic Publishers, 1998.

Zenel, B., and D. Duchamp, “General Purpose Proxies: Solved and Unsolved Problems,” in *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems.* Cape Cod, MA, May 5-6, 1997, p. 87.