System Support for Mobility

Andrew Black and Jon Inouye

Department of Computer Science and Engineering
Oregon Graduate Institute of Science & Technology
P.O. Box 91000, Portland, OR. U.S.A.
Email: {black, jinouye}@cse.ogi.edu

1. Introduction

As we move closer to world-wide networking and the merger of computing and communications, truly mobile computing—computing while on the move—will become as unexceptional as truly mobile communication (using cellular telephones) is today. However, mobile computing poses new challenges for both system and application designers. Unlike telephony, where a new communications and terminal infrastructure was created to provide a well-understood service (peer-to-peer voice communication), and inter-operability was maintained at the central office, to realize the vision of mobile computing we need to adapt existing applications, and inter-operability must be provided for between peer applications and systems.

Almost all of the computer applications that have been built to date make the assumption that their environment is invariant over a significant period of time. Previous generations of bespoke applications were *designed* around assumptions of computer speed, memory, network bandwidth, communications latency and user interface; changes in these assumptions may have necessitated a new design. Current mass-market ("shrink-wrapped") applications more commonly interrogate their environment when they are initialized, but then assume that nothing changes until after they terminate. Even a modest dynamic change, such as reducing the bit-depth of the available display (as might occur when a laptop is unplugged from a stationary monitor), is more than many applications can handle. Similarly, while operating systems are now often built to ascertain at boot time which network devices are available, and to initialize the appropriate drivers, they are typically unable to handle dynamic changes in these same attributes.

The challenge of mobile computing is that *many* attributes of the application environment vary as the computer is moved from place to place. Moreover, the degree of variability is enormous. For example, available network bandwidth may vary by five orders of magnitude. Adapting to this variability is the shared responsibility of the application program and the operating system. We do not believe that there is a "silver bullet" that will automatically make all applications mobile. But we do believe that the use of the Synthetix specialization methodology will both make the problem tractable and will provide a framework for dividing between the application and the system the responsibility for adapting to change.

The remainder of this position paper will describe in more detail the kind of variability that mobile applications will often encounter, and will then outline how we envisage adapting applications and systems to accommodate it.

2. Variability caused by Mobility

From the point of view of the network engineer, a mobile computer may experience change in geography, in network topology, and in interconnection medium. Although at first sight we might feel that it is the responsibility of the operating system to hide these changes from the application, we do not believe that this will be possible in general. Each low-level change can give rise to changes at higher levels of abstraction that cannot be hidden effectively. Instead, there must be a mechanism for initiating application-level adaptation. To understand this better, let us look at some of the assumptions that may be invalid in a mobile environment.

2.1. Bandwidth and Connectivity

The most obvious characteristic of mobility is that, while mobile, a node may become disconnected from the network. If connectivity is maintained, the available bandwidth may change by three or four orders of magnitude. This is because of fundamental physical limits that apply to wireless communication; it is not a problem that will yield to a "technological fix".

2.2. Physical Link Invariance

Network stacks within an operating system assume that the set of network interfaces available on a node is invariant. However, standards like PCMCIA [5] now enable interfaces to be inserted and removed while the computer is running. It ought to be possible to launch an application while connected to a wired network, unplug the wired network interface and replace it by an infra-red network card, and carry the computer to another room while the application continues to work.

2.3. Security domain

Applications sometimes make security decisions based on network topology. For example, if a local subnetwork is considered secure, a remote login program might elect not to use encryption between nodes within the same IP subnet. However, with the advent of Mobile IP [7], a host can retain its IP address as it is detached from a secure subnetwork and reconnected to a remote subnet, even though packets must now traverse an insecure public network to reach it. Switching from a wired interface to a wireless interface may also have security implications, both because it is easier to eavesdrop on the wireless medium and because the change in network may cause packets to be routed differently through the wired network.

2.4. Network Assumptions

Wired networks have low error rates; as a consequence, when packet loss occurs it is usually caused by *congestion*, that is, overflow of a buffer somewhere in a network. As with all overload conditions, the correct response to congestion is to shed load. TCP relies on the end-nodes to shed load; there is no admission control in the network itself. "Slow start" and congestion control polices were added to TCP to allow it to exploit high bandwidth connections without overloading lower bandwidth links.

However, on wireless networks, where packet loss is far more frequently caused by corruption than by buffer overflow, these policies limit the ability of TCP to exploit the available bandwidth. If a packet is lost because of a burst of radio-frequency noise, the best response is to re-transmit it as soon as possible. Instead, because of the assumption that packet-loss is caused by congestion, TCP will back-off and perform a new "slow start". Within the current network architecture, selecting the appropriate retransmission policy requires the cause of the loss to propagate to the end-node.

This model of a dumb network with intelligent end-nodes has served us quite well since the advent of the Internet. End-to-end control of congestion and reliability is a logical result of that model. (Note that telephony has adopted the contrary model: dumb end-nodes connected by an intelligent network.)

It is unclear whether the end-to-end networking model can survive as networks become increasingly heterogeneous. In networks of the future, packets will be lost from a variety of causes; dealing with these losses correctly requires the presence of information on the cause of the loss. It may be more reasonable to take the appropriate action at the link-level, where the required information is available, than to propagate this information to the end-nodes at higher layers.

Similarly, the simple solution to the problem of insecurity of wireless links is to use link-layer encryption and authentication on those links. But this would require major changes to the network infrastructure. Hence, current proposals are to change the end-node software. The proposed IP security mechanisms [1] provide security at the network level; PGP [10] and SSH [9] do so at the application level.

3. Supporting Mobile Applications

There are two approaches to supporting mobility: we can modify our applications, or we can modify our operating systems. We believe that in general it will be necessary to do both.

POP-based mailers like *Eudora* [8] illustrate one way in which applications can accommodate mobility. Previous SMTP-based mail systems relied on mail being "pushed" out to its destination; this worked well so long as the destination was permanently connected. The Post Office Protocol (POP) [6] enables a mail agent on a mobile host to "pull" mail in when required; moreover, the mobile host can switch network addresses between POP sessions without disrupting the delivery of mail.

The Macintosh anonymous FTP client *Anarchie* [4] provides an interesting example of application level adaptation. Whereas most FTP clients will fall over if the underlying TCP connection is broken, Anarchie treats connections as resources that can be cached. In fact, Anarchie aggressively times-out idle connections to limit the load that it imposes on FTP servers. Even after a connection is broken, the directory listings obtained over it remain on the screen. Double-clicking on a filename in such a listing will open a new connection (if necessary) and fetch the appropriate file. Such an application would be relatively unaffected by changes in the client machine's IP address.

However, even for such applications, some system support is necessary. For example, many TCP/IP implementations will continue to transmit packets even when no interface is available, as when a PCMCIA Ethernet card is unplugged, or the telephone is hung up on a PPP connection. As a consequence, applications must "code between the lines" in order to detect network disconnections: notification of such events should be a system service.

Other applications simply cannot be made mobile without system support. In contrast to the FTP client, with our current name service infrastructure an FTP server must have a fixed IP address. This requires modification of the operating system or network. Once the work is done at the system level, the same functionality is available to other applications without duplication of effort. Moreover, system solutions are often more efficient than application-level solutions if protection or communication with other kernel entities is required.

Nevertheless, it will often be impossible to hide the effects of mobility completely. For example, Mobile IP allows a mobile node to maintain connections transparently while migrating between networks. But applications that make the assumption that the bandwidth or security domain will remain invariant for the lifetime of the connection will be sadly disappointed.

Thus, we come to the conclusion that the system and the application must collaborate, which means that the application will need to be notified when some of its assumptions have become invalid. This in turn means that the operating system must be informed of the nature of the application's assumptions.

4. The Synthetix Methodology

Our methodology provides a structured way to evolve both applications and operating systems. The first step is to make explicit an application's assumptions about what remains invariant, and to communicate these invariants to the operating system.The second step is to guard the now-explicit invariants with code that is triggered when they are violated. The third step is to make the system or the application react appropriately to the triggered guards.

Making assumptions explicit differs from traditional forms of specification that require the statement of an explicit precondition. In our methodology, it may well be the case that the application can continue to operate even though its invariants are violated—*provided that it is so informed.* Some mechanism is required to pass invariants from the application to the operating system; in the terminology of Open Implementation [3] this is a meta-interface, since it affects the behavior of the ordinary (service) interface. For example, a real-time multimedia application could pass its invariants though a quality of service (QoS) specification.

Guarding the invariants may be simple or more complex. If the application's invariants map directly onto system parameters, then finding the places in the system where they may be invalidated is not hard. For example, if the application assumes a wired Ethernet connection, and such a connection is available when the application is initialized, then we must place guards in the code that detects the disconnection of an interface. Compile-time tools that we are developing for performance specialization [2] can help to place guards of this kind. However, it may be that the application's invariants are at a higher level of abstraction; examples might be requiring a bandwidth of 5 Mb/s or a secure transmission channel. These invariants can be violated even though the physical network interface does not change; for example, the remote node might move. Guarding such invariants might require active traffic monitoring, rather than the simple insertion of tests.

Communicating the violation of an invariant may require nothing more than a procedure call (if the dependent code is within the system itself), or it may require that an event be raised in the application. The amount of restructuring that the application must perform when an invariant is violated can vary widely. For example, when an application like *Anarchie* is notified that its node address has changed, all it need do is flush its connection cache and continue. In contrast, when a distributed MPEG player is notified of a 99% decrease in bandwidth, it must radically adjust the resolution, depth, and frame rate according to some QoS agreement made with the user.

5. Summary

Support for mobility is an important component of support for world-wide applications. Because of fundamental physical limitations on wireless networking, the gap between wired and wireless networks will widen rather than narrow. Accommodating this divergence of functionality will require changes to both applications and systems. We have proposed a systematic methodology that helps us characterize the ways in which an application depends on system-wide invariants, which will often be properties derived from those with which the operating system is directly involved. If the invariants are violated the application must be informed; it is the job of the application programmer to determine the appropriate response.

6. References

[1] Atkinson, R., "Security Architecture for the Internet Protocol," RFC 1825, August 1995. (ftp://ds.internic.net/rfc/rfc1825.txt)

[2] Consel, C. and Noël, F. "A General Approach for Run-Time Specialization and its Application to C," In *Proceedings of the ACM Symposium on Principles of Programming Languages,* St. Petersburg Beach, Florida, January 1996, pp. 145–156. (http://www.irisa.fr/EXTERNE/projet/lande/consel/)

[3] Kiczales, G. "Towards a new model of abstraction in Software Engineering," In *Proceedings of the IMSA '92 Workshop on Reflection and Meta-level Architectures,* November 1992, pp. 1–11. (The Open Implementation web page is located at http://www.parc.xerox.com/spl/projects/oi)

[4] Lewis, P. Anarchie 1.60. (http://www.share.com/peterlewis/anarchie/anarchie.html)

[5] Mori, M. T., and Welder, W. D. *The PCMCIA Developer's Guide,* second edition, Sycard Technology, 1995. (PCMCIA Home page located at http://www.pc-card.com)

[6] Myers, J., and Rose, M. "Post Office Protocol – Version 3," RFC 1725, November 1994. (ftp://ds.internic.net/rfc/rfc1725.txt)

[7] Perkins, C. "IP Mobility Support," Internet Draft, April 1996. (ftp://ds.internic.net/internet-drafts/draft-ietf-mobileip-protocol-16.txt)

[8] Qualcomm Inc. Eudora Division. (http://www.eudora.com)

[9] Ylonen, T. "SSH Transport Layer Protocol," Internet Draft, June 1996. (ftp://ds.internic.net/internet-drafts/draft-ietf-tls-ssh-00.txt — Secure Shell home page is located at http://www.cs.hut.fi/ssh)

[10] Zimmermann, P. R. *The Official PGP User's Guide,* MIT Press, 1995. (http://www-mitpress.mit.edu/mitp/recent-books/comp/pgp-user.html)