A Hidden Proxy for Seamless & ABC Multimedia Mobile Blogging

Vittorio Ghini, Stefano Cacciaguerra, Fabio Panzieri, Paola Salomoni
Università di Bologna

Computer Science Department, Mura Anteo Zamboni 7, 40127 Bologna, Italia
{ghini, scacciag, panzieri, salomoni}@cs.unibo.it

Abstract

The availability of smart portable telephones, equipped with cameras and microphones, enables portable phone users to capture fragments of audio and video, taken directly from their daily lives. Thus, a new generation of mobile blogs, termed "moBlogs", is emerging. Conceptually similar to textual Web logs, moBlogs collect text, images, audio and video, captured by mobile devices, and make them available on-line. In addition, the availability of different connectivity options on board the same portable phone (from Bluetooth, through 2.5/3G to WLAN) can improve the ability of the phone users to browse and update the moBlogs of interest to them. Within this scenario, in order to simplify the development of mobile applications for smart portable phones, and to enhance their reusability, it is necessary to provide those applications with appropriate communication support that shields them from the heterogeneities of the multiple connectivity options mentioned above. In this sense, the paper presents an application-transparent middleware designed and developed to provide mobile applications with end-to-end, Always Best Connected (ABC), and seamless connectivity, even in case of handoff events and network interface reconfigurations.

1. Introduction

Blogging has gained a great momentum over the Internet, enabling millions of users to share their lives and thoughts. Currently, blogs are substantially based on textual information, created and managed by means of so-called blog applications through a wired PC. However, a new generation of blogs is emerging, prompting the possibility to share information by following people throughout their daily activities, and offering support to multimedia data [1]. These blogs are frequently collective (*friendBlogs*), and created by

applications running on mobile devices (*moBlogs*). Industries are investing in this emerging "life caching" trend, and offer different software technologies in order to meet the blog communities demand (e.g., *Nokia Lifeblog* [2], *Microsoft Sensecam Project* [3]). Similarly to *cyborgLog*, these technologies carry out a first-person recording of an activity, which works as a sort of "memory prosthesis".

The widespread use of interconnected portable devices (camera phones, media players with built-in microphones, portable storage devices) is fueling this phenomenon and increasing the number of applications that provide support to multimedia mobile blogs. These applications run sequences of upload and download operations from the device to the server (updating) and vice versa (browsing) that are usually based on 2.5/3 G networks. However, many devices used by mobile bloggers integrate, or simply support, different types of both wired and wireless connections; thus, these devices can provide their users with seamless ABC connectivity. Seamless connectivity is a popular paradigm for wireless networking that includes different solutions to continuously and transparently switch connectivity through different available network interfaces. Seamless systems dynamically change the network interface being used, either to maintain service continuity (i.e. to overcome handoff or link outages), or to select the best available connection (i.e., to provide ABC communication services) [4]. Connection continuity and quality are essential in mobile *videoBlogs*, which use video as their primary presentation format.

The wide plethora of devices, standards, applications used to offer on-line mobile multimedia blogging suggests the use of a middleware to offer ABC and seamless connectivity. Moreover, in order to simplify the application development and to enable application reusability, it is desirable that this middleware operates transparently to the applications. Owing to these observations, we have developed an end-to-end architecture, named Hidden Network

Improvement Framework (HiNIF), which increases the reliability and throughput of the TCP communications on mobile multihomed systems, and supports the development of seamless, mobile, ABC services, such as mobile blogs. In particular, our middleware architecture is designed to: (i) guarantee communication recovery after temporary wireless/wired link failure; (ii) monitor the system to support network interface reconfiguration and maintain the TCP communication; (iii) select and use the *best* connection when more than one network is available [5], and (iv) provide this support transparently to the applications that use it. In this paper we introduce the HiNIF architecture, and discuss an experimental evaluation of its implementation we have carried out.

This paper is structured as follows. Section 2 summarizes TCP communication issues in mobile multihomed systems. Section 3 introduces the HiNIF architecture; Section 4 discusses its experimental evaluation. Finally, Section 5 concludes the paper.

2. TCP Communication Issues in Mobile Multihomed Hosts

TCP communication services are made available to the applications via the socket API interface. In mobile wireless communication environments, these services may become unavailable to the applications, as a number of communication failures may lead the active TCP connections in a host to fail, possibly causing abnormal termination of the socket API primitives, and raising application level errors. In particular, a handoff disconnection occurs when a mobile host loses the communication between its network wireless card and the network access point; i.e. when a link becomes unreachable (as the host has moved out of the access point range, for example). A different type of failure may be due to an interface being disabled; this may occur when the OS detects a datalink level error in a network interface, and disables it temporarily (i.e. the IP address of that interface becomes invalid). More generally, IP address loss can be caused either by an interface disablement event, such as that mentioned above, or by a network interface reconfiguration. This latter event can occur when (i) a mobile host moves to a new physical subnet, thus requiring a new IP address, or (ii) when the host user selects a network provider whose DHCP server assigns a new IP address to the user host. An IP address loss produces at least two serious consequences on a mobile host. First, the host cannot be reached by the other end-systems. Second, the local transport address (i.e., the pair <IP address; port number>) of any active TCP connection in that host becomes invalid; hence, the host OS aborts those

TCP connections and the socket API primitives return an error to the application.

Further issues arise in the design of communication support, whether the mobile host is multihomed [5] – as current OSs for mobile hosts do not provide sufficient support for managing (i) the *binding* of TCP connections to the local IP address, and (ii) the IP datagram *routing* service – or when dealing with multiple, heterogeneous network interfaces. We term *Local Binding* the action performed by the clients' OS kernel at TCP connection setup time, by associating a TCP connection identifier to the local IP address of the network interface that will be used to transmit the TCP segments. This address becomes the *local IP address* of the TCP connection. The binding mechanism is crucial to TCP based communications, as after the connection has been setup the server uses the local IP address to reach the client host. If that address changes due to a reconfiguration of the network interface, the client host becomes unreachable, and the kernel disables the TCP socket. The IP datagram routing is the operation a host performs when transmitting each IP packet to select an output network interface to send the packet to. Typically, a mobile multihomed host maintains a very simple routing table that enables it to take forwarding decisions based on the IP packet destination address only. Usually (as in IPv4), the routing table features a *default* interface. A host transmits an IP packet through its default interface when the destination address of that packet does not belong to one of the subnets directly connected to it. Unfortunately, the destination becomes unreachable when the default interface link is temporarily unavailable (even if the other interfaces are properly functioning).

3. Architecture

To overcome TCP communication problems in mobile scenarios, the HiNIF architecture operates on both Client/Server end systems, as illustrated in Figure 1. The HiNIF architecture is transparent to the applications it supports, so that these applications are unaware of HiNIF, and communicate using the standard TCP socket primitives. The HiNIF architecture intercepts the application connection requests, and splits each TCP connection (represented by the dashed line) between a Client and a Server into three separate TCP connections (represented by the three continuous arrows). Each Local TCP Connection enables the communication between the local application and its relative local HiNIF instance. The third connection enables the communications between the two end systems across the network. The two local

TCP connections are protected from possible network failures, while the TCP connection between the two end-systems (the enhanced connection) is managed by two HiNIF components called Communication Managers (described later), at the end-points of this connection. These Managers can use their own special purpose protocols over an established TCP connection to optimize the use of available network interfaces, and to detect, and recover from, possible network failures (yet again, transparently to the application).

Figure 1. Splitting the TCP Connection.

The internal structure of the HiNIF architecture, as illustrated in Figure 2, consists of the following four principal components: the *Interceptor*, the *Virtual Network Interface*, the *Enhancement Detector* and the *Communication Managers*.

In summary, the *Interceptor* component implements the basic mechanism that splits the TCP communications into three connections, and allows connection failure management and recovery in an application-transparent way. It consists of an *Outgoing Flows Selector* (OFS) module, and an *Incoming Flow Marker* (IFM) module. The former module intercepts the TCP segments originated from the local application, and redirects them to the Communication Manager (note that this module ignores all non-TCP traffic, which is thus transmitted to its destination, unaffected by HiNIF). The latter module intercepts the incoming TCP segments and delivers them to the application.

The *Virtual Network Interface* (VNI) component, namely a tun/tap interface, maintains the abstraction of a virtual Ethernet interface, (i.e. an Ethernet interface not linked to a physical one). HiNIF associates to the VNI component an IP address which is not visible to the remote hosts. Using a particular routing table configuration, HiNIF causes the binding mechanism to select that IP address as the local address for the outgoing connections, in particular the Local TCP Connections. As these connections are bound to the VNI, they are independent from the physical interfaces (and their faults). Thus, the VNI cannot be affected by the above-mentioned IP address loss problem, and

local connections can continue working even if a network failure occurs. After the binding is completed, the *Interceptor* redirects the TCP segments transmitted over the Local Connection to the Communication Manager, which will be responsible for the communications with the remote peer application.

The *Enhancement Detector* (ED) implements the protocol that enables the HiNIF Communication Manager to detect the presence of a peer HiNIF Communication Manager at the other end of a connection, at connection establishment time (and hence, whether it can set up what we have termed above an enhanced connection). According to this protocol, the ED at the host end requesting the connection establishment sets the URG flag in the TCP header of the first two segments that are transmitted for connection establishment purposes. If it receives an acknowledgement segment containing the URG flag set, a connection has been established with a peer HiNIF instance (rather than with an arbitrary TCP protocol server, for example). The EDs in the client and server HiNIF instances at the end points of that connection notify their local Communication Manager that an enhanced connection has been established. If the acknowledgement segment does not contain the URG flag set, the Communication Manager will act as a simple TCP proxy. Thus, the EDs enable the simultaneous management of both enhanced and conventional TCP connections.

Figure 2. The HiNIF Architecture.

Finally, the *Communication Managers* (CMs) are responsible for the enhanced TCP connection management, and the provision of reliable data transport between the two systems at the end points of an enhanced connection. Specifically, the main functionalities of the CM include (i) the detection of communication failures (handoffs/disconnections,

interface disabling, TCP failures, IP address losses) and reconnections, (ii) the detection of unavailable interfaces, (iii) the management of possible network interface reconfigurations, and (iv) the selection of the best network interface available to connect to a peer CM. In particular, the implementation of the latter functionality (iv) requires that the CM configures both the routing service and its own outgoing TCP connections, in order to determine the interface to be used for segment transmission. In case of network reconfiguration, the CM reconfigures accordingly the routing service. All these functionalities are provided transparently to the application.

The Routing mechanism is not part of the HiNIF architecture but it is opportunely configured at run-time. The HiNIF architecture has been implemented using ANSI C language on Linux OS. In particular, the actual implementation of the Interceptor, Failure Detector and Enhancement Detector components uses the netfilter framework and the iptables packet selection system [6] provided by the Linux kernel since version 2.4.

4. Experimental Evaluation

In this section, we describe an experimental study we have carried out in order to evaluate the effectiveness of the HiNIF architecture in supporting TCP-based services in multimedia mobile blogging. To this end, we used an existing multimedia blog system, typically working on fixed hosts, and executed its application client on a laptop, while moving around, exchanging network cards and reconfiguring the network subsystem. Both the client (on the laptop) and the server (on a fixed host) sides were equipped with the HiNIF system, that provides, transparently, the ABC connectivity.

Our evaluation consisted of the following two phases. The first phase aimed at assessing whether the HiNIF architecture can transparently provide ABC connectivity in mobile scenarios, without requiring any modification to the application. The second phase aimed at evaluating the overhead introduced by the HiNIF architecture, in terms of network delay and bandwidth availability of the “enhanced” connection, as opposed to a standard TCP connection (needles to say, this latter comparison applies only to scenarios where the TCP connection does not suffer from communication failures).

Our experiments were based on EasyMoblog 0.5 [7], an open source application that can be updated simply via e-mail. Specifically, we updated the moBlog by sending two messages containing an audio file and a video file, respectively. Ideally an optimal

client device for our experiment would be a multi-homed PDA, equipped with a camera (e.g., an HP iPAQ h6315 such as that illustrated in Figure 3, which incorporates both a camera, and three-way wireless capabilities - namely, GSM/GPRS, Wi-Fi, and Bluetooth[8]).

Figure 3: iPAQ h6315 Recording a Video.

However, we decided to run our first test using, as mobile host, a laptop computer with Pentium III 1 GHz, 512 MB RAM, a 100 Mbps Ethernet card and two PCMCIA wireless interfaces: an ASUS WL-110, configured as an infrastructure-based 802.11b mode (nominal rate 11 Mbps) and a Linksys 802.11b WPC11, configured as ad-hoc mode (nominal rate 1 Mbps). The server host was a Pentium III 669 MHz with 512 MB RAM and a 100 Mbps Ethernet card. Both hosts ran Linux, kernel version 2.6.11.

Figure 4. ABC Data Transfer.

The first test involved the transmission of a 20 MB video file from the mobile node to the server. The results of this test are illustrated in Figure 4. In this Figure, the x axis represents the time (in seconds) elapsed from the beginning of the transmission; the y axis represents the amount of video data (in megabytes) received at the server. At the beginning of the transmission (point A in Fig. 4), the mobile node connected the server through the 1 Mbps ad-hoc network. After about 40 seconds (point B) we extracted the WPC11 card from the pcmcia socket, and, after a

few seconds we plugged in the 11 Mbps infrastructure-based wireless card. In about one second, the Communication Manager at the mobile node detected the card exchange, reconfigured the IP address, set up a new connection with the server side Communication Manager, and resumed the transmission (point C) from where it had been interrupted. Note that, from the point of view of the blog application, only a transmission delay occurred.

After about 15 seconds, we connected the Ethernet cable to our mobile node. The Communication Manager at this node detected the link beat, configured the IP address of the cabled interface, and set up a new connection with the Communication Manager at the server side. Then, it suspended the transmission through the 11 Mbps wireless LAN (point D), and resumed it (point E) through the cabled network. Note that, in this case, transmission is suspended for a shorter time interval (i.e., the D-E elapsed time, in Fig.4) than in the previous case (i.e. B-C elapsed time, in Fig. 4), as the transmission continued through the WLAN until the Communication Managers set up the new connection via the wired network.

In summary, our first phase evaluation shows that, in spite of the IP address loss and reconfiguration caused by the first wireless card exchange, the data transmission continued without the applications being aware of any communication problem. Moreover, when a new network interface becomes available, which is faster than one already in use, the HiNIF can switch the communication to the new network within a reasonable time (about one second). Thus, in essence, the HiNIF architecture can provide ABC connectivity in mobile scenarios, transparently to the applications.

Table 1. Upload Time in Seconds.

	cabled 100Mb/s		WLAN 11 Mb/s		WLAN 1Mb/s	
	TCP	HiNIF	TCP	HiNIF	TCP	HiNIF
1B	0.01	0.05	0.02	0.07	0.02	0.07
1MB	0.25	0.31	2.06	2.12	11.00	11.08
5MB	1.18	1.26	10.26	10.37	52.72	52.53
10MB	2.37	2.48	20.83	20.62	107.13	107.18
20MB	4.63	4.76	40.98	40.76	213.96	214.07

The second phase of our experimental evaluation involved the upload of a set of files of different size (1B, 1MB, 5MB, 10MB, and 20MB respectively) from the mobile node to the server. Each transmission was carried out twice on each network interface available (1Mbps ad-hoc WLAN, 11Mbps infrastructure WLAN, 100 Mbps cabled Ethernet), once using the enhanced connection provided by the HiNIF and once, for comparison, using the simple TCP connection. During these transmissions, the mobile node was not moved in

space. The aim of that evaluation was to investigate the overhead introduced by the HiNIF. To this end, the test measured the time elapsed from the beginning to the end of each upload; the results of this test are summarized in Table 1. In this table, the term HiNIF refers to the connectivity provided by the HiNIF. In particular, the time elapsed to transport the 1 byte file can be thought of as the time necessary to set up a connection (either “enhanced”, or TCP standard). Thus, the difference between the upload time (for the single byte) provided by the HiNIF and the simple TCP represents the latency overhead introduced by the HiNIF, which averaged to 0.05 seconds. A similar small time difference characterized the upload of larger size files. This proves that, after the connections have been set up, the HiNIF connectivity provides the same data rate as the standard TCP connections. In other words, HiNIF does not limit bandwidth usage.

5. Concluding Remarks

In this paper, we have introduced a middleware architecture that can support effectively ABC seamless connectivity for multi-homed devices, and offer service continuity in mobile multimedia blogging, transparently to the applications. We have described an initial experimental evaluation of this architecture we have carried out. The results we have obtained using laptop platforms show the effectiveness of our approach. Our future work will include the evaluation of our architecture on PDAs running the Linux OS, and the extension of our architecture to deal with non functional application requirements, such as fault tolerance and security.

6. References

- [1] Howard Rheingold, “Smart Mobs: The Next Social Revolution”, Basic Books Eds.
- [2] Nokia Lifeblog, online www.nokia.com/lifeblog/
- [3] Microsoft Sensecam Project, online http://research.microsoft.com/sensecam/project_sensecam.aspx
- [4] V. Ghini, G. Pau, and P. Salomoni, “Always-Best-Served Music Distribution for Nomadic Users over Heterogeneous Networks”, in IEEE Communications Magazine, Vol. 5, May 2005.
- [5] T. Ernst, N. Montavont, E. Paik, C. Ng, K. Kuladinithi, and T. Noel, “Goals and Benefits of Multihoming”, Internet-Draft, February 2005.
- [6] “The netfilter/iptables project”, available online <http://www.iptables.org/>
- [7] EasyMoblog, <http://www.easymoblog.org/>
- [8] HP handheld platforms home page, <http://welcome.hp.com/country/us/en/prodserv/handheld.html>