Java-based Proactive Buffering for Multimedia Streaming Continuity in the Wireless Internet

Paolo Bellavista, Antonio Corradi, Luca Foschini
Dipartimento di Elettronica Informatica e Sistemistica - Università di Bologna
Viale Risorgimento, 2 – 40136 Bologna – ITALY
Phone: +39-051-2093001; Fax: +39-051-2093073
{pbellavista, acorradi, lfoschini}@deis.unibo.it

Abstract

New challenging deployment scenarios are accommodating portable devices with limited and heterogeneous capabilities that roam among wireless access localities during service provisioning with session continuity requirements, such as in multimedia streaming. The paper proposes an original two-level buffering strategy to maintain streaming continuity independently of client roaming at provision time. In particular, it focuses on a specific component of the proposed support infrastructure, i.e., the pure Java buffering component, which has shown to outperform the standard Java Media Framework in both streaming initialization time and imposed overhead.

1. Two-Level Buffering for Streaming Continuity during Handoffs

The popularity of wireless devices and the increasing availability of Wi-Fi Internet Access Points (APs) are stimulating the provisioning of distributed services to a wide variety of mobile terminals, with heterogeneous and often limited resources. Even if device and network capabilities are increasing and increasing, the development and deployment of wireless applications are going to remain a very challenging task, in particular when dealing with *continuous services*, i.e., applications that distribute time-continuous data flows to their clients, such as in the case of audio/video streaming.

Let us consider the common deployment scenario where wireless solutions extend the accessibility to the traditional Internet via APs working as bridges between fixed hosts and wireless devices [1]. A notable example is the case of IEEE 802.11 APs that support connectivity of Wi-Fi terminals to a wired local area network. In the following, we indicate these integrated networks with fixed Internet hosts, wireless terminals, and wireless APs in between, as the *Wireless Internet (WI)*.

We claim the need of middleware solutions to dynamically build overlay support networks for the provisioning of WI continuous services to portable devices. Hence, we have developed the **M**obile agent-based **U**biquitous multimedia **M**iddleware (MUM*), a dynamic and flexible infrastructure to support both streaming quality adaptation and session continuity, independently of client roaming in the WI. A more detailed MUM description is out of the scope of the paper and can be found in [2].

This paper focuses on an essential aspect of our middleware: avoiding provisioning interruptions when a client roams from one wireless locality to one another (*wireless cell handoff*) at runtime. Many research efforts tend to solve this problem by working at the network layer [3, 4]; a few techniques have been proposed also at higher layers [5]. Our handoff technique entirely works at the application layer by exploiting an original two-level buffering solution that integrates with our Wi-Fi mobility prediction approach [7].

Client-side data buffering is a usual approach in multimedia streaming over wired networks to smooth congestions that may occur on the client-server path. Some recent research activities have started proposing client buffering also to overcome short-duration disconnections during service delivery in wireless environments [6]. They all assume that clients have enough memory to maintain pre-fetched streaming data and this assumption is often inapplicable in the WI scenario, e.g., where the client is either a PDA or a cell phone.

We propose an additional level of data buffering at some intermediate nodes along the client-server path, i.e., at the proxy nodes in the wireless localities where the associated clients are currently connected. We claim that proxy-based buffering can significantly contribute both to minimize client-side resource usage and to guarantee streaming continuity during handoff. In fact, our previous research experiences have shown that handoff completion time depends on several fac-

* Additional details and the code of the MUM prototype are available at `http://www.lia.deis.unibo.it/Research/MUM/`

tors, e.g., proxy-server hop distance, number of active components in the server-to-proxy path, and network congestion state. In several deployment scenarios, that time interval may be definitely not negligible, thus making unfeasible buffering solutions only at the client side [2]. Second-level proxy buffering exploits the memory resources of workstations traversed by multimedia flows on the wired network.

In addition, our middleware works to proactively migrate proxy-located buffers to the expected next wireless access localities of currently served clients, before client handoffs. In particular, our handoff facility exploits the time saved thanks to proactive buffer migration to perform time consuming operations, e.g., to activate/setup proxies in predicted localities and to begin rebinding operations between proxies and remote streaming servers. The handoff facility exploits an innovative technique for mobility prediction within Wi-Fi environments called Received Signal Strength Indication-Grey Model (RSSI-GM for shortly) [7].

2. Buffering Mechanism Implementation

The section presents the primary core mechanism of the MUM handoff facility, i.e., the proxy/client buffering mechanism. Our buffering solution is strictly coupled with the portable library adopted for multimedia streaming, i.e., the Java Media Framework (JMF). We developed a proactive buffer prototype, implemented in terms of a circular buffer, to verify the feasibility of the MUM approach from the performance point of view, also when adopting the application-level Java-based JMF. In the following, we first present some main elements of JMF, needed for the full understanding of the following, and then we detail the design and implementation of the MUM buffer solution.

JMF is the SUN Java-based framework proposed for multimedia object management. JMF adopts the RealTime Protocol (RTP) for video streaming and the RealTime Control Protocol (RTCP) to monitor the network status at provision time. JMF processes the frames of a multimedia flow by passing them through a pipeline, called plug-in chain, composed of various stages; each plug-in can perform a specific flow transformation. JMF simplifies multimedia application development by hiding frame transformations at the library level and by providing higher level APIs both to abstract frame sources/sinks, e.g., `Data-Source/DataSink`, and to encapsulate the construction and usage of plug-in chains, e.g., `Player` and `Processor`. JMF is also in charge of buffering functions and exposes APIs for flow buffering control, e.g., the `BufferControl` object that can be obtained from `Player` or `Processor` via `getControl()`.

Let us note that high-level JMF APIs simplify multimedia application development, but do not always offer the fine control granularity required to realize advanced and customized services. In addition, the MUM handoff facility needs functions to extract, set, and manage directly the circular buffer; JMF does not support this kind of functions. Moreover, from our experience, JMF buffering sometimes exhibits quite unpredictable performance, depending on the underlying operating systems and Java Virtual Machine (JVM) versions. For all these reasons, we have decided to develop an original buffering mechanism outside JMF both to control directly buffering functions and to operate with finer granularity directly at the frame level.

The above reasons suggested us to deeply explore and use the lower level mechanisms available in JMF to directly and precisely control flow progress and frame-level buffering. In particular, we have decided to directly construct and manage plug-in chain stages and all Java threads that contribute to transform frames and move them towards the pipeline, as depicted in Figure 1. For the sake of simplicity, the figure exemplifies the operations of the MUM buffer for the specific plug-in chain built at the client to render an H263 presentation transmitted over RTP. This chain consists of 4 stages: the raw buffer parser collecting RTP packets, the H263 decoder, the YUV to RGB converter and the video renderer.

Moreover, our buffer implementation does not endanger portability and can run over any JVM-equipped host: in fact, MUM is completely JMF-compliant, does not modify the JMF implementation, and only achieves the flexibility and efficiency needed by accessing lower-level JMF mechanisms, typically hidden when using the higher-level JMF APIs.

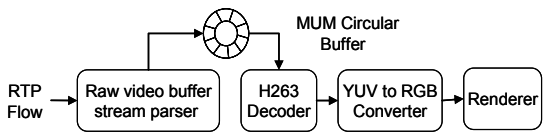

Figure 1. Client Plug-in Chain

3. Experimental Results

The section presents experimental results about the circular buffer implementation to point out how, by controlling directly the circular buffer and plug-in chain, it is possible to improve the usual JMF performance. First, the section analyzes the plug-in initialization phase; then, it considers the MUM behavior at runtime and evaluates how the interposition of client/proxy buffers affects CPU load.

The used testbed consists of Sun Blade 2000 workstations equipped with 900MHz processors, 1024MB RAM, and connected by a 100 Mbps Ethernet LAN.

The workstations are equipped with SunOS v5.9 operating system and JVM v1.4.2_03-b02, and exploit JMF Performance Pack for Solaris v2.1.1e. Heterogeneous clients with limited hardware/software capabilities are represented by Asus laptops with IEEE802.11b connectivity and Windows2000, the same JVM version, and the JMF Performance Pack for Windows. In our experiments we used a H263 encoded video stream (length=20'24", composed by 18244 frames, frame dimension=176x144 pixels, frame rate=14,9 frames/s). All experimental results reported in the following are average values over a set of 100 runs.

The standard JMF plug-in chain initialization tends to be as general as possible: when there is a new in/out flow, JMF tries to apply all possible de/coders to the flow. This produces long initialization times, due to both the loading of all possible plug-in descriptors and the control of all possible dependencies. MUM exploits the knowledge of presentation descriptions and client profiles to previously determine needed plug-ins for the delivered multimedia presentation. Further details about our distributed metadata storage are available in [8]. Experimental results demonstrate that MUM direct plug-in chain construction drastically reduces initialization time at both the client and the server, as reported in Figure 2. In addition, the wide prototype testing and performance evaluation have contributed to isolate the main JMF library bottlenecks [2].

The average time for usual JMF chain initiation at the server is 303ms, while our custom solution builds the chain in only 94ms; similarly, at the client the delay passes from 374ms to 110ms (including buffer initialization time). At the proxy, the performance improvement is reduced: the reason is that proxy plug-in chain consists of only two stages, since proxies only forward incoming RTP packets to clients, and do not check neither packet payload nor plug-in dependencies.

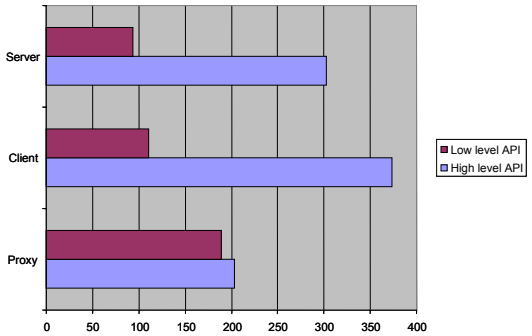

Figure 2. Plug-in chain initiation time

Moreover, we evaluated runtime CPU load by comparing classic JMF and MUM buffering solutions. Direct plug-in chain programming reduces CPU load from 14,65% to 11,42% at the server node, and from 5,23% to 3,25% at the client, while there are no significant improvements at proxies. Our implementation outperforms JMF at client and server nodes by eliminating some control threads for either plug-in chain dependences control or plug-in chain state change notification, e.g., to notify the end of the initialization phase. Proxy plug-in chains, consisting of only two stages, are less affected by these improvements.

In summary, our two-level buffering outperforms commonly used, JMF-embedded, buffering mechanisms. The promising performance results obtained are stimulating further related research activities. We are working on enabling the runtime adaptive decision of buffer size at both the client and the proxy. In addition, we are developing wide-scale handoff simulations to extensively benchmark our proposal and to compare its performance with the other primary research prototypes in the field [3, 4].

Acknowledgments

Work supported by the MIUR FIRB WEB-MINDS and the CNR Strategic IS-MANET Projects.

References

[1] M. S. Corson, J. P. Macker, V. D. Park, "Mobile and Wireless Internet Services: Putting the Pieces Together, *IEEE Communications*, Vol. 39, No. 6, 2001.

[2] P. Bellavista, A. Corradi, L. Foschini, "MUM: a Middleware for the Provisioning of Continuous Services to Mobile Users", *IEEE Int. Symposium on Computers and Communications*, 2004.

[3] A.T. Campbell et alii, "Comparison of IP micromobility protocols", *IEEE Wireless Communications*, Vol. 9, No. 1, 2002.

[4] S. Debashis et alii, "Mobility Support in IP: A Survey of Related Protocols", *IEEE Network*, Vol. 18, No. 6, 2004.

[5] M.E. Wesley, "At What Layer Does Mobility Belong?", *IEEE Communications*, Vol. 42, No. 10, 2004.

[6] F.H.P. Fitzek, M. Reisslein, "A Prefetching Protocol for Continuous Media Streaming in Wireless Environments", *IEEE Journal on Selected Areas in Communications*, Vol. 19, No. 10, 2001.

[7] P. Bellavista, A. Corradi, C. Giannelli, "Mobility Prediction for Mobile Agent-based Service Continuity in the Wireless Internet", *Int. Workshop on Mobility Aware Technologies and Applications (MATA'04)*, 2004.

[8] P. Bellavista, A. Corradi, L. Foschini, "MUMOC: an Active Infrastructure for Open Video Caching", *Int. Conference on Distributed Frameworks for Multimedia Applications (DFMA'05)*, 2005.