Short Paper

A WAP-based, Push-enabled Mobile Internet Application Platform[*]

HSUAN-HAO CHEN, KUOCHEN WANG AND HUNG-CHENG SHIH
Department of Computer Science
National Chiao Tung University
Hsinchu, 300 Taiwan

We propose a WAP-based, *Push-enabled* mobile Internet application platform, called *MAP*, to provide extensive services for users in the mobile environment. The goal of this platform is to provide a flexible, scalable and rapid-service-creation environment for mobile Internet applications to operate in. MAP has four main components, which have been designed and implemented: (1) a *WAP micro-browser* for mobile devices, (2) a *WAP simulator* for desktops or notebooks, (3) a WAP Push proxy gateway, the *MBL Gateway*, and (4) a distributed mobile agents server, *Wagent*. This platform enables mobile clients to access legacy information systems, Intranets and WWW services conveniently. It applies the advantages of the *mobile agent* paradigm and *WAP Push* technologies to extend Internet services to wireless environments. It also makes it possible to push critical information, such as news and stock prices, to mobile users in real time. We have also evaluated the performance of the MBL Gateway, which is a key component of the platform, using a realistic traffic model. Experiment results show that the MBL Gateway is more efficient than the other two notable open source WAP gateways in terms of average response time. The MBL Gateway reduces the average response time by up to 25% and 87% compared to the Standalone Kannel Gateway and Original Kannel Gateway, respectively, under the highest load (270 requests/sec when the session arrival rate $\lambda = 0.015$).

Keywords: mobile internet application platform, WAP micro-browser, WAP simulator, WAP push proxy gateway, distributed mobile agent server

1. INTRODUCTION

In recent years, the Internet has allowed millions of wireline users to access various services easily. These services are always on, always available, and easy to use. They also enrich people's lives and reduce business costs. In the meantime, wireless voice communication has also grown at a rapid pace and achieved wide acceptance. Cellular

Received April 30, 2004; revised August 31, 2004 & February 16, 2005; accepted April 18, 2005.
Communicated by David H. C. Du.

[*] This work was supported by the NCTU EECS-MediaTek Research Center under grant Q583 and the MOE Program of Excellence Research under grant 89-E-FA04-1-4. The comments made by the reviewers and the Editor helped improve the quality of this paper.

[*] A brief version of this paper was presented at *the 7th Mobile Computing Workshop*, Hsinchu, Taiwan, 2001, pp. 44-50.

phones have become indispensable for mobile users. However, most of the technologies developed for the Internet have been designed for desktops, large computers, and wired networks with high bandwidth. Mobile devices have limited resources in terms of computing power and display capabilities compared to desktop computers, and wireless networks also have low bandwidth. In addition, independent wireless data carriers use their own technologies and mobile devices from different manufacturers, which makes it hard for them to talk to each other. Therefore, we need a mobile Internet application platform that can effectively deal with the constraints imposed by both the Internet and wireless technologies. It must support open industry standards for mobile application development [1, 2] and be scalable across a variety of wireless standards [28, 29].

The Wireless Application Protocol (WAP), developed by the WAP Forum [3], is a leading global open standard for applications run on wireless networks. WAP provides a uniform technology platform with consistent content formats for delivering Internet or Intranet based information and services to mobile phones and other mobile devices. The WAP standard defines the application environment, the communication protocols, a wireless markup language (WML) [4] and a scripting language (WMLScript) [5]. The major enhancement in WAP 1.2 over WAP 1.1 is support for delivering push contents to mobile clients. Note that there are SMS-based message-delivery systems in place today that resemble Push, but that the WAP-based Push model adds more interactive functionality to the push process.

Previous studies on the development and deployment of applications for mobile wireless environments are reviewed in the following. In [35], WAP-based system capable of multiplexing data flows at the transport level was proposed. An architectural comparison between the SIM Toolkit [36] and WAP to support mobile e-commerce applications was made in [37]. A system for the development of WAP-based applications, in which site development is automated, was proposed in [38]. In [39], a general architectural framework for developing and deploying portable applications and services accessible by WAP-compliant mobile terminals was presented. The main differences between this framework and our MAP platform are that it does not incorporate a WAP Push capability to enhance user experience with active mobile services [3], and that only WWW-based contents are provided.

In this paper, we propose a WAP-based, Push-enabled mobile Internet application platform to provide extensive services for users in mobile wireless environments. This platform not only provides mobile connectivity to help users access WWW or Intranet services but also works with legacy information systems. It applies the advantages of the *mobile agent* paradigm and *WAP Push* technologies to extend versatile services to mobile wireless environments.

2. PROPOSED MOBILE INTERNET APPLICATION PLATFORM ARCHITECTURE: MAP

The proposed mobile Internet application platform (MAP) architecture, as shown in Fig. 1, is based on the WAP model, and all the components are compliant with the WAP 1.2 specifications. There are four main components in the platform: the *MBL Browser*, *WAP & i-mode Simulator*, *MBL Gateway*, and *Wagent Server*.

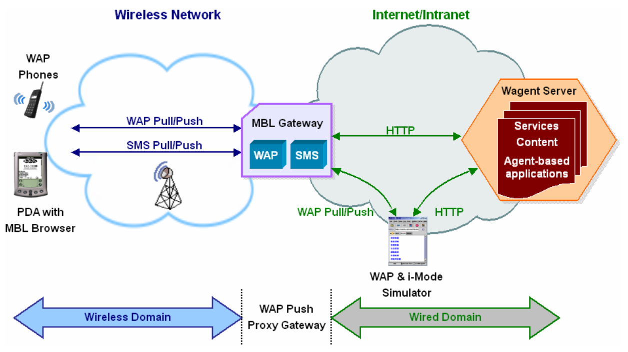

Fig. 1. MAP architecture.

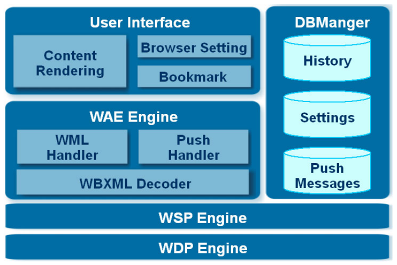

Fig. 2. Architecture of the MBL browser.

2.1 MBL Browser

The MBL Browser is a WAP micro-browser that can run on a wide variety of devices that incorporate the Java Kilobyte Virtual Machine (KVM) [6]. It enables mobile users to access WAP contents and services by using mobile devices like Palm PDAs. It also supports unconfirmed Push at default port number 2948 and Big 5 character set encoding. We used the J2ME (Java 2 Platform, Micro Edition) Connected Limited Device Configuration (CLDC) [6] and its KVM to implement our browser on the Palm OS. Fig. 2 shows the architecture of the MBL Browser.

The WDP (Wireless Datagram Protocol) [7] and WSP (Wireless Session Protocol) Engines set up a wireless connection between a mobile device and the WAP gateway by using the standard WAP protocols. They also receive unconfirmed Push messages at port 2948. The WBXML (WAP Binary XML) [8] Decoder is responsible for decoding the binary WML stream received from the WAP gateway and organizing it in a Document Object Model (DOM) [9] tree that can be handled by the WML Handler or Push Handler. If the received stream is a Push message, the Push Handler will determine whether it is an SI (Service Indication) [10] or SL (Service Loading) [11] message and cope with it.

The *Content Rendering* component is responsible for displaying WML contents and handling scrolling, clicking, or other input actions from users. The *Browser Setting* lets a user edit preferences, such as a WAP gateway IP address, and enable/disable some WML tags, WBMP (Wireless Bitmap) [12] images, etc. The *Bookmark* is responsible for keeping track of the user's favorite URLs (Uniform Resource Locators). The Browser Setting and Bookmark modules are also responsible for loading and storing the user's preferences and URLs by using persistent storage, the *Settings* database. The *History* module is responsible for recording previously navigated URLs.

2.2 WAP & i-Mode Simulator

Our WAP & i-Mode Simulator is the first mobile Internet browser that supports both WAP and i-Mode contents. It allows WAP or i-Mode application developers to use regular input and navigation based PCs as mobile devices to test their applications. The WAP & i-Mode Simulator is a complete implementation of a WAP browser. It supports the full set of WML1.2 tags, WMLScript and WAP 1.2 Push specification.

2.3 MBL Gateway

The MBL Gateway is a Java 2-based WAP Push Proxy Gateway, which was inspired by the open source Kannel WAP Gateway [13]. Our goal is to develop a cross-platform, scalable, efficient WAP gateway.

2.3.1 External interfaces of the MBL gateway

The MBL Gateway has interfaces to four external entities, as shown in Fig. 3:

- WAP-enabled mobile devices, using WAP protocols;
- WAP-enabled mobile devices, using SMS (Short Massage Service) protocols;
- HTTP servers, using HTTP to fetch WAP contents;
- PIs (Push Initiators), using the Push Access Protocol (PAP).

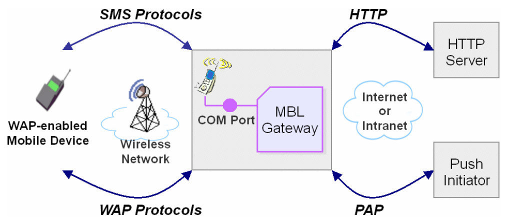

Fig. 3. External interfaces of the MBL gateway.

The MBL Gateway enables people to use any WAP-enabled mobile device (cellular phone, PDA, etc.) to browse the web, send or receive e-mail, and access services on the Internet or Intranets. The gateway is also the access point for the PI on the Internet to push messages to Push-enabled devices. In addition, the MBL Gateway is also capable of sending and receiving short messages. Once the GSM modem has been activated, the gateway does not require a WAP connection when sending push messages with the GSM modem.

2.3.2 Internal structure of the MBL gateway

The internal structure of the MBL Gateway is shown in Fig. 4. We have implemented

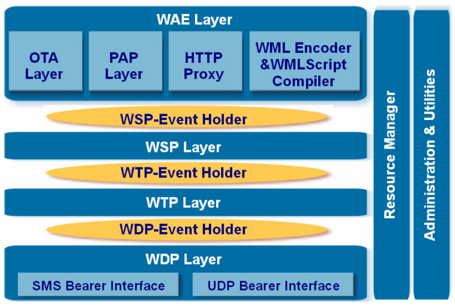

Fig. 4. Structure of the MBL gateway.

the entire WAP protocol stack with multiple WAP layer threads. Once the gateway starts, the main process creates an instance of each layer thread according to the administrator's configuration. The communication between the adjacent WAP layer threads occurs via WAP EventHolders. To support WAP Push in the MBL Gateway, we have implemented the PAP and OTA (Over the Air) layer threads. The PAP layer thread communicates with the PI by using PAP over HTTP, and the OTA layer thread is responsible for pushing the messages to the mobile device through the lower WAP protocol stack. The HTTP Proxy thread handles each HTTP request and caches WAP pages that are accessed by WAP clients. When the HTTP Proxy thread receives WAP pages from content servers, it asks the WML Encoder or WMLScript Compiler to transform the textual data into an appropriate binary form. The Resource Manager manages the resource usage in the MBL Gateway. It collects unused objects for later use and asks the Java virtual machine to recycle the garbage memory when the gateway is idle. The Administration & Utilities module provides a graphical user interface that the administrator can use to reconfigure the gateway at runtime.

2.4 Wagent Server

Even though the MBL Browser and MBL Gateway are capable of enabling WAP clients to access WAP applications on the Internet, we still need an application server to provide a flexible, platform-independent, rapid-service-creation environment for applications to dwell in.

2.4.1 Structure of a wagent server

A Wagent Server, as shown in Fig. 5, consists of four major components: the *Web Server Engine*, *Appliances*, *Agents*, and *Push Initiator* (PI). To provide the functionalities of a web server, we have modified the packages of the W3C Jigsaw Web Server [14] to serve as the *Web Server Engine* in the Wagent Server.

Moreover, a Wagent Server can serve not only as a web server but also as a distributed mobile agents system. Each server contains a set of local appliances that encapsulate resources or capabilities such as peripheral device drivers or database connections. It also hosts a population of agents: mobile computational objects that use local appliances and communicate with each other. In order to provide Push services for mobile WAP clients, we enable the PI to run on each server and accept Push requests from the Web Server

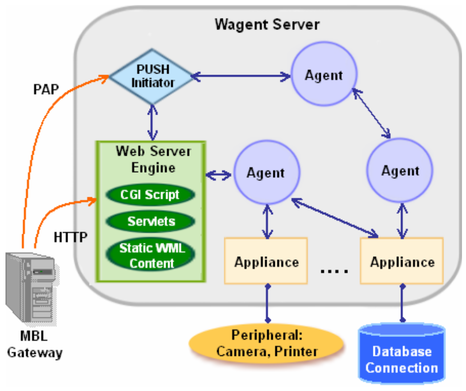

Fig. 5. Structure of a Wagent server.

Engine or agents. The PI communicates with the MBL Gateway by using PAP over the Internet.

2.4.2 System services of a Wagent server

A Wagent Server provides five necessary system services:

- It provides an environment in which agents can live.
- It provides facilities for the life-cycles and mobility of agents.
- It provides services which are local to the server and a mechanism to which agents can make requests in order to access services.
- It provides a set of services that allow agents and appliances to discover and interact meaningfully with each other.
- It provides a security scheme to protect a host from its agents.

Each Wagent Server typically has a server manager agent that contains a registry that maintains the membership of a server in the global Wagent network. A set of Wagent servers forms an ad-hoc Wagent network.

3. DESIGN AND IMPLEMENTATION

In this section, we will address design and implementation issues, and problems that we encountered when we designed and implemented the platform.

3.1 Issues Related to the MBL Browser

Most of the problems encountered when developing the MBL Browser were caused by the resource constraints imposed on mobile devices. Although J2ME CLDC provides core class libraries covering the user interfaces, networking, and persistent storage that can be used in PDAs, it still has some limitations, mainly due to insufficient network handling. Thus, the networking functions of the MBL Browser were written so as to solve these problems.

3.1.1 Avoid using third party libraries

In the early stage of the implementation, we used kAWT [16], which is a simplified version of AWT [17] for KVM, to develop the user interface of the browser. However, we found that the program size and memory requirement both imposed tremendous demands because of the use of the kAWT library. Later, we avoided using third party libraries so to keep the program size and memory usage as small as possible. Another good practice was using a user database to store large tables or data rather than keeping them in memory.

3.1.2 Use it or dispose of it

When objects, databases, or network connections are no longer being used, we dispose of them or close them as soon as possible. Some user interface components use much memory, so we have tried not to make the user interface too complicated. We have also carefully designed the database structures, such as index, linkage, or fixed size records, so that only necessary records (not the whole database) are loaded into the program memory.

3.2 Issues Related to the MBL Gateway

We have focused much effort on enhancing the performance of the gateway. Most of these efforts involved the use of programming techniques, such as object reuse, thread pooling, selecting an efficient algorithm and appropriate data structures for different situations, and using buffering to maximize I/O performance [18-20].

3.2.1 Layer-to-layer communications

Communication between different WAP layer threads and between entities within a layer is accomplished by means of *WAP events*. A WAP event corresponds to a particular service primitive defined in the WAP specifications. There is an *EventHolder*, as shown in Fig. 6, between two adjacent layers threads. The events stored in queues are scheduled according to their levels of priority. The earlier they are created, the higher their priority.

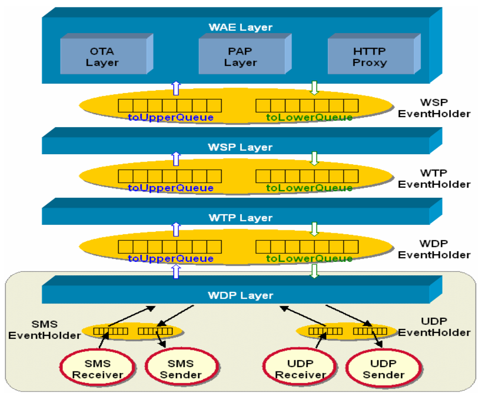

Fig. 6. Communications between different WAP layers.

3.2.2 Encoding WML and compiling WMLScript

To encode a WML document, we need an XML parser that will take a WML document as input and make its elements and attributes available for processing. We use the Java™ API for XML Processing [21], version 1.1, to process WML documents. Once the WML Encoder gets the textual content of a WML page, the encoder parses it and generates a DOM document. Then, the encoder converts it into binary form according to the public identifier of the document. The WML Encoder converts the strings in the WML page using a proper character set that the cellular phone accepts. To compile the WMLScript source code, we utilize the Java CUP (Java based Constructor of Useful Parsers) [22] package to implement the WMLScript Compiler.

3.2.3 Accessing the internet efficiently

To enhance the performance of the WAP gateway, an efficient means of accessing the Internet is necessary. In the MBL Gateway, we have implemented an HTTP proxy to cache WAP pages that are frequently accessed by WAP clients.

3.3 Implementation of the WAP Push Architecture

Fig. 7 shows the push architecture of MAP. The *Push Initiator* (PI) in the Wagent Server has two modules. The *Request Listener* listens for push requests from other applications and passes the requests to the *PAP Client*. The PAP Client then interacts with the MBL Gateway (WAP Push Proxy Gateway) and sends the information required by the MBL Gateway for push message delivery. To support *WAP Push* in the MBL Gateway, we implement the *PAP* and *OTA* layers in addition to the WSP layer. The PAP layer thread listens to port 8080 and communicates with the PI by using PAP over HTTP. For each request, there is a corresponding response. Once the PAP layer thread accepts a Push message submitted by the PI, it passes the Push content to the OTA layer thread.

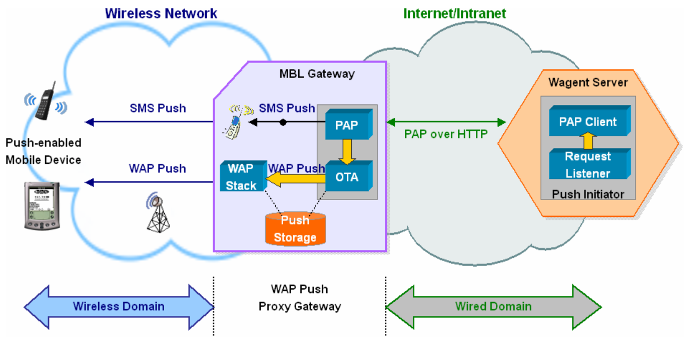

Fig. 7. The push architecture in MAP.

If it is a connectionless Push, the OTA layer thread pushes the message directly to the non-secure WDP port in the client. If it is a connection-oriented Push, the OTA layer thread asks the *Session Initiation Application* (SIA) in the client to set up a new WSP session with the gateway. To support connection-oriented Push at the client-side, an SIA and an *Application Dispatcher* (AD) must be present in the client device. Since no mobile devices support WAP Push at present, the MBL Gateway uses SMS to send text messages for the purpose of notification.

The difference in the SMS implementation between the MBL Gateway and Kannel Gateway is that our gateway connects to a GSM mobile station (MS) instead of the short message service center (SMSC) [24]. We treat an MS as a GSM modem that enables the the MBL Gateway to access the GSM network directly. Once the PAP layer thread has received a push message sent from the PI, the PAP layer thread may choose to package the push message into one or several short messages and transmit them to the mobile device.

3.4 Issues Related to the Wagent Server

3.4.1 RMI and object serialization

To facilitate the creation and manipulation of distributed objects for a rapid-service-creation environment, the Wagent server makes extensive use of remote method invocation (RMI) [25] and object serialization. The type safe nature of Java prevents an object from being cast into types that are not specifically implemented. This design provides security for objects. Thus, remote objects cannot call methods that are not exposed through a remote interface.

3.4.2 Agent mobility

Although Java provides support for moving codes between virtual machines, supporting agent mobility is still inconvenient. When codes move from one virtual machine to another, the new virtual machine needs a copy of the class definitions that the agent might require. In many uses of mobile codes this is not a problem, because various virtual machines can share a common, centralized code repository. Since Wagent is designed to avoid centralization, no such strategy is possible. We have implemented a Wagent Server that depends on the class definitions to support an agent that can migrate from server to server.

Because Java is late-binding, one cannot statically enumerate all the classes upon which an agent depends. There is always a possibility that an agent will suddenly attempt to load a new class that is no longer available locally and the server that the agent came from has disappeared. Since servers are long-lived entities, this problem should not occur too often. Another problem is *versioning*. Given agents' mobility, it is possible that when an agent is revised, older versions will remain within the Wagent. Java does not differentiate well between two versions of the same class. One way to solve this problem is to modify the agent names to reflect the version of a particular agent.

4. PERFORMANCE EVALUATION

The WAP gateway performs two main operations: protocol translation and content conversion. Since these operations are computationally intensive, the WAP gateway, which is a key component, may become a bottleneck. We have conducted experiments to evaluate the performance of our MBL Gateway, the original Kannel WAP Gateway [13], and a Standalone Kannel WAP Gateway [34] that removes the built-in load balancing capability of the original Kannel WAP Gateway by means of a realistic WAP traffic model.

4.1 Experiment Setup

The experimental results were obtained under the configuration shown in Table 1 and the network environment shown in Fig. 8. To evaluate the performance of the WAP gateways, we adopted the traffic model that was proposed in [27] to generate WAP traffic. We have also implemented a WAP benchmark program that can simulate a large number of users simultaneously accessing the Internet through a pre-configured WAP gateway. This program generates WAP traffic according to the traffic model and measures the response time. The WAP benchmark program takes a file list that contains the filenames and paths of WAP pages that reside on the content server. The program computes the popularity of each page to determine when the page should be loaded.

Table 1. Configuration for the performance evaluation.

Consideration	Configuration
Network Environment	LAN: 100Mb/s Ethernet
Client Hardware and OS	IP: 140.113.88.116, CPU: Pentium Ⅲ 700, RAM: 64M, Network Adapter: RealTek RTL 8139 Fast Ethernet, OS: Red Hat Linux 6.2 + CLE v0.9p1
Gateway Hardware and OS	IP: 140.113.88.118, CPU: Pentium Ⅲ 800, RAM: 256MB, Network Adapter: Intel PCI EtherExpress Pro100, OS: Red Hat Linux 6.2 + CLE v0.9p1
Server Hardware and OS	IP: 140.113.88.120, CPU: Pentium Ⅲ 700, RAM: 128M, Network Adapter: RealTek RTL 8139 Fast Ethernet, OS: Red Hat Linux 6.2 + CLE v0.9p1
WAP Client Benchmark Software	A homemade WAP benchmark program
Content Server Software	Apache 1.3.14 [26]

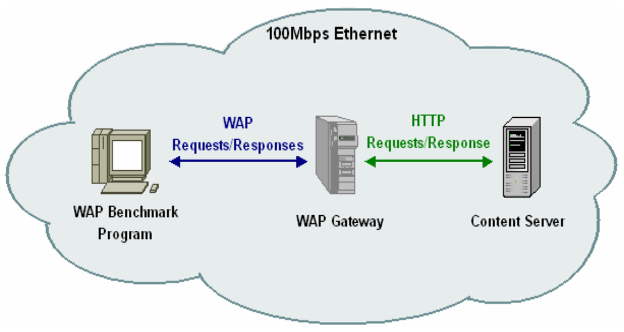

Fig. 8. The network environment.

The main difference between WAP pages and WWW pages is in the structure and size of a viewed page. A WAP page is much smaller than a WWW page due to its binary encoding and the fewer and smaller inline objects (WBMP images) included. The majority of WAP pages are WML files with an average of 0.4 embedded WBMP images. A total of 3,000 WAP documents, including WML, WBMP, and WMLScript files, were used in the experiment. The file sizes ranged from 44 bytes to 2,391 bytes. Table 2 summarizes the characteristics of the WAP pages. The parameters [27] used to generate WAP traffic are summarized in Table 3. New WAP client session arrivals followed the Poisson distribution [27, 30, 31]. The lowest WAP workload (when the session arrival rate $\lambda = 0.0005$) was approximately 10 requests per second, and the highest load (when $\lambda = 0.015$) was 270 requests per second. The number of clicks corresponded to the number of consecutive WAP pages which a WAP client requested during a session based on the inverse Gaussian distribution [27, 30, 33]. The client's user inter-click idle time (silent time) between the retrieval of two successive WAP pages was modeled as a LogNormal distribution [27].

Table 2. The characteristics of the WAP pages.

Characteristic	Value
Number of files	3000
Number of WML files	2000
Number of WBMP files	800
Number of WMLScript files	200
Average file size	531 bytes
Average encoded file size	104 bytes
Mean number of embedded objects per WML page	0.4
Mean number of links per WML page	3

Table 3. The parameters used to generate WAP traffic.

Variable	Law	Lowest Load	Highest Load
Session arrival	Poisson Process ($0.0005 \leqq \lambda \leqq 0.015$)	10 requests/sec	270 requests/ sec
Variable	Law	Mean	Std deviation
Number of clicks	Inverse Gaussian ($\mu = 8$; $\lambda = 5$)	8	10.1
Inter-click idle time	LogNormal ($m = 2$; $\sigma = 0.8$)	10.2 sec	9.6 sec

4.2 Experimental Results

To evaluate the performance of the WAP gateways, we varied the session arrival rate and measured the response time of WAP pages. The three different WAP gateways were evaluated in both connection-oriented and connectionless modes. Figs. 9 and 10 show the experiment results. The MBL Gateway reduced the average response time by

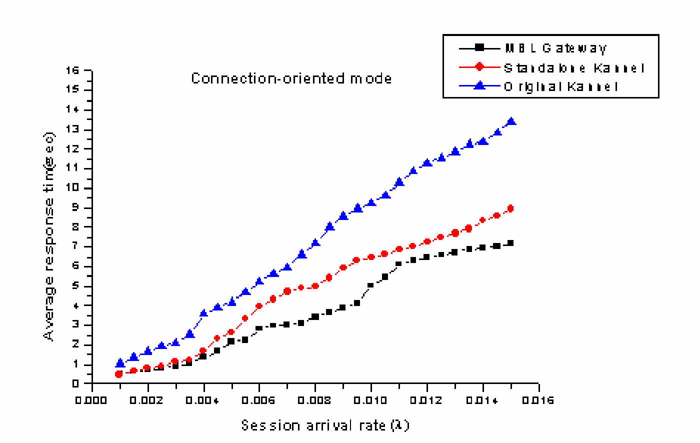

Fig. 9. The average response time vs. the session arrival rate for the three WAP gateways in connection-oriented mode.

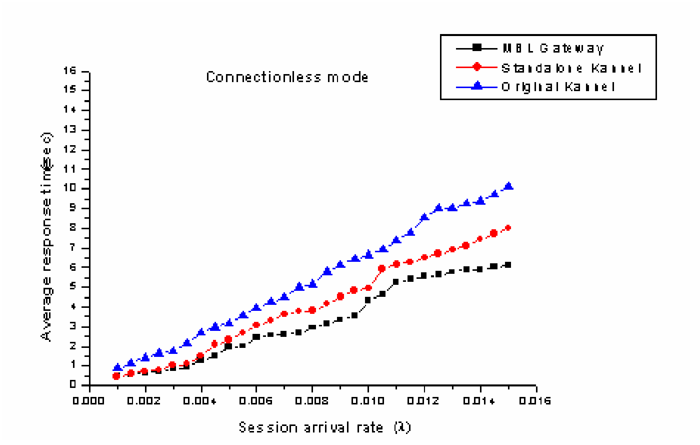

Fig. 10. The average response time vs. the session arrival rate for the three WAP gateways in connectionless mode.

up to 25% and 87% compared to the Standalone Kannel Gateway and Original Kannel Gateway, respectively, under the highest load (270 requests/sec when the session arrival rate $\lambda = 0.015$).

The experimental results show that the MBL Gateway is quite efficient compared to the other two gateways. This is because we have implemented an HTTP proxy module in the MBL gateway to cache WAP pages that are frequently accessed by WAP clients. The HTTP proxy module reduces the response time associated with obtaining WAP pages and the network traffic between the WAP gateway and content servers. In short, our implementation of an efficient HTTP proxy module is the main reason why the performance of the MBL Gateway is better than that of the other two gateways, in which the HTTP proxy mechanism is not implemented.

In addition, a monolithic gateway might become overloaded and is not sufficient to handle a huge amount of WAP traffic. Upgrading the gateway with a faster model usually results in high cost and still can not enable the gateway to scale up with demand. An efficient way to cope with growing demand is to add extra hardware resources so as to provide a clustered gateway architecture. That is, turning a monolithic MBL gateway into a clustered MBL gateway is a cost effective way to build a scalable, reliable, high-performance MAP platform. The proposed clustered MBL WAP gateway architecture has been integrated with an efficient load balancing mechanism in the *Front-end Distributor* to make it possible to assign a request to the selected real gateway in the cluster that can offer the best service, as shown in Fig. 11 [34]. In this way, our MAP platform can provide a flexible, scalable, rapid-service-creation environment, where mobile Internet applications can easily dwell in the mobile agent-based *Wagent Server*. It also enables more mobile users to access mobile Internet services concurrently and efficiently via the clustered MBL WAP gateway.

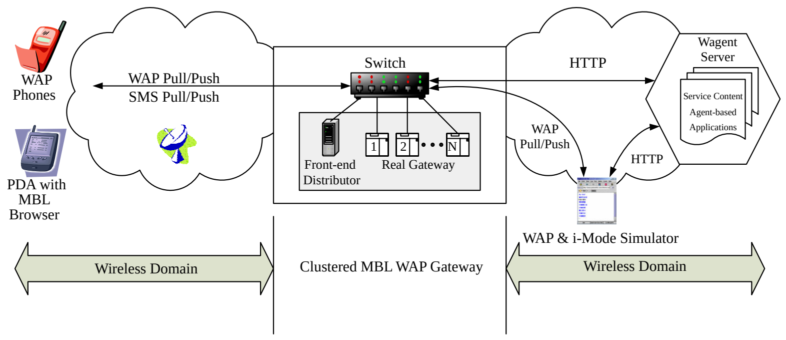

Fig. 11. Scalable MAP architecture.

5. CONCLUSIONS

We have designed and built a WAP-based, Push-enabled mobile Internet application platform, called MAP, which enables WAP clients to access legacy information systems, Intranets or WWW services. MAP also provides a flexible, scalable and rapid-service-creation environment in which applications can dwell. It includes four major components: the WAP Browser for mobile handsets, the WAP & i-Mode Simulator for desktops or notebooks, the WAP Push Proxy Gateway, and a distributed mobile agents system, the Wagent Server. It enables mobile users to easily access any Internet resources from any WAP device at anytime, anywhere. We have also shown the efficiency of the MBL Gateway by comparing its performance with that of other two existing gateways, and we have demonstrated the versatility of MAP by implementing several useful applications.

REFERENCES

1. K. Read and F. Maurer, “Developing mobile wireless applications,” *IEEE Internet Computing*, Vol. 7, 2003, pp. 81-86.
2. K. K. Tan, C. Y. Soh, and K. N. Wang, “Development of an internet home control system,” in *Proceedings of the IEEE International Conference on Control Applications* (*ICCA*), Vol. 2, 2002, pp. 1120-1125.

3. Open Mobile Alliance, http://www.openmobilealliance.org.
4. Open Mobile Alliance, "Wireless markup language version 2 specification," 2001.
5. Open Mobile Alliance, "WMLScript language specification," 2000.
6. J2ME^{TM} CLDC and Killobyte Virtual Machine (KVM), http://java.sun.com/products/cldc/.
7. Open Mobile Alliance, "Wireless datagram protocol specification," 2001.
8. Open Mobile Alliance, "Binary XML content format specification," 2001.
9. W3C, "Document Object Model (DOM) level 1 specification version 1.0," http://www.w3.org/DOM/, 1998.
10. Open Mobile Alliance, "WAP service indication specification," 2001.
11. Open Mobile Alliance, "WAP service loading specification," 2001.
12. Open Mobile Alliance, "Wireless application environment specification," 2002.
13. Kannel: Open Source WAP and SMS Gateway, http://www.kannel.org.
14. Jigsaw – W3C's Java Web Server, http://www.w3.org/Jigsaw.
15. The kAWT project, http://www.kawt.de.
16. Sun Microsystems, "Abstract window toolkit," http://java.sun.com/products/jdk/awt, 1997.
17. S. Wilson and J. Kesselman, *Java™ Platform Performance Strategies and Tactics*, Addison Wesley, 2000.
18. G. McCluskey, "Thirty ways to improve the performance of your Java™ programs," http://www.glenmccl.com/jperf/index.htm, 1999.
19. J. Shirazi, *Java™ Performance Tuning*, O'Relly, 2000.
20. Sun Microsystems, "JAVA^{TM} API for XML processing version 1.1," http://java.sun.com/xml, 2001.
21. Java based Constructor of Useful Parsers, http://www.cs.princeton.edu/~appel/ modern/java/CUP.
22. ETSI/TC, "Technical realization of the short message service point-to-point," Version 4.6.0, Technical Report Recommendation GSM 03.38, Version 5.6.0 (Phase 2+), ETSI, 1997.
23. Sun Microsystems, "Java™ remote method invocation specification," http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html, 2003.
24. Apache HTTP Server, http://httpd.apache.org/.
25. Z. Liu *et al*., "Traffic model and performance evaluation of web servers," http://www.inria.fr/rrrt/rr-3840.html, 1999.
26. K. Hung and Y. T. Zhang, "Implementation of a WAP-based telemedicine system for patient monitoring," *IEEE Transactions on Information Technology in Biomedicine*, Vol. 7, 2003, pp. 101-107.
27. A. Andreadis, G. Benelli, G. Giambene, and B. Marzucchi, "A performance evaluation approach for GSM-based information services," *IEEE Transactions on Vehicular Technology*, Vol. 52, 2003, pp. 313-325.
28. J. E. Pitkow, "Summary of WWW characterizations," http://decweb.ethz.ch/WWW7/1877/com1877.htm, 1998.
29. J. J. Huang, M. S. Chen, and H. P. Hung, "A QoS-aware transcoding proxy using on demand data broadcasting," in *Proceedings of the IEEE INFOCOM*, 2004, pp. 2050-2059.
30. V. Cardellini and P. S. Yu, "Collaborative proxy system for distributed web content

transcoding," in *Proceedings of the 9th International ACM Conference on Information Knowledge Management*, 2000, pp. 520-527.

31. T. H. Lin, K. C. Wang, and A. Y. Liu, "An efficient load balancing strategy for scalable WAP gateways," *Computer Communications*, Vol. 28, 2005, pp. 1028-1037.
32. T. Enderes, "Design and implementation of a multiplexing framework for the wireless application protocol," Technical Report, Institute for Communications Engineering, University of Karlsruhe, Germany, 1999.
33. ETSI, GSM 11.14: Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM-ME) Interface, http://www.etsi.org.
34. K. Sabatakakis, M. Zumbuhl, and S. Krotsch, *Mobile eCommerce*, Andersen Consulting, Zurich, Switzerland.
35. J. Leppanen, T. Laakko, and M. Kylanpaa, *Prototype Development for the WAP Application*, VTT Information Technology, Finland, 1999.
36. M. Cannataro and D. Pascuzzi, "A component-based architecture for the development and deployment of WAP-compliant transactional services," in *Proceedings of the 34th Hawaii International Conference on System Sciences*, 200, pp. 7073.

Hsuan-Hao Chen (陳軒皓) received the B.S. degree in Computer Science and Information Engineering from Catholic Fu Jen University, Taiwan, in 1999 and the M.S. degree in Computer and Information Science from National Chiao Tung University, Taiwan, in 2001. He is currently a project manager in the Network and Multimedia Institute, Information Appliance Technology Center, Institute for Information Industry. He has been working on software development for mobile devices as well as WAP technology implementation for several years. His research interests include distributed objects, embedded systems, wireless Internet, and mobile handset applications.

Kuochen Wang (王國禎) received the B.S. degree in Control Engineering from National Chiao Tung University, Taiwan, in 1978, and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Arizona in 1986 and 1991, respectively. He is currently a Professor in the Department of Computer and Information Science, National Chiao Tung University. From 1980 to 1984, he worked on network management, and design and implementation of the Toll Trunk Information System at the Directorate General of Telecommunications in Taiwan. He served in the army as a second lieutenant communication platoon leader from 1978 to 1980. His research interests include wireless (sensor) networks, mobile computing, and power management for mobile handheld devices.

Hung-Cheng Shih (施宏政) received the B.S. degree and the M.S. degree in the Department of Computer and Information Science from National Chiao Tung University, Taiwan, in 1997 and 2002, respectively. He is currently a Ph.D. student in the Department of Computer and Information Science, National Chiao Tung University. His research interests include mobile computing, wireless Internet, 3G telecommunication systems, and power management.