

DESIGN AND IMPLEMENTATION OF JAVA-BASED HTTP INPUT CHANNEL FOR INTEGRATED WLAN AND GPRS NETWORKS

Jun-Zhao Sun, Jukka Riekk, Marko Jurmu, Jaakko Sauvola
Department of Electrical and Information Engineering
P.O.Box 4500, FIN-90014 University of Oulu, Finland

Abstract – HTTP is the most crucial protocol enabling client-server applications. With the Internet converging and becoming mobile accessible, to adaptively manage HTTP connection is challenging. This paper describes the design and implementation of architecture for Java-based HTTP input channel for multi-access wireless networks. A HTTP input channel can adapt its behavior to the changing network status. Adaptation includes both sensible selecting and dynamic switching the underlying connection used, as well as to cope with disconnection situation. Experiments were performed in a GPRS-WLAN integrated environment to demonstrate the operational correctness of the architecture. Performance metrics were measured and analyzed.

1. INTRODUCTION

Client-server application is the dominant application in the Internet and this condition remains true when the Internet is becoming mobile accessible. HTTP is the most crucial protocol enabling client-server applications. Web browsing and Multimedia Messaging (MMS) are the representative applications on the basic of HTTP protocol with mobile devices. There is no doubt that HTTP will continuously be one of the most important protocols in future mobile Internet.

Next generation mobile communication system is emerging with heterogeneous wireless accesses as the leading feature. Network connectivity with varying quality of service (QoS) must be offered anytime and anywhere. However, networking capabilities, in terms of e.g. data rate, coverage, subscriber volume, supporting mobile velocity, anti-interference, suitable transmitting environment, are highly different from diverse access techniques. Moreover, wireless network QoS parameters are dynamically changing from time to time, in terms of reliability and availability, bandwidth, delay, jitter, response time, and packet loss.

The feasibility of heterogeneous access systems is preliminarily depended on the networking capabilities of mobile devices. To be multi-mode or multi band is the basic requirement for device to be connected into multiple networks. Currently the amount of terminals having equipped with than one network interfaces is increasing, e.g. laptop with LAN, WLAN, Bluetooth, an IrDA, mobile phones with GSM, GPRS, Bluetooth, and IrDA, etc. It has been well recognized that no single access technique can fulfill all the requirements. New method is needed to manage multiple access systems so

that diverse communication resources can form wireless connections in a simultaneous, collaborative, and complementary fashion

In this paper the design and implementation of architecture for Java-based HTTP input channel is described. HTTP input channel is an application layer solution for the management of multi-access wireless system for mobile device. A HTTP input channel is a unidirectional HTTP URL connection whose network interface under use can be automatically selected during the creation and seamlessly switched during the operation. As uploading is rarely needed, HTTP output channel was not included in our implementation.

The dynamic management of network connectivity is achieved through Channel Management Agent (CMA) by providing simple APIs to applications. Application simply gets a HTTP input channel from CMA and uses it to download remote file from e.g. a web server. If mobile device is equipped with multiple wireless connections, then the best connection can be selected to create the channel. During the input operations on the channel, the mobile device may move into or out of the overlapping area of multiple accesses. In this case the connection used by the channel can be seamlessly switched without being perceived by the application. When connection is temporarily unavailable input request to the channel can be suspended waiting for resumed later.

We have implemented a prototype with all the specified functions. Java is chosen as the programming language due to its feature of platform independency. PDA with integrated WLAN and GPRS connections is used as mobile device. Experiments were performed to demonstrate the operational correctness of our solution. The rest of the paper is organized as follows: Section II presents the overall architecture. Section III describes key points in the design. Section IV presents experimental setup and results. Finally, Section V concludes the paper with a discussion.

2. ARCHITECTURE

In multi-access situation where one mobile device is equipped with multiple wireless network adapters, there are three major issues that must be sufficiently resolved without the explicit control of applications, including:

- **Network interface selection.** At the moment when application requests a HTTP connection, the best interface should be used as underlying connection.
- **Disconnection treatment.** During the period that no valid network connection is available, input request from application should be suspended, and then resumed later when one interface becomes available.
- **Connection switching.** As application is reading from HTTP server, network status may change all the time. The reading operation should persist while ensuring the connection under usage is the best one.

To fulfill these requirements, new software components have to be incorporated into the mobile device. Figure 1 illustrates the architecture protocol stack. To enable HTTP input channel based new network application, two enabling components are added into the original protocol stack, Connection Monitor (CMor) and Channel Management Agent (CMA). CMor is responsible for the collection of transient network information of a local host. It also provides interfaces for CMA and applications to subscribe connection events and get notification later.

CMA is the core for adaptive network connectivity management. The management is realized by the creation and maintenance of HTTP input channel. A channel is a logical link between HTTP client and server applications. Each channel uses a specific type of network interface to transfer data from server to client. CMA provides a mechanism to re-evaluate the best connection for each channel. The connection for one channel can then be dynamically switched, while leaving the channel unchanged and so making applications imperceptible. When no network interface can fulfill the requirements, CMA can suspend the operation of the channel and wake it up later. Application can also get notification when a channel event occurs.

We adopt a pure client side solution, which means no special change is needed at HTTP server side. This greatly eases the deployment and reduces the changes needed to existing applications. To utilize the function provided, only extra software components need to be installed at mobile device side. There is no influence to existing legacy applications. The APIs provided by CMA is simple and operations to HTTP input channel is similar with HTTP URL connection interfaces. It is also easy to modify existing applications to utilize adaptive management functions.

3. DESIGN AND IMPLEMENTATION

3.1 Connection Monitoring

CMor module is used to collect and provide network context. Two different methods can be used for the obtaining and provision of corresponding network status and condition information, i.e. *explicit query* and *event subscription*. In any case, the connection monitor plays a reactive role, i.e. calmly waiting for query or passively reporting event occurrence.

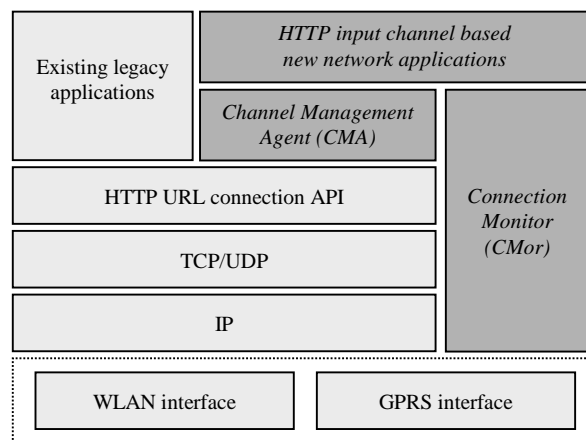


Figure 1. Architecture protocol stack.

CMA and applications can explicitly inquiry local network information with APIs provided by CMor. In CMor's implementation network connectivity information is obtained by invoking underlying IP Helper APIs. Host network information concerns available network interfaces enabled by system and configured with a valid IP address, including:

- Number of network interfaces, due to the fact that a modern mobile device may be equipped with multiple network interfaces, e.g. Bluetooth, IrDA, modem, Ethernet, WLAN, GSM, GPRS, etc.
- Connection list, consisting of connection indexes and corresponding connection name. The connection index forms the unique identification of each connection.
- Static information of each network interface, including name, type (e.g. PPP, Ethernet, Token Ring, etc.), IP address, gateway IP, speed.
- Operation status of each network interface, e.g. available, operable, connecting, connected, idle, sleeping, transmitting, receiving, unconnected, unreachable, disabled, etc.
- Packet statistics information, i.e. received, sent, discarded, and dropped IP packet number on each network interface.

Event-driven method is to subscribe some special network events and then be informed when they happen. CMor allows the subscription of the following connection events that can occur on local host.

- “**connection up**” event occurs when a new connection is available (i.e. a new network interface is available and assigned with a IP address), due to e.g. plugging in a PCMCIA card or moving to a particular area.
- “**connection down**” event happens when a existing connection is not available any more, due to e.g. pulling out a adapter card or moving out of a area.
- “**address changed**” event happens when the IP address assigned to a network interface has been changed, due to e.g. handoff between access points.

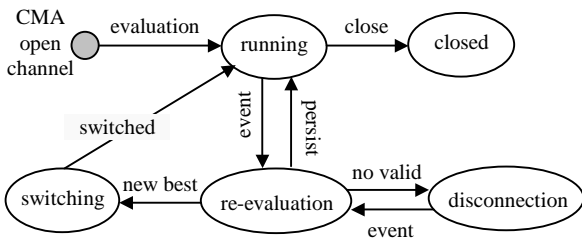


Figure 2. Channel execution states and transition

CMA subscribes all the three connection events from CMor on behalf of channels and notifies all channels when events occur. Channel can decide its behavior accordingly.

3.2 Channel Creation

The core adaptation mechanisms for connectivity management are mainly realized by CMA by adaptively maintaining channel's life cycle. A channel is at different states during the operation. Figure 2 illustrates the execution states of a channel. Application can obtain a HTTP input channel by simply invoking CMA's *openHTTPChannel* interface. An evaluation process is then launched to select currently the best local connection to be used for this channel.

CMA provides an interface for connection evaluation and re-evaluation. Evaluation is the process of finding the best local connection at the moment according to context information and input parameters. Connection evaluation is done both at channel creation and at run time when connection events occur, as shown in Figure 2. An open framework is created for connection evaluation. This framework allows a wide set of context information being taken into consideration, including e.g. user preferences, application requirements, network condition, device status, etc. In our implementation connection with the best speed will be selected.

If the creation succeeds, CMA adds the new channel into its local maintained channel list. State for the channel is then set as "running". A channel can later be removed from the channel list in case the application closes it at some time. Each channel contains a private data object for the storage of channel information including type, connection index, IP address, traffic class, as well as server URL, etc.

CMA uses HTTP URL connection API for creating a HTTP channel. If device has multiple active connections, the problem of how to open the HTTP connection exactly through a preferred connection arises. There will be no benefit to us if we just leave the problem of determining local connection to be used to system. For example when a WLAN connection is up besides the old GPRS connection, we may hope to open our HTTP channel through the WLAN connection for better bandwidth. However the system may automatically open the channel by using GPRS.

The system is forced to use the preferred connection by changing the default gateway of the system to the routing

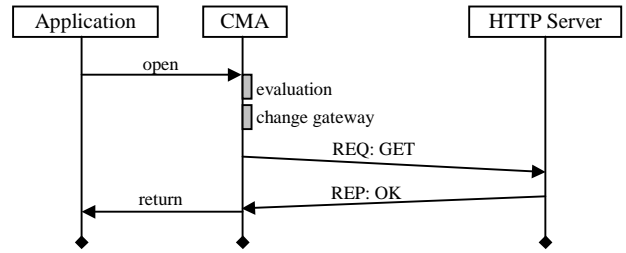


Figure 3. HTTP input channel creation

table. Routing table is searched for the gateway entries. Each gateway entry has an IP destination of "0.0.0.0". All the gateway entries will then be temporarily changed with metric to 2 except the one for the selected connection whose metric value is set to 1. After this treatment the HTTP connection will be established with the selected connection as wished. Figure 3 depicts the message sequence of HTTP channel creation.

Application can subscribe channel events and get notification later when channel events occur. Application can then adjust its behavior accordingly. There are two types channel events. One is "disconnection" event that occurs when reevaluation indicates that there is no valid connection available. Another is "local switched" events that occurs when the system changes local connection used by the channel.

3.3 Channel Connection Switching

If, by evaluation, no valid connection is available, the channel will enter "disconnection" state. In this state reading operation is blocked and channel will not be waked up for re-evaluation until a new event occurs. On the other hand if new best connection is found, the channel will initiate a connection switching session.

HTTP input channel is working in a standalone fashion, in which only client side connection and input operations are controllable. First the same operation described in Section 3.B is used to change currently default gateway of the system by manipulating routing table entries. Then a new HTTP URL connection is established through the new best network interface. In order to continue the file download from the break point A "RANGE" property is set before the real connection (i.e. HTTP GET request). During the switching Input operations are blocked.

Usually in Java stream methods buffered input stream is supported by the HTTP GET request method. That is, input operation gets more bytes from remote file than required by application, and stores additional data into local buffer for further reading. Therefore system needs not make real input operation for each read request by the application. When channel switches its connection to a new one, some additional bytes in the old input buffer may be still available. HTTP channel maintains a local read buffer. Additional available data is then drawn out and stored into this local buffer before the actual HTTP connection to the remote object is made with

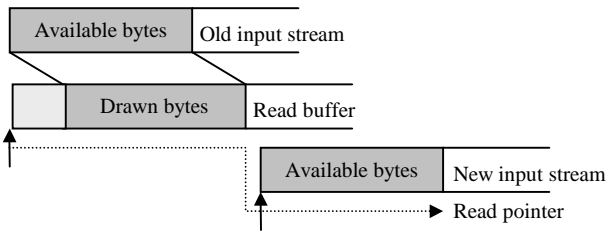


Figure 4. Available bytes drawn out

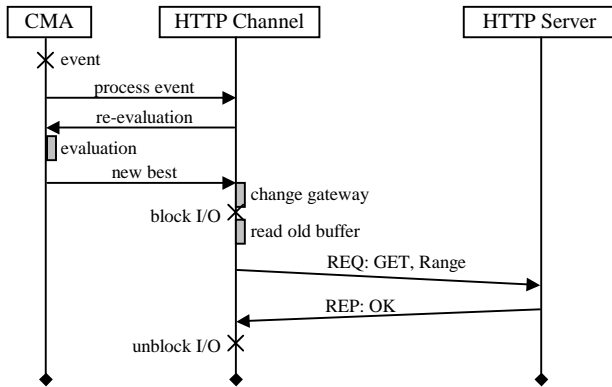


Figure 5. HTTP input channel switching

the new selected connection. After the switching read operation will first try the local buffer to get application input data, as shown in Figure 4. Message sequence for HTTP channel switching is illustrated in Figure 5.

4. EXPERIMENTS

We have implemented the whole architecture. The implementation contains a set of Java classes and a native library. Current Java implementation is based on Sun JDK 1.1.8. This is to ensure the best compatibility because most Java Virtual Machines (JVMs) for mobile devices are still at early version. The native library is used to access native IP Helper APIs and interacts with CMor Java class by Java Native Interface (JNI). We have implemented and tested the Dynamic-Link Library (DLL) files for both Pocket 2002 and Window 2000. Experiments were performed with the implementation to demonstrate the operational correctness of the architecture as well as to measure and analyze performance metrics. According to our design during HTTP input channel connection switching packet loss is avoided. So packet delay is the major performance overhead.

4.1 Experimental setup

The test environment consists of a user device (PDA) and a public web server interconnected through two accesses and the Internet, as illustrated in Figure 6. User device is COMPAQ iPAQ 3970 running Windows Pocket PC 2002. IBM J9 is used as Java JVM. A Lucent Orinoco Gold PCMCIA card was equipped for Wi-Fi wireless access. We used panOulu [1] as the WLAN network. A NOKIA 7650 mobile phone serves as a GPRS modem connected with PDA

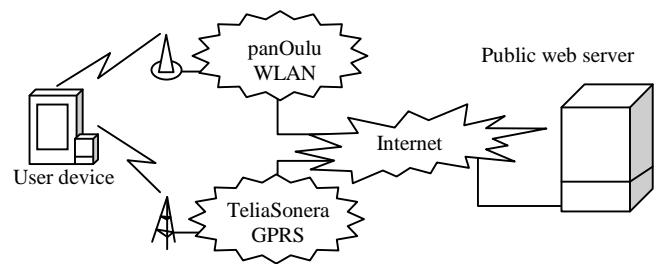


Figure 6. WLAN-GPRS integrated environment

through Bluetooth. TeliaSonera commercial GPRS connection was used. The public web server was used to test HTTP input channel, from where a MP3 music file is to be downloaded.

Experiment is performed that user downloads a MP3 music file (372 Kbytes) from the public web server with PDA. The URL of the music file is at:

<http://www.mwscomp.com/sounds/mp3/finland.mp3>.

Buffer size is set to be 2KBytes. Four scenarios were used in the experiment, including

1. Both of the two access networks are available and no connection event occurs, to simulate the situation of download at hotspot.
2. Only GPRS access is available to the end, to simulate the scenario of outdoor download when no WLAN.
3. Both of the two access networks are available at the beginning and WLAN is lost a moment later, to simulate the case of moving out of hotspot.
4. Only GPRS access is available at the beginning and WLAN access is available a moment later, to simulate the scenario of moving into hotspot area.

A timer is started after the channel is opened. Then to each data input the time and data size are recorded. Note in the experiment no physical movement was performed. Rather, network connection was changed manually by plug/pull PCMCIA card in/out iPAQ to simulate the movement into/out WLAN covered area.

4.2 Experimental results

Table 1 summaries the results for HTTP input channel creation, with plain HTTP URL connection as reference. A bit of time overhead occurs due to the operations on connection evaluation and changing default gateway.

Figure 7 indicates the results of all four scenarios in HTTP channel experiment. The concrete values are summarized in Table 2. In Figure 7 (a) as scenario 1 WLAN connection was selected with a setup time (time until the first packet) of 1.28s and total time 44.69s. In scenario 2 GPRS was used with the setup time as 4.63s and total time 87.81s. In Figure 7 (b) result of scenario 3 shows the switching from WLAN to GPRS. Packet delay time (time between last packet through old connection and first packet through new connection) is 4.16s. Finally in scenario 4 connection switching from GPRS to WLAN occurred. Packet delay is 1.33s, with buffered data of 1448 bytes were drawn out from old input stream.

Table 1 HTTP input channel creation

	WLAN	GPRS
Plain	0.94 s	4.49 s
Channel	1.28 s	4.63 s

Table 2 HTTP input channel connection switch

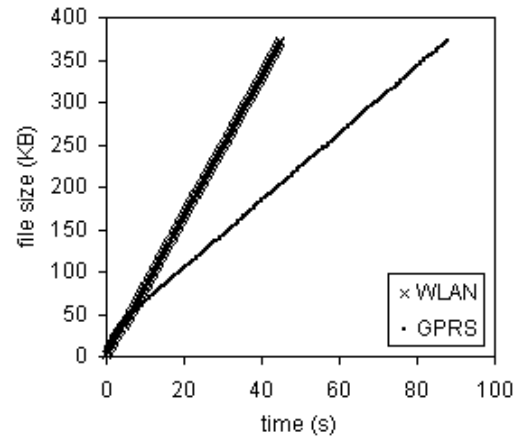
	Packet delay	Drawn bytes
WLAN to GPRS	4.16 s	0
GPRS to WLAN	1.33 s	1448

It is easy to understand that to HTTP channel connection active switch (switch due to “*connection up*” event) is quicker than passive switch (switch forced by “*connection down*” event). To detect a connection event at network level needs a relatively long time. During the period original connection can still be used if available. Another reason to this is that in our experiment WLAN access network has a shorter setup time than GPRS. Result shows to HTTP channel connection switching time are on the order of access setup time.

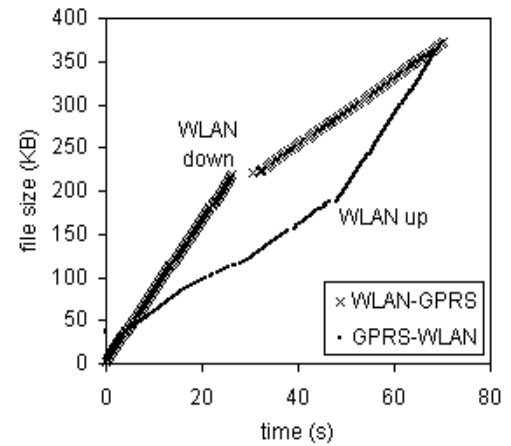
5. DISCUSSION AND CONCLUSION

Our solution on adaptive HTTP input channel is closely related to the research of mobility management. Plenty of research has been carried out on mobility management. Mobile IP [2, 3] manages terminal roaming at network layer based on the extension to IPv4 or IPv6, which provides transparent mobility support to application. Vertical handoff [4] can be used for heterogeneous connections like WLAN and GPRS. In [5] an end-to-end host mobility solution is proposed at transport layer based on TCP extension. There are some proxy/gateway-based solutions at transport layer [6] as well. Research has also been done for application level mobility solution based on SIP [7]. Our solution focuses on HTTP protocol and manages mobility at application level with newly introduced concept: channel. There is no change needed for HTTP protocols, which eases the deployment. Moreover, connection switching occurs on the granularity of channel instead of device, which means multiple (most possibly heterogeneous) connections can be simultaneously used by one user device for different channels

The architecture proposed aims to provide adaptive connectivity management to HTTP based network applications. By employing HTTP input channel both disconnection and connection switching are enabled. Experiments show there is no packet loss during connection switching and packet delay is on the order of access setup time for HTTP channel. Currently CMA supports the creation of HTTP input channel and TCP related channels. While our design provides an open platform for extension, we are extending the CMA so that more channel types will be supported later, including e.g. connected UDP channel, multicast channel, HTTP output channel, as well as virtual port based TCP related channels. Moreover, rich context information will be taken into consideration to achieve more intelligent evaluation on the best connection to be used.



(a) no switch



(b) switch

Figure 7. HTTP channel experiment result

6. ACKNOWLEDGMENT

Financial support by National Technology Agency of Finland is gratefully acknowledged.

7. REFERENCES

- [1] panOulu public access network, <http://www.panoulu.net>.
- [2] C. Perkins, ED., “IP Mobility support for IPv4,” IETF RFC 3344, August 2002.
- [3] D. Johnson, C. Perkins, and J. Arkko, “Mobility support in IPv6,” IETF Internet Draft, draft-ietf-mobileip-ipv6-24.txt, June 30, 2003.
- [4] Mark Stemm and Randy H. Katz, “Vertical handoffs in wireless overlay networks,” *ACM Mobile Networks and Applications (MONET)*, vol. 3, no. 4, 1998: 335-350.
- [5] Alex C. Snoeren and Hari Balakrishnan, “An end-to-end approach to host mobility,” *Proc. 6th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, Boston, Massachusetts, United States August 2000: 155–166.
- [6] P. Bhagwat, D.A. Maltz, and A. Segall, “MSOCKS+: an architecture for transport layer mobility,” *Computer Networks*, Vol. 39(4), 2002: 385–403.
- [7] H. Schulzrinne and E. Wedlund, “Application layer mobility support using SIP,” *ACM Mobile Computing and Communications Review*, vol. 4, no. 3, 2000: 47-57.