A Cross-layer Signaling and Middleware Platform for Multi-interface Mobile Devices

Yung-Chien Shih[1], Kai-Cheng Hsu[2], and Chien-Chao Tseng[1]

[1] Department of Computer Science, National Chiao Tung University,
1001 Ta Hsueh Rd., Hsinchu, Taiwan 300, ROC
[2] WiMAX Development Team, MediaTek Inc.,
No. 1, Dusing Rd., Hsinchu Science Park, Hsinchu, Taiwan 300, ROC
ycshih@csie.nctu.edu.tw, mojahsu@gmail.com, cctseng@cs.nctu.edu.tw

Abstract. This paper presents a middleware platform approach to provide Cross-layer Signaling and Network Event Notification mechanisms for network-aware applications. Because a mobile device may be equipped with multiple network interfaces to attach different network as it moves, a network-aware application running on the mobile device must react promptly to the changes of network environment. In order for the applications to detect network changes, the proposed middleware platform provides APIs for setting up network configuration and acquiring low-layer statuses. Therefore, an application can detect network changes promptly via Network Event Notification mechanism. We also use two network-aware applications, namely a Mobility Manager and a Modified Kphone, as examples to demonstrate the effectiveness of our middleware platform.

Key words: Middleware, Cross-layer Signaling, Network Event Notification, Network-aware Application, Handover

1 Introduction

With the advance in wireless techniques, mobile devices, such as *Personal Digital Assistant (PDA)*, smart phone, and tablet PC, has become a popular electronic product recently. A mobile device may be equipped with multiple wireless network interfaces, such as WLAN, GPRS, or 3G adaptors, to attach to different networks as it moves. Therefore, an application running on the mobile device may visit different networks and encounters the changes in bandwidth, delays, or IP addresses. In order to tackle network fluctuations, network-aware applications that can adapt themselves to the changes in network environment have now become a major research topic in recent years.

A network-aware application may need to issue system calls periodically to retrieve lower-layer statuses, such as IP addresses, entries of routing table, and connectivity of network adaptors. However, the network statuses normally do not change frequently. Periodical system calls with short intervals may waste system resources for getting the same information. On the other hand, with long intervals between system calls, the applications can not react to the changes of network environment promptly.

Furthermore, a mobile device with multiple network interfaces needs a mobility manager to monitor statuses of network adaptor and perform handover decision accordingly. In order to acquire the statuses and conduct a handover, a mobility manager needs to use system calls to communicate with underlying network protocol stacks, such as link layer, network layer, or transport layer. However, the use of system calls is not only error-prone but also not portable.

In order to solve above problems, we proposed a software platform for the development of network-aware applications. The platform adopts a middleware [1] approach and provides *Application Programming Interfaces (APIs)* for the application to use Cross-layer Signaling Mechanism. Based on the mechanism, an application can interact with the lower-layer network protocols to acquire network statuses and manage network interfaces. Besides, the platform also provides an Event Notification Mechanism for an application to register interested events of network changes so that the middleware can notify the application immediately when an event of interest occurs.

We also use network-aware applications, namely a Mobility Manager and a Modified Kphone [2], as two examples to demonstrate the effectiveness of out proposed platform. The Mobility Manager can monitor statuses of multiple network interfaces simultaneously and perform handover decision accordingly. The Modified Kphone is a *Voice over IP (VoIP)* application with a new handover decision module that can interact with our proposed platform.

The rest of this paper is organized as follows. In Section 2, we describe previous research on middleware and cross-layer design. Then in Section 3, we present the architecture of our middleware platform, including a Cross-layer Signaling Mechanism and an Event Notification Mechanism. We show the two examples of network-aware applications through our proposed platform in Section 4. Finally, Section 5 concludes the paper.

2 Related Work

The handover issues of a mobile device with multiple interfaces are popular research topic recently. Although Mobile IP [3] mechanism can achieve heterogeneous handover, a mobile device cannot perform the handover smoothly. A reason of this problem is that an application cannot acquire promptly the underlying statuses, such as network link up, link down or wireless signal strength, because each network protocol layer works independently with each other. Therefore, in recent years, several researchers have proposed middleware or cross-layer mechanism for the upper-layer application to acquire the statuses of the lower-layer protocols.

2.1 Middleware

Many researchers has proposed middleware approaches to supporting fast handoff in heterogeneous network environment [4], [5], [6], [7], [8], [9]. In the research,

the middleware, provide interaction mechanisms among applications and lower-layer protocols. Accordingly, the middleware is situated between applications and the *Operating System (OS)* as shown in Fig. 1. It hides the complexity of error-prone system calls, OS APIs, from applications by encapsulating the complex system calls in into simple middleware APIs to provide high level functions for applications. Therefore, applications need not handle the connectivity and heterogeneity of the underlying networks. In stead, they can manage the networks and react to the changes of networks accordingly through middleware APIs.

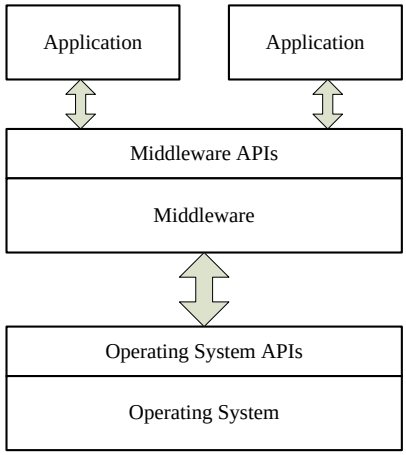

Fig. 1. The design concept of middleware was introduced in some research. The middleware is implemented between application and operating system, and then it provides high level functions for application to interact underlying protocol stacks.

If a programmer develops an application via the middleware, there are three benefits to develop applications via the middleware: easy porting, quick development and error avoidance. For application porting, because developing an application via middleware APIs can hide heterogeneity of OS, programmers need not modify any program codes when transplanting applications to another OS. For example, *Java Virtual Machine (JVM)* is one illustration of the middleware concept. It allows the same program code to run on different OSs. As for the application development and error avoidance, since middleware APIs provide simple interfaces for programmers to acquire and configure statuses of lower-layer protocols, programmers need not care for the details of system calls. Consequently, programmers can develop applications quickly and correctly.

Although many proposals adopted middleware concept to perform handover decision in a heterogeneous network environment, applications can not detect the changes of network environment immediately. We need other mechanisms to complete our middleware platform.

2.2 Cross Layer Design

According the report of research [10], traditional network protocol that is used on mobile devices is not efficient because protocol stacks are working independently and thus those protocol stacks do not know statuses of other stacks. To solve the problem, the Cross Layer Feedback method has been mentioned in some previous work [11], [12]. This method provides an interaction mechanism for a protocol of a layer to share statuses with one of another layer. For example, a link layer protocol can share statuses with an application layer protocol and thus applications can adjust transmission rate according to the link statuses.

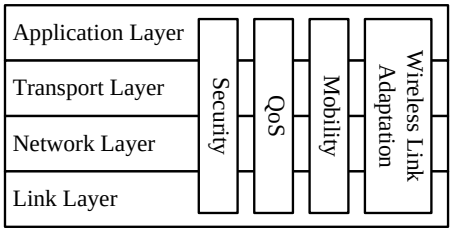

Fig. 2. The concept of Cross-Layer Notification Mechanism has been mentioned in past research. This mechanism allows a network protocol to inform other protocols the changes of network statuses proactively.

The Cross-Layer Notification Mechanism has been mentioned in past study [13]. This study suggested that a network protocol need a mechanism to inform other protocols about changes of network statuses. Therefore, cross-layer architecture, shown in Fig. 2, was proposed to satisfy needs of different applications, such as security, QoS or mobility requirement. For example of wireless link adaptation, when link quality of current attachment network is changed, the link layer will notify an application layer program of the link quality of each wireless adaptor or the transport layer of the maximum transmission rate. Furthermore, applications can inform the link layer to switch wireless adaptor.

2.3 Driver-Level Network Event Notification

The Driver-Level Network Event Notification Mechanism has been mentioned in our past research [14]. This mechanism adopted signaling mechanism to notify applications implemented in user space. Furthermore, it also modified OS scheduling algorithm to eliminate the needs for an application to issue system calls periodically to retrieve low-layer statuses. Implementation of the mechanism includes three major parts: (1) Event Notification, (2) Process Management and (3) Scheduler. In our implementation, before a process in the kernel space returns to the user space, the mechanism will check whether the registered events of this process occur or not. If the registered event occurs, the mechanism will call the

corresponding procedure firstly. In this paper, we adopted our proposed event notification mechanism and then modified some algorithms and data structures to satisfy requirements of our goal.

3 System Architecture

In this section, we will detail system architecture of the proposed middleware platform including Cross-layer Signaling and Event Notification mechanisms.

3.1 Middleware Platform and Cross-layer Signaling

System architecture of proposed middleware platform can be divided into two major parts that include user space and kernel space. The middleware was implemented in user space and between application layer and underlying network layers and provided APIs for the network-aware applications, such as mobility manager and modified Kphone, to interact with underlying stacks of network protocol as shown in Fig. 3. For interaction between different network protocol stacks, the proposed middleware platform must provide Cross-layer Signaling Mechanism including statuses of underlying stacks acquisition, control messages dispatch and events notification for the applications. Therefore, the APIs are divided into three kinds including control, query and event interfaces. The control interface is used to notify underlying protocol stacks of changing statuses, such as adding or deleting entries of routing table, changing default gateway or attachment *Access Point (AP)*. The query interface is used to acquire specific status of underlying protocol stacks, such as wireless signal strength, IP address configurations or data transmission rate. The event interface allows the network-aware application to register interesting events and get event notification mmediately when an interesting event occurs. Our implementation of event interface will be detailed in next subsection.

On the other hand, because the middleware must help the network-aware applications to interact with underlying protocol stacks, control and query messages received from applications need to translate into corresponding system calls. Therefore, we implemented the system call functions in proposed middleware, called Middleware Core, to interact with specific protocol layer located in kernel space. This part adopts cross-layer signaling design to order specific layer to do something and acquire statuses of specific layer. For example, an application can acquire wireless signal strength of WLAN adaptor or configure IP address directly. Furthermore, the Middleware Core will maintain some data structures, such as registered event tables and event queues, for applications to query or use.

To adopt proposed middleware platform, complex system calls can be reduced to simple APIs and thus an application can easy to use. Furthermore, to acquire and configure low-layer statuses via the Middleware APIs, a network-aware application can be developed quickly and error-less, and thus application developer can pay attention more to design handover policies.

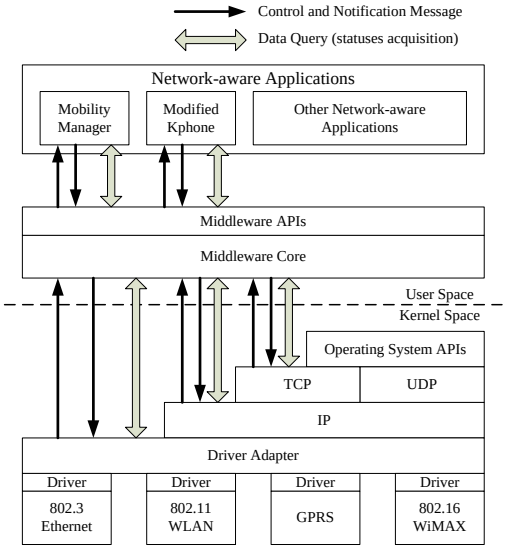

Fig. 3. The proposed middleware platform can be divided into two main parts: user space and kernel space. The middleware was implemented between the network-aware applications and underlying protocol stacks located in kernel space. Therefore, the Middleware APIs allow the network-aware applications to acquire and configure statuses of underlying protocol stacks.

3.2 Network Event Notification Mechanism

The event interface includes two kinds of method to notify the network-aware applications: synchronous and asynchronous process. In the synchronous process, a network-aware application must use Middleware API to initiate one or more than one event queues, and then register those queues and interesting events, called registered event, in Middleware Core as shown in Fig. 4. To illustrate this process, in the example program of synchronous process, an application must use the *win_event_init* function call and configure the parameter as "`NULL`" to initiate an event queue. Therefore, the application can use the *win_event_register* function call to register interesting events in Middleware Core, and then the Middleware Core will maintain a *Registered Event Table* including all interesting events of the application, such as LINK_UP, LINK_DOWN and NETDV_UP in Registered Event Table 1. When an event occurs, the Middleware Core will check all of Registered Event Tables and then push the event to corresponding event queues. Finally, the application can use *win_check_event* function call to poll event queues periodically.

In the asynchronous process, an application must create an event handler to prepare for event notifications. Using this process, the application can get event notification immediately when an event occurs. For example, an event handler that is named for "`my_event_handler`" must be created firstly, and then the application uses *win_init_event* function call and configures the parameter as the

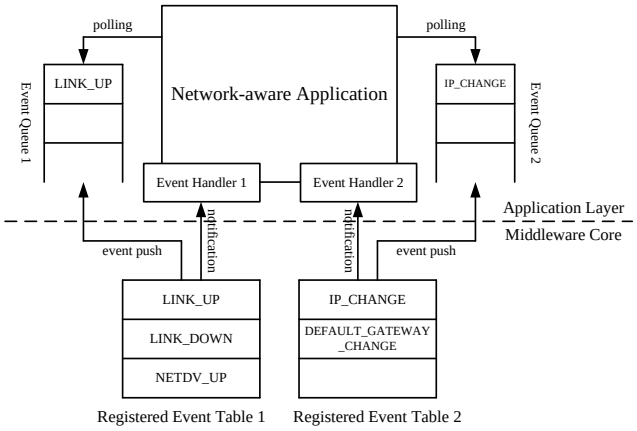

Fig. 4. There are two kinds of process of Event Notification Mechanism including synchronous and asynchronous. In the synchronous process, the Middleware Core will push event to event queue when an interested event occurs. On the other hand, in the asynchronous process, the Middleware Core will notify event handler when an interested event occurs and then the network-aware application can know the occurrence of an event immediately.

handler name to initiate asynchronous process as shown in example program of asynchronous process. Therefore, the application can use *win_event_register* function call to register interesting events similarly. The Middleware Core will dispatch event notification to corresponding event handlers when an event occurs so the application knows event occurrence and the corresponding handler procedure will run immediately.

In the kernel level, we adopted the Driver-level Network Event Notification Mechanism that is developed by our research partner and mentioned in Section 2 to support our Event Notification Mechanism. Furthermore, we had added novel events, such as ROUTING_TABLE_CHANGE, to extend the notification mechanism so the middleware platform can be used to help network-aware application that it gets interesting event notifications correctly and immediately.

An Example Program of Synchronous Process

```
event_descriptor = win_event_init(NULL);

win_event_register(event_descriptor, NETDEV_UP);
win_event_register(event_descriptor, IP_CHANGE);
...

if(win_check_event(event_descriptor, wevent)
   == R_HAVE_EVENT) {
   ...
}
```

An Example Program of Asynchronous Process

```
void my_event_handler(struct win_event* wevent) {
   printf(Event Happen!!\n);
}

int main() {
   ...
   event_descriptor = win_event_init(my_event_handler);
   win_event_register(event_descriptor, NETDEV_UP);
   win_event_register(event_descriptor, IP_CHANGE);
   ...
}
```

4 Network-aware Application

Based on our middleware platform, a program developer can develop a network-aware application quickly and easily. In this section, we will demonstrate two network-aware applications including Mobility Manager and Modified Kphone.

4.1 Mobility Manager

A mobility manager for mobile device should include three major functions: display of network status, network configuration and handover policy. Therefore, we implemented the Mobility Manager application that used Middleware APIs to acquire and configure statuses of underlying protocol stacks. For example, in the Fig. 5, wireless signal strength of APs can be displayed on a single graphic interface. Furthermore, a user of mobile device can use a graphic configuration interface to set attachment AP and choose acquisition method of IP address as shown in Fig. 6.

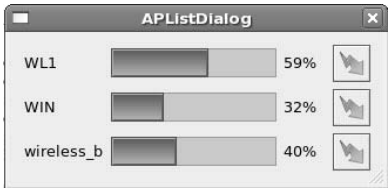

Fig. 5. This graphic interface will display wireless signal strength of APs that a mobile device can attach. For example, this figure displays three signal strengths of AP including WL1, WIN and wireless_b.

For mobile device handover, we need to know the changes of network environment and then mobile device chooses another network to attach according

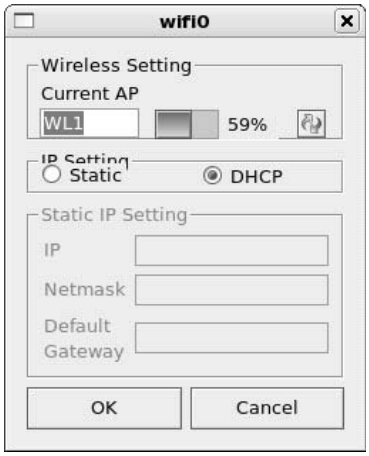

Fig. 6. This graphic interface allows user of mobile device to configure current attachment AP and choose acquisition method of IP address, such as static or DHCP.

user preference if current attachment network is unreachable or low quality. In our implementation, the Mobility Manager allows user to choose manual or automatic handover and to assign preferred interface as shown in Fig. 7. User can configure a profile of handover policy that records when handover can be performed and what something is needed to do in handover, if the Mobility Manager be configured automatic handover mode.

Fig. 7. User of a mobile device can use this graphic interface to choose manual or automatic handover and to assign preferred interface.

In summary, a mobile device can acquire network statuses, such as current IP address and signal strength of APs, and configure those network statuses easily if the Mobility Manager application runs on this mobile device. Furthermore, the mobile device can make handover decision automatically according

network environment changes, such as link quality or reachable network, and user preference.

4.2 Modified Kphone

The Modified Kphone is another example of network-aware application that is sourced from a VoIP application called Kphone and is modified to perform handover via the Middleware APIs. We need to program some new procedures to make handover decision because the original Kphone application cannot support handover. In the Kphone communication, a *Mobile Node (MN)* and a *Correspondent Node (CN)* are transmitting voice data via *Real-time Transport Protocol (RTP)* packets to another. We need to ensure that *Synchronization Source Identifier (SSRC)* in RTP header is identical when the MN changes its IP address because same SSRC in RTP header implies same connection between MN and CN.

In our implementation of Modified Kphone, we divided handover procedure into four steps as shown in Fig. 8. First, a MN receives a network event notification, such as wireless signal strength below a threshold, and then it triggers network layer handover that includes changing attachment network, acquiring a new IP address and configuring default gateway. Second, the MN uses the new IP address and original SSRC to send Re-invite message to CN. Third, the CN will change destination IP address of sending packets according source IP address of received packet and the SSRC when it receives the Re-invite message. Finally, the CN will send ACK message to the MN and thus an application layer handover is completed.

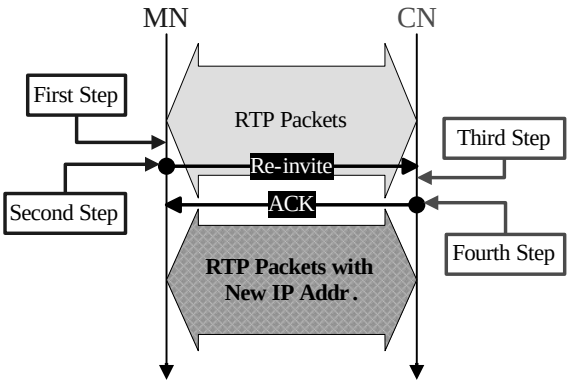

Fig. 8. The handover procedure of Modified Kphone can be divided into four steps. If a network event notification be sent to Modified Kphone and it decides to perform handover, this handover procedure that includes first, second, third and fourth step will be run.

In summary, our Modified Kphone application can perform handover quickly and correctly because the middleware platform can provides layer-2 trigger and some detail statuses for an application. Furthermore, a program developer can pay attention more to design of handover policies as a result of the middleware platform provides several mechanisms for the developer to implement applications.

5 Conclusion

In this paper, we designed and implemented a middleware platform that includes Cross-layer Signaling and Network Event Notification mechanisms. The middleware was implemented in user space and then it provided Middleware APIs for applications to configure and acquire statuses of underlying protocol stacks. Based on Cross-layer Signaling mechanism, control and query messages can order each protocol stack to do something directly. Beside, the Network Event Notification can help application that it is aware of changes of network environment immediately.

We also demonstrated two examples of network-aware application that include Mobility Manger and Modified Kphone. Those examples illustrated two kinds of Event Notification method and how to use in our middleware platform. According our illustration, if a network-aware application is developed via our middleware platform, this application can be developed quickly and easily.

In future works, we will continue to extend novel network events when a new type of network adaptor is created. Furthermore, we consider developing a handover analysis tool via the middleware platform and then the tool can help programmer to determine cause of handover delay.

References

1. Bakken, D.: Middleware. `http://www.eecs.wsu.edu/~bakken/middleware.pdf`
2. Kphone Project, `http://sourceforge.net/projects/kphone`
3. Perkins, C.: IP Mobility Support for IPv4. IETF RFC3344, Nokia Research Center, August (2002)
4. Sun, J., Riekki, J., Jurmu, M., Sauvola, J.: Adaptive Connectivity Management Middleware for Heterogeneous Wireless Networks. In: IEEE Wireless Communications, December (2005)
5. Sun, J., Tenhunen, J., Sauvola, J.: CME: a middleware architecture for network-aware adaptive applications. In: Proceedings 14th IEEE PIMRC2003, Beijing, China (2003)
6. Hawick, K.A., James, H.A.: Wireless Issues for a Mobility Management Middleware. In: CCN2002, August (2002)
7. Tian, Y., Frank, S., Tsaoussidis, V., Badr, H.: Middleware Design Issues for Application Management in Heterogeneous Networks. In: Networks 2000 (ICON 2000)
8. Li, B., Nahrstedt, K.: A Control-Based Middleware Framework for Quality of Service Adaptations. In: IEEE Journal of Selected Areas in Communication, 17(9): pp. 1632–1650 (1999)

9. Kreller, B., Park, A.S.B., Meggers, J., Forsgren, G., Kovacs, E., Rosinus, M., Siemens A.G.: UMTS: A Middleware Architecture and Mobile API Approach. In: IEEE Personal Communications, April (1998)
10. George, X., George, C.P.: Internet Protocol Performance over Networks with Wireless Links. In: IEEE Network, 13(4): pp. 55–63 (1999)
11. Clark, D.D.: The Structuring of Systems using Upcalls. In: ACM Symposium on Operating Systems, pp. 171–180 (1985)
12. Cooper, G.H.: The Argument for Soft Layer of Protocols. In: Tech. Rep. Tr-300, Massachussets Institute of Technology, Cambridge, May (1983)
13. Carneiro G., Ruela, J., Ricardo, M.: Cross-Layer Design in 4G Wireless Terminals. In: IEEE Wireless Communications, 11(2): pp. 7–13, April (2004)
14. Chou., T.J.: Design and Implementation of Driver-Level Network Event Notification Mechanism in Linux. In: Thesis in Wireless Internet Lab., National Chiao Tung University, Hsinchu, Taiwan (2005)