

## Short Paper

---

# Tapster: Transparent Proxy Redirector for Mobile Computing

TZU-CHI HUANG, CE-KUEN SHIEH, BO-YANG LAI AND YU-BEN MIAO\*

*Department of Electrical Engineering  
National Cheng Kung University  
Tainan, 701 Taiwan*

*\*Computer and Communications Research Laboratories  
Industrial Technology Research Institute  
Hsinchu, 310 Taiwan*

In mobile computing, the proxy-based software may access the proxy far from the visited network rather than the nearby proxy. Accordingly, the proxy-based software's communication hardly gets advantages from the proxy and may suffer disturbances. We propose the Tapster as the transparent proxy redirector for mobile computing. The Tapster can automatically redirect applications' requests to a nearby proxy without software configuration and modification. Accordingly, developers can focus on designing their applications without considering mobility and compatibility issues. The Tapster supports various network applications not merely WWW applications. Besides, the Tapster works at the mobile device to maintain security mechanisms, make the optimal communication path, and mitigate the load of network devices. We implement the Tapster in Windows XP and have experiments. The experiments show that (1) the Tapster has the negligible overhead, (2) the Tapster redirects applications' requests to the nearby proxy to shorten communication delay, and (3) the Tapster works without software configuration and modification.

**Keywords:** proxy, tapster, packet redirection, mobile computing, network address translator, transparent proxy redirector, proxy-based software

## 1. MOTIVATION

Proxies bring the Internet many advantages [1], *e.g.* controlling access, conserving network bandwidth, reducing communication delay, and alleviating origin servers' loads. Therefore, various proxies are developed to meet requirements of network applications, *e.g.* a HTTP proxy for WWW, a FTP proxy for file transmission, and a SOCKS proxy for firewall traversal. Besides, most software nowadays can communicate through a proxy and is referred to as the proxy-based software.

However, the emergence of mobile computing technology, *i.e.* using portable devices to access the Internet via wireless communication, has a negative impact on the proxy-based software. Designed for fixed network environments, the proxy-based software is unaware of the underlying mobile device's movement. When the mobile device

---

Received December 14, 2005; revised March 16 & May 23, 2006; accepted June 26, 2006.  
Communicated by Ten-Hwang Lai.

moves, the proxy-based software on it still accesses the Internet through the proxy configured in the software (abbreviated as *configured proxy*) rather than through the nearby proxy. Because the *configured proxy* may be far from the visited network, the proxy-based software's communication hardly gets advantages from the proxy and may suffer disturbances, *e.g.*, getting congested, suffering long delay, encountering a firewall's obstacle, or being intercepted. Accordingly, the proxy-based software should have a way to use the nearby proxy instead of the *configured proxy* in mobile computing.

In some cases, switching to a nearby proxy may not always get the advantages. For example, a nearby fresh proxy may cause a communication delay due to cache miss while a nearby busy proxy may become a communication bottleneck to worsen the communication quality. However, the cases happen to the *configured proxy* as well because the proxy-based software does not switch to another proxy. Nevertheless, using a nearby proxy is unavoidable to mobile computing in most cases. For example, a mobile device may be allowed to access the Internet only through the nearby proxy in the same administration domain while the *configured proxy* may refuse to serve the requests from the mobile device in a foreign network. For another example, a firewall may only allow the communication initiated by the nearby proxy behind it. As far as the cases are concerned, the proxy-based software on the mobile device should communicate through the nearby proxy.

Manually changing the proxy configuration of the software at a handoff may work but troubles people. Preparing a proxy list (a Proxy Auto-Config file) merely satisfies the WWW software such as a browser, but hardly works for other software and for the unexplored network whose proxy address cannot be predicted in advance. The IETF Web Proxy Auto Discovery (WPAD) draft [2] describes a number of mechanisms for discovery of nearby HTTP proxies based on DHCP, SLP [3], DNS queries [4, 5], *etc.* The draft surveys a new DHCP option with code 252, the SLP support, a "well known" alias of "wpad" in the DNS database [4], and a specific DNS SRV record [5]. Similarly, the IETF draft "Finding Stuff" [6] uses a DNS "well known" alias and a DNS SRV record to locate services in a domain with only prior knowledge of the domain name. The mechanisms in the drafts seem promising to be extended to find other nearby proxy services [7], but software has to be aware of mobility to use the mechanisms.

Making the software mobility-aware works but has several primary drawbacks. First, user-level software has difficulty in detecting the happening of a handoff because current operating systems seldom provide mobility information and services such as the notifications of a handoff and the new address acquirement. Second, current operating systems do not export to user-level software all data in DHCP options during the DHCP handshake, but certain DHCP option data is critical to mobility. Third, many ways exist for getting a nearby proxy address but challenge the compatibility of the proxy-based software that merely support some of them [2-5]. Fourth, creating the mobility-aware software burdens developers with mobility issues rather than with application designs. Fifth, modifying the proxy-based software has to work with source codes. For example, software may use Session Initiation Protocol [8] and the DHCP option for SIP servers [9] to handle end-to-end communication through a nearby SIP proxy. However, the software has to detect the happening of a handoff and then initiates a SIP re-INVITE procedure. Besides, current operating systems do not export to applications the SIP server addresses in the DHCP option. As another example, software may get the nearby proxy address

through DNS queries [4, 5] but it neither caches the DNS reply against using obsolete addresses nor has an idea about when to send the query again. It is because mobility information such as notifications about the happening of a handoff and the acquirement of a new address is currently held in the operating system and unavailable to user-level software.

Therefore, people need a way to make the proxy-based software communicate through a nearby proxy without software configuration and modification in mobile computing. However, currently no proposal appears to address the issue from the viewpoint. An application-aware router in a content distribution network [10, 11] may be applied to the issue by monitoring application requests and redirecting them to a nearby proxy by means of URI modification. However, the way has several drawbacks: (1) rewriting URI [11] violates certain security mechanisms and hardly works for encrypted application data; (2) scanning packet payloads overloads the network device; (3) it merely supports HTTP-based communication [10]; (4) when the mobile device and the nearby proxy reside at the same network, it makes a sub-optimal communication path because application requests have to traverse the router before reaching the nearby proxy; and (5) it needs to configure and manage the network device to work with the nearby proxy, but the cooperation may be impossible when they belong to different management domains.

In the paper, we propose the Tapster as the transparent proxy redirector for mobile computing. The Tapster automatically redirects applications' requests to the nearby proxy without software configuration and modification. Accordingly, the present proxy-based software can continue working without drawbacks resulting from mobile computing. Leaving out the burden to consider mobility issues, developers can focus on application issues when designing their proxy-based software. Current proposals for getting a nearby proxy address [2-5] can be integrated into the Tapster to improve the compatibility. Moreover, the Tapster supports various network applications not merely WWW applications. The Tapster works at the mobile device to maintain security mechanisms, make the optimal communication path, and mitigate the load of network devices. Without bothering network administrators, people configure and manage the Tapster at their mobile devices according to application requirements. We implement the Tapster in Windows XP and have experiments.

## 2. TAPSTER

### 2.1 Overview

The Tapster, as the thick bar shown in Fig. 1, monitors packets flowing through the mobile device and provides a redirection service for the proxy-based software on demand. When the proxy-based software sends a request to the *configured proxy*, the Tapster intercepts the request and forwards it to the nearby proxy. If the proxy-based software uses a connection-oriented protocol such as TCP, the Tapster does the handshake with the local protocol software on behalf of the *configured proxy*. After that, the Tapster forwards the request to the nearby proxy through an established connection. For being transparent to applications, the efforts make the proxy-based software believe that it communicates through the *configured proxy* when the mobile device moves. The Tapster

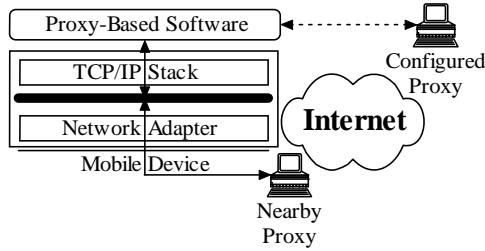


Fig. 1. Tapster overview.

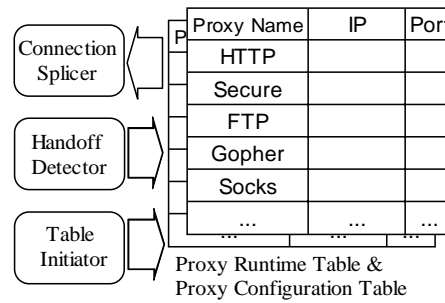


Fig. 2. Tapster components.

works in the mobile device rather than in a network device, *e.g.* an access point or a router, to avoid becoming a bottleneck.

## 2.2 Tapster Components

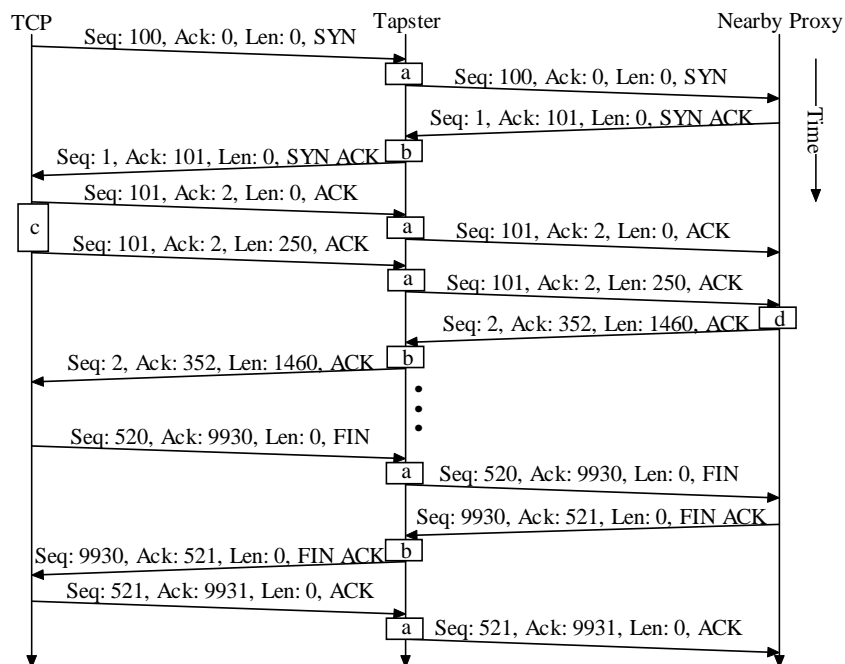
As shown in Fig. 2, the Tapster comprises a Connection Splicer, a Handoff Detector, a Table Initiator, a Proxy Runtime Table, and a Proxy Configuration Table. The Proxy Configuration Table keeps the proxy-based software's proxy configurations. The Proxy Runtime Table caches the nearby proxy's information. The two tables have similar formats. The Table Initiator loads the proxy-based software's proxy configurations from a file or manual input to the Proxy Configuration Table at start-up. The overhead of loading the proxy configurations can be ignored because it happens only once at the Tapster initiation and the proxy configurations almost have addresses of the often-used proxies, *e.g.* the HTTP proxy. After the Tapster initiation is done, the proxy-based software in the mobile device can enjoy the benefits brought by the Tapster without any further configuration while roaming.

The Handoff Detector detects the happening of a handoff and updates the Proxy Runtime Table with the nearby proxy's information. The happening of a handoff can be detected by listening to a router's advertisements [12] or a network adapter driver's notifications. Thereupon the Handoff Detector follows various ways [2-5] to get the nearby proxy's information. The Handoff Detector can passively monitor certain packets flowing through the operating system or actively negotiate with remote servers for the nearby proxy's information. For example, it may scan the packet carrying the Proxy Server Configuration in the DHCP option [7] when the operating system initiates the DHCP handshake. Otherwise, it may query DNS [4, 5]. Finally, it updates the Proxy Runtime Table with the nearby proxy's information.

The Connection Splicer monitors packets, looks them up in the table, translates their addresses and ports, does the handshake with the local protocol software for connection-oriented protocols, and establishes a connection with the nearby proxy on demand. When intercepting an outbound packet whose destination matches an entry in the Proxy Configuration Table, the Connection Splicer consults the Proxy Runtime Table to translate the packet's destination address and port before propagating it to the network. Conversely, the Connection Splicer restores the packet returned from the nearby proxy by translating its source address and port before forwarding it to the local protocol software.

The way is similar to the function of the well-known Network Address Translator (NAT) device [13] in the Internet.

Besides doing address and port translations, the Connection Splicer splices the connections. Splicing connections (in other words, splitting a connection into two segments) is a well-known technique widely used to solve issues in mobile computing without software modification, *e.g.* overcoming the problems of high bit error rate and frequent disconnection to improve TCP throughputs by I-TCP [14] and M-TCP [15], and establishing a mobile connection with an ordinary server through MSOCKS [16]. Despite their different purposes of splicing connections from ours, we use it to hold application transparency, maintain the continuity of connection-oriented protocols, and redirect requests to a nearby proxy on demand. The Connection Splicer responds to connection requests [16] from the local protocol software on behalf of *configured proxies* as the proactive procedure for network applications over connection-oriented protocols. Accordingly, the proxy-based software believes that it communicates through the *configured proxy* when the mobile device moves. Meanwhile, the Connection Splicer establishes a connection with the nearby proxy before delivering the application's request. Fig. 3 briefly shows how the Connection Splicer in the Tapster works for a network application over TCP.



- a: The packet's destination IP and port are translated from the configured proxy into the nearby proxy.
- b: The packet's source IP and port are translated from the nearby proxy into the configured proxy.
- c: The application sends the request to the configured proxy.
- d: The nearby proxy replies to the request with the data.

Fig. 3. Working for a network application over TCP.

Deploying the Tapster in mobile devices not only alleviates the load of network devices but also permits people to manage and configure the Tapster according to application requirements (before the Tapster start-up). Working below the local protocol software can free the proxy-based software from configuration and modification when the mobile device roams through the Internet. Accordingly, the present proxy-based software can work as usual. The Tapster can detect the happening of a handoff and apply various methods to get a nearby proxy address, so developers can focus on designing their proxy-based software without considering mobility and compatibility issues.

Directly forwarding application requests to the nearby proxy can avoid not only the problems resulting from the use of a far proxy but the happening of sub-optimal communication when the mobile device and the nearby proxy reside at the same network. Because the Tapster leaves application data unchanged and merely modifies a packet's network layer and transport layer headers, it not only easily maintains certain security mechanisms whose integrality checks cover only application data but also supports various applications without understanding applications' protocols.

For providing stronger security assurances, the Tapster may cooperate with network layer and transport layer security mechanisms such as IPSec [17]. To this end, the Tapster should be interposed between generic TCP/IP protocol stacks and the security mechanisms in the mobile device. It applies the security mechanisms to outgoing packets after the Tapster's work, while applying the security mechanisms to incoming packets before delivering packets to the Tapster. The way can guarantee user communication against certain security risks, *e.g.* being eavesdropped, because user communication is encapsulated by the security mechanisms before going on the Internet. However, it cannot avoid other deny-of-service (DoS) attacks due to using a phony nearby proxy advertised locally by an attacker. We argue that the similar risk exists in using other popular Internet services such as DHCP [18] or SLP [3] and the way to authenticate a nearby proxy is beyond the scope of the paper since it is proxy-dependent.

### 3. EXPERIMENTS

We implement the Tapster in Windows XP because many computers take Windows as operating systems. A hardware-independent filter driver in Windows XP redirects packets from the kernel to the Tapster in a user-level process and then injects them back to the kernel. Although handling packets in a user-level process may incur some performance penalty due to the context switch, the way makes the Tapster portable since many operating systems support the packet redirection service, *e.g.* the Divert Socket in Linux and FreeBSD. When Windows XP acquires an IP address from the DHCP server, the Tapster uses the Proxy Server Configuration in the DHCP option [7] to update the Proxy Runtime Table accordingly.

We construct networks as shown in Fig. 4 for evaluating the Tapster's performance and verifying its workability. We take WWW as the demonstration because the well-known application usually communicates through a proxy and dominates the Internet traffic. The mobile device runs Windows XP but other computers run Windows 2000 server. A Squid proxy is deployed in Router 1 and Router 2 while Microsoft IIS and GNU Wget are taken as the WWW server and the browser. A 10 Mbps Hub emulates the

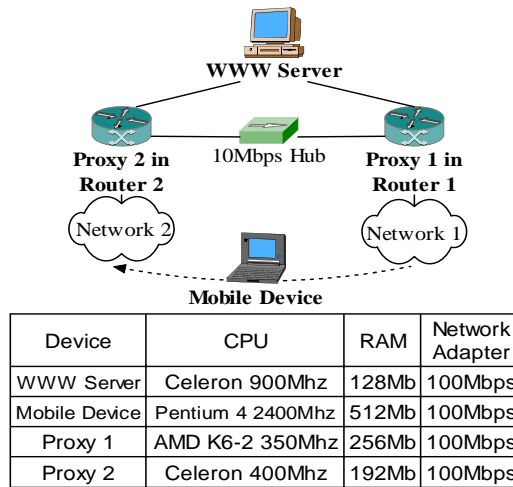


Fig. 4. Experimental networks.

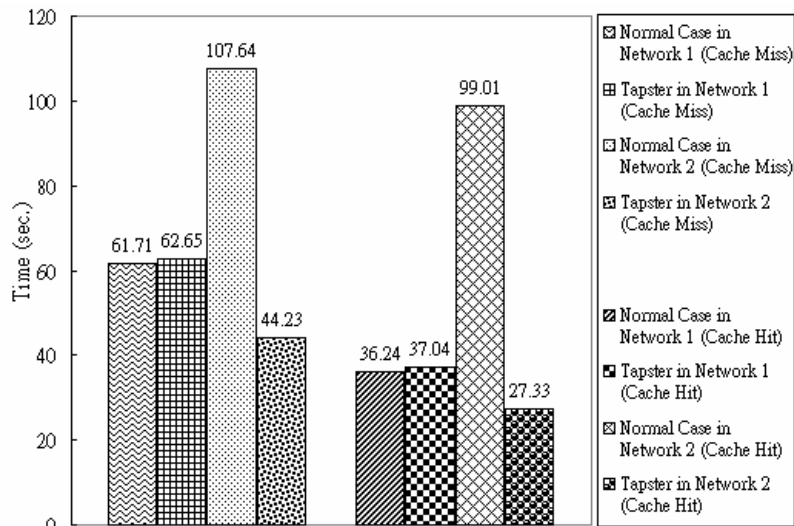


Fig. 5. Performance comparisons between tapster and normal case.

long distance between Proxy 1 and Proxy 2. For avoiding interference such as noise in wireless LAN, the mobile device is connected to different networks via 100 Mbps Ethernet to emulate the movements.

We prepare a 100 MB file in the WWW server and configure the browser in the mobile device to retrieve the file through Proxy 1. The time that the browser takes to receive the file is recorded. The test is executed when the mobile device is in Network 1 and Network 2 under a proxy’s cache hit and cache miss. Besides, we record the situations without the Tapster for comparisons and refer to them as the normal cases. The results are shown in Fig. 5.

In Network 1, the Tapster averagely incurs the negligible overhead of 0.87 second per 100 megabytes in comparison with the normal case because the Tapster needs to check the headers of each packet flowing through the mobile device. When the mobile device roams to Network 2, the browser in the mobile device still retrieve the file through Proxy 1 in the normal case. However, the Tapster redirects the requests to Proxy 2 and improves more than 63.41 seconds per 100 megabytes. What dominate the difference in time are not only the communication delay between Network 1 and Network 2 but also the hardware differences between Proxy 1 and Proxy 2. The former is proportional to the distance from the mobile device to the *configured proxy* while the latter is out of the user's control. It shows that the Tapster can shorten communication delay for the proxy-based software in mobile computing by automatically redirecting requests to the nearby proxy without software configuration and modification.

Besides, the cache miss of the Tapster in Network 2 costs 44.23 seconds per 100 megabytes but the cache hit of the normal case in Network 2 costs 99.01 seconds per 100 megabytes. The cache miss of the Tapster outperforms the cache hit of the normal case for 54.78 seconds per 100 megabytes when the mobile device travels in Network 2 because the mobile device has to fetch cached data from the *configured proxy* far from the visited network in the normal case. The result eliminates the performance penalty concern due to cache miss in a fresh nearby HTTP proxy and justifies the design of the Tapster – redirecting requests to a nearby proxy instead of the remote one in mobile computing.

#### 4. CONCLUSIONS

In the paper, we propose the Tapster as the transparent proxy redirector for mobile computing. The Tapster automatically redirects applications' requests to the nearby proxy without software configuration and modification. Accordingly, the present proxy-based software can continue working without drawbacks resulting from mobile computing. The Tapster can detect the happening of a handoff and apply various methods to get a nearby proxy address, so developers can focus on designing their proxy-based software without considering mobility and compatibility issues. Moreover, the Tapster supports various network applications not merely WWW applications. Since the Tapster works at the mobile device, it can maintain security mechanisms, make the optimal communication path, and mitigate the load of network devices. Without bothering network administrators, people can configure and manage the Tapster at their mobile devices according to application requirements. We implement the Tapster in Windows XP and have experiments. The experiments show that (1) the Tapster averagely incurs the negligible overhead of 0.87 second per 100 megabytes because it needs to check the headers of each packet flowing through the mobile device, (2) the Tapster can redirect requests to the nearby proxy to shorten communication delay caused by the proxy-based software, and (3) the Tapster can work without software configuration and modification.

#### REFERENCES

1. S. Podlipnig and L. Boszormenyi, "A survey of web cache replacement strategies,"

- ACM Computing Surveys*, Vol. 35, 2003, pp. 374-398.
2. I. Cooper, P. Gauthier, J. Cohen, M. Dunsmuir, and C. Perkins, "Web proxy auto-discovery protocol," draft-cooper-webi-wpad-00, 2000.
  3. E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol, version 2," RFC 2608, 1999.
  4. M. Hamilton and R. Wright, "Use of DNS aliases for network services," RFC 2219, 1997.
  5. A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," RFC 2782, 2000.
  6. R. Moats and M. Hamilton, "Finding stuff," draft-ietf-svrlloc-discovery-10, 1998.
  7. S. K. Balasubramanian, M. Alexander, and G. Neumann, "DHCP option for proxy server configuration," draft-ietf-dhc-proxyserver-opt-04, 2004.
  8. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session initiation protocol," RFC 3261, 2002.
  9. H. Schulzrinne, "Dynamic host configuration protocol (DHCP-for-IPv4) option for session initiation protocol (SIP) servers," RFC 3361, 2002.
  10. V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The state of the art in locally distributed web-server systems," *ACM Computing Surveys (CSUR)*, Vol. 34, 2002, pp. 263-311.
  11. M. Day, B. Cain, G. Tomlinson, and P. Rzewski, "A model for content internet-working (CDI)," RFC 3466, 2003.
  12. S. Deering, "ICMP router discovery messages," RFC 1256, 1991.
  13. P. Srisuresh and K. Egevang, "Traditional IP network address translator (Traditional NAT)," RFC3022, 2001.
  14. A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," in *Proceedings of the 15th International Conference on Distributed Computing Systems*, 1994, pp. 136.
  15. K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," *ACM Computer Communication Review*, Vol. 27, 1997, pp. 19-43.
  16. D. A. Maltz and P. Bhagwat, "MSOCKS: an architecture for transport layer mobility," in *Proceedings of the 17th IEEE INFOCOM*, Vol. 3, 1998, pp. 1037-1045.
  17. S. Kent and R. Atkinson, "Security architecture for the internet protocol," RFC 2401, 1998.
  18. R. Droms and W. Arbaugh, "Authentication for DHCP messages," RFC 3118, 2001.

**Tzu-Chi Huang (黃祖基)** received his B.S. and M.S. degrees in Electrical Engineering from National Cheng Kung University in 1997 and 1999. He was a system program engineer responsible for the developments of network device drivers and related protocols in Silicon Integrated Systems (SiS) Corp. Now, he pursues the Ph.D. degree in the Department of Electrical Engineering at National Cheng Kung University. His research interests include mobile computing, network protocols, and operating systems. He has much experience in protocol engineering.

**Ce-Kuen Shieh (謝錫堃)** received his B.S., M.S., and Ph.D. degrees in Electrical Engineering from National Cheng Kung University, in 1977, 1983, and 1988 respectively. Currently, he is a professor in the Department of Electrical Engineering at National Cheng Kung University. His research interests include distributed and parallel processing, operating systems, computer networking, and compiler.

**Bo-Yang Lai (賴柏仰)** received his B.S. degree from National Cheng Kung University in 2005. He is a M.S. student in the Department of Electrical Engineering at National Cheng Kung University. His research interests include mobile computing and network protocols.

**Yu-Ben Miao (苗育本)** holds a Ph.D. degree from the National Cheng Kung University. He is currently a project leader of Internet Telecommunication Department in Information & Communication Laboratory, Industrial Technology Research Institute. His current focus is on IP telephony, wireless communication, quality of service, and mobile networks technical areas.