THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-97-08

A Mobile TCP Socket

Xun Qu, Jeffrey Xu Yu and Richard P. Brent

April 1997

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical.Reports@cs.anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

`http://cs.anu.edu.au/techreports/`

Recent reports in this series:

TR-CS-97-07 Richard P. Brent. *A fast vectorised implementation of Wallace's normal random number generator*. April 1997.

TR-CS-97-06 M. Manzur Murshed and Richard P. Brent. *RMSIM: a serial simulator for reconfigurable mesh parallel computers*. April 1997.

TR-CS-97-05 Beat Fischer. *Collocation and filtering — a data smoothing method in surveying engineering and geodesy*. March 1997.

TR-CS-97-04 Stephen Fenwick and Chris Johnson. *HeROD flavoured oct-trees: Scientific computation with a multicomputer persistent object store*. February 1997.

TR-CS-97-03 Brendan D. McKay. *Knight's tours of an 8 × 8 chessboard*. February 1997.

TR-CS-97-02 Xun Qu and Jeffrey X. Yu. *Mobile file filtering*. February 1997.

A Mobile TCP Socket

Xun Qu
Computer Sciences Laboratory, RSISE
The Australian National University
Canberra, ACT 0200 Australia

Jeffrey Xu Yu
Department of Computer Science
The Australian National University
Canberra, ACT 0200 Australia

Richard P. Brent
Computer Sciences Laboratory, RSISE
The Australian National University
Canberra, ACT 0200 Australia

Abstract

In this paper, we propose a mobile TCP socket which is the second step of our two step approach to support mobility. The first step of our approach is to support portability as reported earlier. In our mobile TCP socket, a mobile mapping is introduced, which maps TCP associations to underlying TCP connections. The mobile mapping can be implemented in the socket layer and on top of TCP/IP protocol layers. Our approach achieves high compatibility with current TCP/IP protocols, due to the fact that neither of the two steps in our approach impose any changes on TCP/IP protocols nor the underlying Link layer. At the same time, our approach can reduce both the propagation cost for distributing location information of mobile hosts and the forwarding cost for forwarding IP datagrams across the Internet. The concept of mobile mapping and its implementation details are given in this paper.

1 Introduction

Truly ubiquitous mobile computing requires that programs on mobile computers be able to process continuously, as in a distributed environment. However, the current TCP/IP protocols are designed for hosts which have unique Internet-wide IP addresses and are not supposed to move from one internetwork to another while network applications are running. In TCP/IP-based systems, the hardness of mobile communication stems from IP routing and addressing schemes. The IP address has two related usages. First, it is used as a unique identifier for a host. Second, it is involved in routing algorithms. If a host moves into a new network, its old IP address can not be used for normal TCP/IP communications, because all packets intentionally addressed to this host will be routed to its old network[1]. Most mobile IP solutions[4, 5, 22, 11, 12, 14, 3] support mobility in the network layer from which the problem stems. Therefore, the current IPv4[17] needs to be enhanced by some special routing mechanisms which have to be supported by mobile hosts and/or routing facilities. This will be the major barrier for quick, wide and easy deployment of the existing mobile IP solutions.

We differentiate between mobile and portable protocols. A portable protocol can provide communication services when the host stays in different networks. However, the portable communication services are intermittent in terms of two meanings: no communication services can be provided during the period of moving, and important system parameters may be changed, that will cause loss of communication connections with their peer systems. A mobile protocol is a protocol that can keep all current open connections alive and provide continuous communication services at all times.

Our previous work on portable IP communication employs the existing protocols and extends their services to support portable communication[15]. Basically, our portable solution relies on existing pro-

[1] The network id in the old IP address instructs the routers to forward packets in this way.

tocols, such as DHCP and DNS, and does not need any modifications to TCP/IP protocols and routing facilities. In this paper, we propose a mobile solution for TCP/IP-based host systems on top of our portable support system. The core of mobile solution is mobile TCP socket which can be implemented in the socket layer and on top of the TCP/IP layers. This approach can achieve high compatibility with current TCP/IP systems for the following three reasons. First, we adopt the same API of standard socket. Second, we don't require any special functionality in the network or the transport layers to support mobility. Finally, we don't require all networks, where mobile hosts may visit, to support special routing mechanisms. In terms of performance, our mobile solution can reduce both the propagation cost for distributing the location information of mobile hosts and the forwarding cost for forwarding IP datagrams across the Internet.

The remainder of this paper is organised as follows. In Section 2, related work will be presented. In Section 3, we discuss the main issues we are concerned with. Section 4 briefly outlines our previous work on portable support. In Section 5, we introduce our two step approach for supporting mobility. Preliminaries are given in Section 6. In Section 7, the concept and the issues for mobile mapping are discussed. In Section 8, we discuss a mechanism, called virtual port, to support mobile mappings. A detailed example using our protocol is given in Section 9. We conclude our work in Section 10.

2 Related Work

At first, we briefly describe the terms we will use in this paper regarding mobile communications. A mobile host, is a TCP/IP-based host which can change its attach point to different internetworks and access network services as a fixed host. Each mobile host has a home network, in which the mobile host is assigned a permanent IP address which is used by regular TCP/IP protocols to communicate with each others. In contrast, a network in which mobile hosts temporarily stay is called a foreign network, or a guest network.

In TCP/IP protocol suite, as shown in Figure 1, there are four protocol layers. The ultimate goal of mobile solutions is to provide network applications with mobile communication services in the layers under the Application layer. Existing solutions can be distinguished based on the primary layer on which they provide mobility.

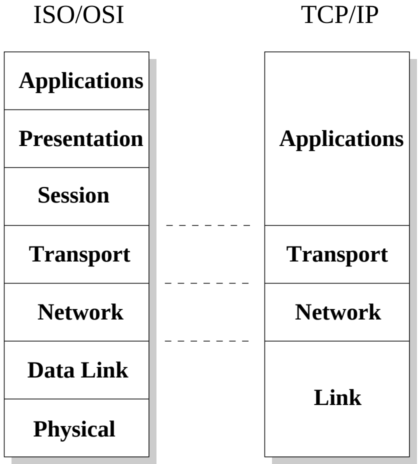

Figure 1: TCP/IP protocol Stack with OSI/RF

Network Layer Solutions

Recent research work on mobile TCP/IP has been conducted at the network layer from where the key problem stems. These solutions aim at supporting mobility in the IP protocol layer. There are four major Mobile Host Protocol (MHP) proposals, namely, Sony MHP[22], Columbia MHP[4, 5], IBM

MHP[11, 12][2] and IMHP[3, 13, 14]. The IMHP proposal was derived from the others, and has now been adopted by IETF as the Internet draft standard. All of these four MHPs share a common technology that is to provide mobility at the network layer and keep the IP addresses of mobile hosts constant to all the upper layers. Hence, no transport or application protocol entities will be aware of any mobile operations. All mobile functions are completed at the network layer and mobility features are hidden from the upper layers.

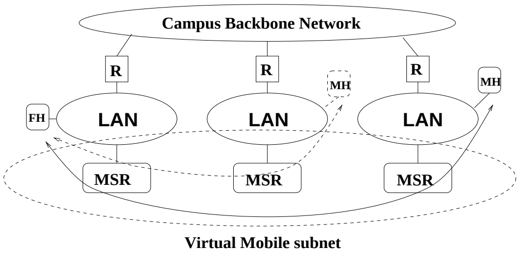

Figure 2: Columbia MHP

The Columbia MHP proposes a virtual mobile subnet which is composed of a set of mobile support routers, denoted as MSR, scattered on a campus network. The campus network can consist of several internetworks including some remote internetworks over the global Internet (Figure 2). All packets destined to mobile hosts will first be routed to a nearest MSR which tunnels the IP packets to the current MSR with which the destination mobile host is connected at the moment. When a mobile host migrates into a different location, the mobile host will first register itself with the local MSR which in turn informs all of other MSRs of the location of this mobile host. Mobile hosts always use their permanent IP addresses in such a virtual mobile subnet.

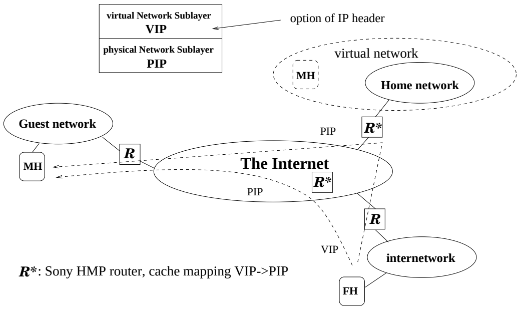

Figure 3: SONY MHP

The Sony MHP envisages a virtual network on top of normal IP internetworks that are called physical networks. Mobile hosts are treated as fixed hosts in a virtual network in the sense that their virtual network addresses are constant. In each IP datagram, a new IP option is used to accommodate the virtual network address. When mobile hosts roam into different internetworks, their physical network addresses are changed. The location of each mobile host is kept in its home mobile router and is propagated to all Sony routers, which will cache location information for correctly routing consequent IP datagrams. Therefore, any of these Sony routers can map the virtual network network address to the physical network address and forward IP datagrams to mobile hosts.

The IBM MHP[20] employs a standard IP option, loose source and record routing(LSRR), to forward IP datagrams to mobile hosts. When a mobile host moves into a new location, it first registers itself with a local base station and informs its home mobile router of its new location as well. The mobile host uses its original permanent IP address in the new location. If a fixed host sends an IP datagram to this mobile host, it sends this IP datagram to the home mobile router of this mobile host. The home mobile router will forward the IP datagram to the current base station where the mobile host stays by

[2] David Johnson proposed similar MHP using IP LSRR[6, 7]

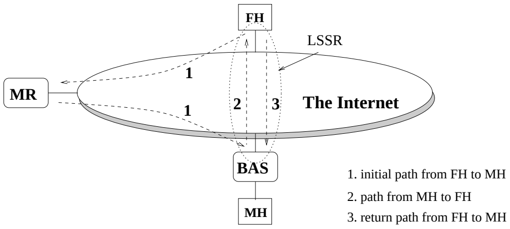

Figure 4: IBM MHP

assigning this base station as the next router in the LSRR option (route 1, in Figure 4). Then the base station delivers the IP datagram to the mobile host locally. The mobile host can send IP datagrams back to the the fixed host directly. If the router at fixed host's internetwork is LSRR enable, the data can then be directly sent from the fixed host to the mobile host by assigning the base station as the next router in the LSRR option.

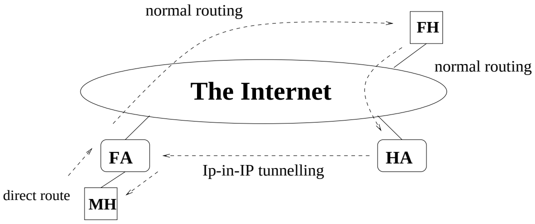

Figure 5: IMHP

IMHP uses home agents and foreign agents to route datagrams to/from mobile hosts (Figure 5). Each mobile host has two IP addresses: a temporary current IP address and a permanent home IP address. The home IP address is the unique identifier of a mobile host. The current IP address is a care-of IP address which indicates the current location of a mobile host. In a foreign internetwork, a mobile host will register itself with a local foreign agent and in return obtain a care-of IP address. Then the mobile host will inform its home agent of the care-of address. Afterwards, the incoming IP datagrams will first reach its home agent, then be forwarded to mobile host's current care-of address using IP-IP tunnelling. The care-of IP address can be the IP address of the foreign agent, or a temporary IP address assigned to the mobile host. In the former case, the foreign agent is responsible to strip the IP-IP tunnelling packets and deliver the enclosed datagrams locally to the mobile host. In the latter case, the mobile host has to do the tunnelling communication with its home agent.

Solution	Forwarding Point(s)	Method	Location Propagation	Protocol
Columbia	multiple MSRs	IP-IP tunnelling	among MSRs	MICP
Sony	multiple SRs	normal IP	among SRs	new IP option
IBM	home MR	normal IP	between MR and BAS	LSRR
IMHP	HA	IP-IP tunnelling	between HA and FA	new protocol

Table 1: Summary of Mobile IP Solutions: Note that MSR(Mobile Support Router; SR(Sony Router); MR(Mobile Router); BAS(Base Station); HA(Home Agent); FA(Foreign Agent); and LSRR(Loose Source Routing).

As a brief summary, the network layer solutions are aimed to provide a seamless mobile communication services to transport and application layer entities, hence the basic requirement is to keep the IP address constant to the upper layers. Since the existing routing infrastructure cannot properly forward IP datagrams to mobile hosts that are away from their home internetworks, MHPs focus on two basic tasks: propagation of location information and forwarding IP datagrams through current IP routing systems in the global Internet. When a mobile host moves from one internetwork to another, the MHPs are responsible to propagate the new location information to all forwarding points. Therefore, any forwarding point can forward IP datagrams to mobile hosts. As in Table 1, Columbia and Sony MHPs use multiple forwarding points distributed across Internet. The location information is exchanged within these special routers. Columbia mobile support router uses a special MICP[4] to exchange mobile hosts location information, whereas Sony router collects and caches mapping between the virtual network address and the physical network address from new IP option. IBM MHP and IMHP use a single forwarding point. The location information is exchanged between a mobile host and a single forward point. All these solution use either IP-IP encapsulation (tunnelling) or source routing to forward datagrams.

Solutions in Other Layers

The Mobile Data Link(MDL) solution[10] suggests that mobility be transparently supported at the lowest Link layer above which all running protocol entities are not conscious of any mobile support. If a mobile host is in its home network, all data link packets will be sent and received in a normal way. If the mobile host moves into a guest network, as shown in Figure 6, the home access point, AP_2, and the guest access point, AP_1, will cooperate with each other to forward data link packets to/from the mobile host by encapsulating data link packets into IP datagrams. These encapsulating IP datagrams are embedded into data link packets again to be sent to local IP routers via the local area network. In this case, the mobile host will be treated as to reside in the home network. This architecture can span over multiple physical networks.

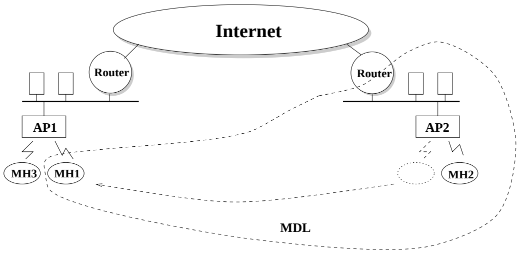

Figure 6: MDL Communication

The main advantage of MDL is that any network and transport protocols can run over MDL and obtain transparent mobile communication services. However, in the MDL systems, there must be an access point in all participating networks to support link layer packets forwarding. All access points have to know each others for correct forwarding. Since every forwarded packet will be encapsulated into an IP packet, in turn it has two network and link layer headers and trailers included. The double encapsulation reduces the percentage of payload in such link layer packets and reduces the efficiency of data transfer. Moreover, all packets to a mobile host, which is currently not in its home network, will be routed to its access point at its home network at first and then be forwarded to the access point in the guest network. Access points are a potential bottleneck in addition to detour routing.

The mobile protocol can also be implemented in the transport layer. I-TCP [1] suggests that a single TCP connection to a mobile host be separated into two connections. One is the special mobile TCP connection between the mobile host and the mobile support router. The other is a normal TCP connection between the mobile support router and the corresponding host, as in Figure 7. In the mobile support router, there is a mapped socket, $(IP_m, Port)$, which represents the socket on a mobile host. From the viewpoint of a fixed host, this socket looks like a normal socket since it has a constant identifier $(IP_m, Port)$. When the mobile host moves into another internetwork, the mobile host will first set up a mobile TCP connection with a new mobile support router and the new mobile support router will take over the normal TCP connection from the previous mobile support router by creating the same mapped socket. However, because a mapped socket has to have a unique IP address and port number, all mobile support routers for a single TCP connection must act as if it were the owner of this socket. Therefore, with this solution, all mobile hosts are limited into one internetwork.

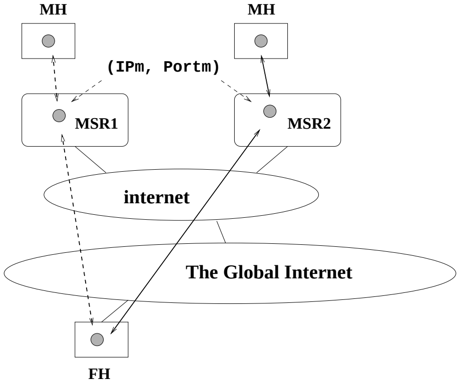

Figure 7: Indirect TCP Solution

3 Issues

As claimed in [4], the network layer seems to be a reasonable place to provide mobility. We focus on the network layer solutions. Although these network layer solutions mentioned above have achieved a degree of success, all of them face one or both of the problems of poor performance and backward compatibility[9].

As for backward compatibility, none of the existing mobile solutions are fully compatible with existing Internet protocols and infrastructure at the network layer where they are intended to solve the mobility problem. The Columbia and IBM MHPs limit mobile hosts to visit such internetworks where mobile support stations exist. Sony MHP needs more special routers in Internet for forwarding IP datagrams. Although IMHP claims that no special requirements will be imposed on the internetworks where mobile hosts will visit, it assumes that a mobile host can send IP datagrams using its home IP address directly in a foreign network. Unfortunately, most of the current routing devices do not route local IP datagrams with a foreign IP address as source address. Some networks even perform source address filtering for security reasons. Therefore, a foreign address can not get through the firewall[2]. Hence, IMHP still needs to impose some special support on the visiting internetwork.

The performance issues rely on the cost to propagate location information, C_p, and the cost to forward datagrams, C_f. Obviously, more forwarding points and wider propagation of location information will achieve suboptimal routing, i.e. lower C_f. But this will result in less backward compatibility and high C_p. On the other hand, if we employ fewer forward points, the C_p will be very low but at the same time C_f will be higher. In such a case, all traffic from fixed hosts to mobile hosts has to be forwarded by the home networks of the mobile hosts. This indirect and triangle routing has significant impacts on performance.

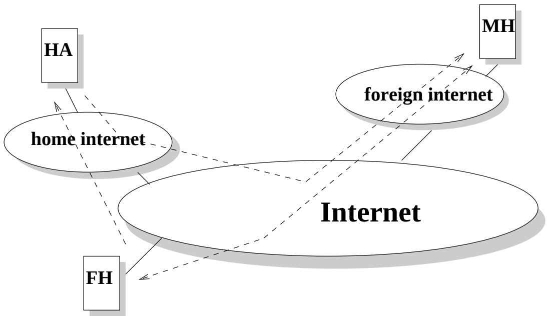

Figure 8: Mobile IP Triangle Routing

In the following we will outline the performance decreasing due to the indirect and triangle routing, and will show the major benefits if such indirect and triangle routing can be avoided. At first, we identify a metrics by which we can measure the performance of routing protocols. The metrics used in RIP[20] is the number of hops. In the newer OSPF, the metric is a dimensionless cost which can be measured in throughput, round-trip time, reliability, etc. The cost model we use here is based on the above two metrics. Let V be the volume of data being transfered, and let c be cost associated with sending a unit of data[3]. Then the total cost will be as follows.

$$C_f = \sum_{i}^{d} c_i \cdot V$$

where d is the number of hops for transferring data between two ends.

As shown in Figure 8, if a mobile host is away from its home network and is communicating to a fixed host. Although the mobile host can send IP datagrams directly to the fixed host, all packets from the fixed host to the mobile host will be forwarded by the home agent of the mobile host. Let $C_{f_{imhp}}$ be the total cost of a communication session, for example, telnet, ftp or Web accesses. Let $\overrightarrow{V}$ and $\overleftarrow{V}$ be the volume of data being transferred from/to the fixed host and to/from the mobile host, respectively.

$$C_{f_{imhp}} = (\sum_{i}^{d_1} c_i + \sum_{j}^{d_2} c_j) \overrightarrow{V} + \sum_{k}^{d_3} c_k \cdot \overleftarrow{V}$$

where d_1 is the number of hops between the fixed host and the home agent of the mobile host, d_2 is the number of hops between the home agent and the mobile host, and d_3 is the number of hops between the mobile host and the fixed host. Here, we ignore the traffic overhead of IP-IP encapsulation. Without the indirect and triangle routing, the total cost of communication will be the following:

$$C_{f_{dir}} = \sum_{l}^{d_3} c_l \cdot \overrightarrow{V} + \sum_{k}^{d_3} c_k \cdot \overleftarrow{V}$$

Obviously, there is no indirect and triangle effectiveness when sending data from the mobile host to the fixed host. Therefore, there is no difference in terms of communication cost for such data transmission. In addition, the ratio of $\overleftarrow{V} \,/\, \overrightarrow{V}$ is less than 1 and is supposed to be very small. As reported, for telnet access, the ratio of $\overleftarrow{V} \,/\, \overrightarrow{V}$ is around 1/20. The ratios for ftp and Web accesses are much less than that figure. Therefore, we ignore this cost: $\sum_{k}^{d_3} c_k \cdot \overleftarrow{V}$ in our discussion. The ratio between C_{imhp} and C_{dir} is given below.

$$\frac{C_{f_{dir}}}{C_{f_{imhp}}} = \frac{\sum_{l}^{d_3} c_l}{\sum_{i}^{d_1} c_i + \sum_{j}^{d_2} c_j}$$

It is a well-understood fact that the number of hops for triangle routing will be greater than that for direct routing. Therefore, we obtain $d_3 < d_1 + d_2$. If the whole network is assumed to be balanced in terms of cost metrics of links, every c_m will be the same for transferring a unit of data. Obviously, the ratio between $C_{f_{imhp}}$ and $C_{f_{dir}}$ is determined by $d_3/(d_1 + d_2)$ which is less than 1. Two common cases are considered. First, suppose that mobile hosts and fixed hosts are located in the same network. The ratio between $C_{f_{imhp}}$ and $C_{f_{dir}}$ will be 0.5. Second, suppose that a mobile host is in a guest network which is far away from the home network of the mobile host and suppose that the mobile host wants to access data in the guest network. A larger figure can be easily expected.

Next, we assume that the whole network is unbalanced so some c_m can be very high and some c_n can be very low. Note that in a local network the traffic will be balanced and can be managed to be balanced. Therefore, if both mobile host and fixed host are in the same home network, the ratio is still expected to be about 0.5. When a mobile host moves to a foreign network, in theory, the value of C_{dir}/C_{imhp} can be greater than 1. An optimal path is expected to be found on top of current routing infrastructure. In fact, using current routing infrastructure, it would be very difficult to take every c_m associated with every

[3] For simplicity, we don't include the processing cost at routers.

link into consideration when determining an optimal path[4]. In addition, as indicated in [8], datagrams to mobile hosts are often routed along paths that are significantly longer than optimal. The probability of a router being a bottleneck in a longer path is much higher than that of a short path.

Consequently, it is highly desirable if we can avoid indirect and triangle routing in a mobile environment and keep the communication cost as less as possible. Meanwhile, the compatibility should be paid enough attentions to. The major contributions of our solution are a) provide high compatibility with current TCP/IP systems, and b) keep both forwarding cost, C_f, and propagation cost, C_p, as low as possible.

4 Previous Work on Portable System

Our mobile solution proposed in this paper is primarily based on our previous work on portable TCP/IP solution [15] which avoids indirect and triangle routing in a portable environment. The basic idea of our portable solution is to decouple two-tie use of IP addresses in the conventional TCP/IP environment in which IP addresses are used to identify hosts in Internet wide and to route IP datagrams to them. In our portable system, when a portable host arrives at a new location, the portable host will be dynamically assigned a temporary IP address, called current IP address, which will allow the portable host to communicate with other systems in a regular way on the condition that its DNS name and its permanent IP address (used at its home network) are unchanged and used as the unique identifier. There are three component systems in portable TCP/IP system:

- A portable host, has portable support functions and uses an regular TCP/IP protocol stack. The enhanced portability can dynamically reset TCP/IP to new system parameters in a new network environment.

- A Home Portable Support System, H-PSS, is the system residing in the network where portable hosts have their permanent IP addresses. H-PSS receives information from portable host and dynamically updates its database. The H-PSS serves to other host systems as a DNS server as well.

- A Foreign Portable Support System, F-PSS, is the system working in a network where portable host will visit. F-PSS is responsible for allocating a local IP address and assigning system parameters including default router, MTU and other parameters to a visiting portable host. F-PSS will contact H-PSS of the portable host to authenticate it if security is required. As an option, the basic portable functions including the assignment of IP addresses and system parameters to portable host can be taken over by DHCP, which is an Internet standard and has been populated in the Internet.

When a portable host migrates into a network, the portable host follows the following procedure to go back on line and to work as a non-portable system to its peer communication systems.

- portable host broadcasts its requests to find a local F-PSS or DHCP server in order to obtain new system parameters, in particular the current IP address.

- If there is F-PSS, F-PSS will talk with this portable host's H-PSS and authenticates the portable host's requests. If the request is granted, F-PSS will issue a temporary IP address and assign other system parameters to portable host.

- After resetting TCP/IP, portable host sends back the new IP address and other location-related information back to its H-PSS. H-PSS will check whether the F-PSS is bogus. This can avoid fault routing IP datagrams.

As far as we can see above, in a foreign network where portable hosts are visiting, the portable hosts look like a fixed system in the sense that portable hosts initiate communication sessions with other systems in a normal way using a new assigned IP address. When a system needs to talk to the portable host

[4] IETF proposed an additional routing scheme in [8] for extensive operations to its IMHP. However, it imposes additional burden on both mobile hosts and correspondent systems to encapsulate datagrams so as to complicate these systems too.

residing in the foreign network, it uses the portable host's home IP address or DNS name to inquire H-PSS in order to obtain portable host's current IP address. The Figure 9 shows the entire procedure of communications among these systems.

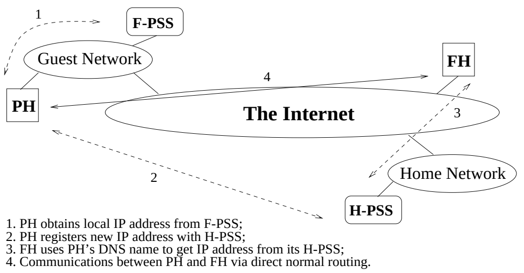
1
F-PSS
Guest Network
FH
4
PH
The Internet
3
Home Network
2
H-PSS
1. PH obtains local IP address from F-PSS;
2. PH registers new IP address with H-PSS;
3. FH uses PH's DNS name to get IP address from its H-PSS;
4. Communications between PH and FH via direct normal routing.

Figure 9: Portable Host System

Our portable IP support system does not require special support from any foreign networks and routers except the standard protocols, such as DHCP, BOOTP etc. The main advantages of portable system are as follows.

- **Direct routing:** It can save communication costs compared to indirect and triangle routing. Since portable system does not require special routing mechanisms, a portable system and its correspondent system communicate through existing routing infrastructure. Unlike IMHP which requires IP-IP encapsulation, our portable system consumes less network bandwidth and CPU time at both home support systems and either special routers at foreign network or at mobile hosts.

- **Few requirements are imposed:** Only standard DHCP or BOOTP services are required at foreign networks. Special protocols are implemented only on the portable system itself and its home portable support system. Then it is easy to deploy portable systems in the Internet and give large roaming areas for mobile hosts in Internet.

- **Ease of implementation:** Because a portable system does not modify network and transport layer protocols, not only can it talk to normal TCP/IP system, but it satisfies the most likely comparability with any internet environment without any portable support at a foreign network. The users of our portable system can access server systems as normal using telnet, ftp, email, etc.

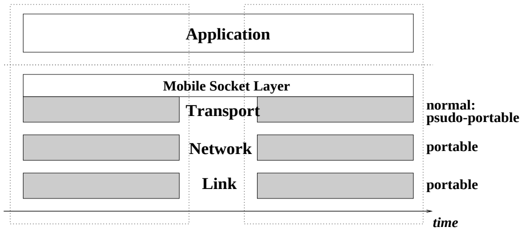

Figure 10: Two-step Solution

5 Mobile TCP Solution

Our portable support system plays an important role in our mobile TCP solution. In our mobile TCP solution, we further exploit the advantages of our portable system, and enhance it with mobile TCP communication services. Figure 10 portraits a layering structure of our mobile system. To provide application layer with mobile TCP services, the mobility in our mobile solution is provided in two separate steps.

- **(Step-1) Portability:** Up to transport layer, the portability is supported by unmodified TCP/IP protocols with additional portable support module. The Link layer in TCP/IP protocol suite is supposed to be portable since various types of Ethernet, Token Ring or serial line protocols (e.g. PPP or SLIP) offer portability in the sense that users can easily plug into and unplug from the network connector at the physical and link layer. Our portable system is adopted as given in the previous section.
- **(Step-2) Continuity:** In our approach, TCP/IP protocols themselves are intact which guarantees that our mobile system can communicate with all TCP/IP systems. The continuity is provided at the socket API layer. The socket API is not a protocol layer in TCP/IP protocol stack, but it is an access interface to transport services in all BSD-clone TCP/IP implementations as well as in Windows environment (Winsock 1 and 2). We call our enhanced socket layer Mobile Socket Layer(MSL).

With our TCP mobile solution, mobility is equal to portability plus continuity. Our mobile solution is twofold. First, the mobile TCP support is at socket layer, all underlying protocols need not to be modified to support mobile TCP communications. Secondly, the deployment of our mobile system is much easier than existing MHP solutions in terms of employing current routing infrastructure. The implementation details will be discussed in the later sections.

Assume that an application process at mobile host MH_a is communicating with a peer application App_a at another host by using the BSD socket in a network Net_a, and further suppose that MH_a needs to move to another network Net_b meanwhile the connection with App_a need remain uninterrupted. Using this situation as an example, the continuity provided by MSL has to support two main functions, namely, the continuous TCP association and I/O support. First, active TCP association between a pair of TCP sockets must not be interrupted by any losses of underlying communication links caused by movement of mobile hosts. When MH_a leaves Net_a, the TCP connection will be broken. As soon as MH_a arrives at Net_b, the portable system will first assign system parameters including a new IP address to MH_a. Then, after TCP/IP protocols resume functioning, MSL will then reopen new TCP connections for pending associations that is for communicating with App_a in an automatic and transparent way. Second, During moving, no network I/O will be performed. In order to provide a seamless socket services, MSL keeps accepting I/O requests from the upper layer even when TCP connections have been broken. The pending I/O requests will be performed after the connection is reopened.

6 Preliminaries

6.1 TCP Association and TCP Connection

The BSD socket API is a set of system calls which provide applications with accesses to communication services. As first introduced in TCP protocol[18], the concept of socket was defined as an endpoint of TCP communication and identified by an IP address and a TCP port number. In BSD Unix socket API, a socket is further extended to a protocol-independent communication endpoint. Given there are several protocol suites (e.g. OSI, XNS, SNA and Internet TCP/IP), each socket will be bound to one of them, which is called *Domain*. Each *domain* can have more than one candidate transport or network layer protocols to be further assigned to a open socket. As shown in Figure 11, in the Internet domain, three types of sockets are available including datagram (UDP), data stream (TCP) and raw socket.

A socket is presented as a triple, *(protocol, IP-address, port-number)*. The binding between two sockets is call an association which is defined as a 5-tuple[19]:

(protocol, local-IP-address, local-port-number, peer-IP-address, peer-port-number)

Importantly, an association is Internet wide unique, that means an association between two sockets can be set up once at a time, no two same associations can exist simultaneously.

In this paper, we restrict ourselves to TCP. A TCP association will be established between an active open socket and a passive one, or called client and server. A TCP association is defined as

(TCP, local-IP-address, local-port-number, peer-IP-address, peer-port-number)

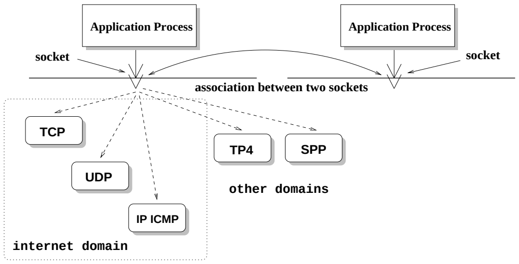

Figure 11: Socket and Protocols

For a TCP association between two TCP sockets, a TCP connection, *(local-IP-address, local-port-number, peer-IP-address, peer-port-number)*, is established in the transport layer. The TCP association is then mapped to the corresponding TCP connection. The transport layer is responsible for real communications.

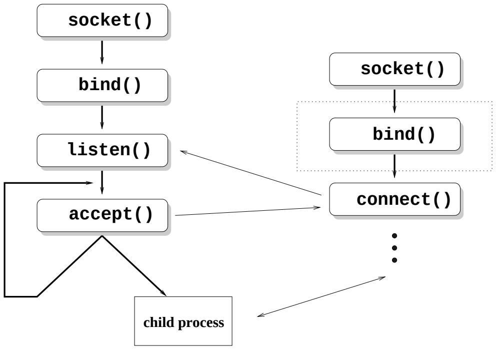

Figure 12: Sequence of System Calls

6.2 Socket System Calls

A typical sequence of socket system calls are shown in Figure 12. The server first creates a socket, and then binds it to a local address, namely a local-IP address and a port number. All client programs must know that port number if they intend to use the services provided by the server. The next step at the server site is to use `listen()` and `accept()` in order to wait for any incoming connection requests. When a client requests a connection, `accept()` will return a new socket for the server to response. The initial socket is only used for acceptance new incoming requests.

6.3 The TCP I/O Semantics

The TCP I/O semantics is mainly affected by its internal buffer and the I/O system calls which manipulate the internal buffers. Using 4.4BSD-Lite as an example[23], there are two groups of socket I/O system calls that manipulate the internal buffer, namely, the *write system calls* and the *read system calls*. They include `write, writev, sendto, sendmsg` and `read, readv, recvfrom, recmsg`, respectively.

Figure 13 shows both the control and data follows for the *write system calls*. Application processes pass to the socket layer data and control messages including the destination IP address by calling one of these four write system calls. Two socket layer calls `soo_write` and `sendit` keep such requests in an internal data structure. The socket layer `sosend` first tries to obtain enough buffer space to hold all data if possible, and then copy them to the sending buffer by calling the TCP layer subroutine — `pr_usrreq`. Figure 14 shows both control and data flows for *read system calls*. The two socket layer calls, `soo_read` and `recvit`, keep user requests in an internal structure with which a socket layer call `soreceive` performs real data transfer from the TCP receiving buffer to the application process buffer.

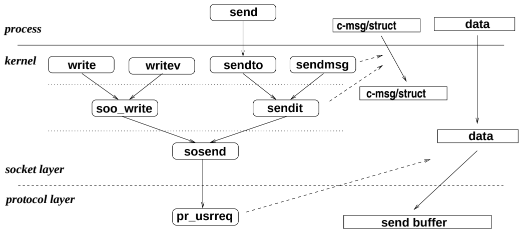

Figure 13: Write System calls and Buffering

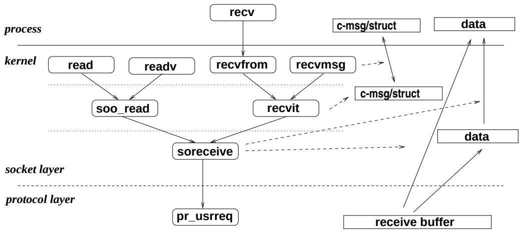

Figure 14: Read System calls and Buffering

As for write calls, locally the data being sent are first copied into the socket layer buffer, and then to the TCP layer's sending buffer. For synchronous I/O socket, write system calls return after data have been copied into the TCP sending buffer. For asynchronous I/O socket, though write calls return immediately, the signal of I/O completion is sent to application process after data are properly put into the TCP sending buffer. Therefore, there are no any chances of loss of data when transferring data to the TCP sending buffer. However, no error occurring in copying data locally does not ensure that data are successfully transfered to the remote site. Closing socket will cause loss of data in the TCP sending buffer. Usually `close` closes TCP socket. At default, `close` call will return immediately and all data in the TCP sending buffer are sent out. When tearing down the TCP connection, application processes can discard pending data to be sent by set the socket option `SO_LINGER`. Another system call `shutdown` performs similar closing function.

As for read calls, TCP protocol will not discard any data correctly received. If there are no buffers in the receiving buffer, TCP will close its receive window. All data out of the receiving window are discarded and will not be acknowledged. If processes provide a small read buffer, the socket layer will only transfer up to the size of that small read buffer. The remainder will be supplied for next read. Similarly with the sending buffer, the receiving buffer will be cleared by `close` and `shutdown`.

A socket can be closed in two ways — *read half* and *write half*. When closing *write half*, application can further indicate whether the pending data are to be discarded or not.

In summary, TCP socket guarantees that data will be correctly deliver to remote process without duplication and loss in normal data transfer period. Socket I/O system calls will not introduce additional possibility of loss of data. When sockets are closing, the data remaining in the sending and receiving buffers may be discarded by TCP entity.

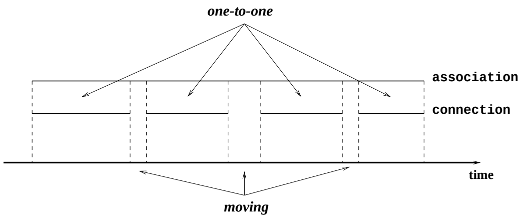

Figure 15: Life time of Association and Connection

7 The Continuity of TCP is Mobile Mapping: The Concept and Issues

In this section, we first introduce the concept of mobile mapping and discuss the issues on how to implement such mobile mapping.

7.1 Mobile Mapping

In conventional TCP/IP host systems, the relationship between a TCP association and a TCP connection is 1:1. When application processes request to open or close an TCP association, in response, the socket layer underneath will request the transport layer to setup or tear down a corresponding TCP connection correspondingly. The association and connection can be considered as to have the same life time.

Before the discussion of mobile mapping between a TCP association and a TCP connection, two terms have to be clarified. A **home-IP** is an IP address assigned to a mobile host permanently when the mobile host resides in its home network. A **current-IP** is an IP address temporarily assigned to a mobile host when it moves to different guest network.

Our strategy is briefly given as follows.

- Keeping TCP associations unchanged. Since the home-IP address is a unique identifier of a mobile host, we use home-IP addresses in TCP association even when the mobile host is in a guest network. The TCP association is independent of the location of the mobile host.

- Resetting TCP connections when mobile hosts move to a new location. The current-IP addresses are used in real TCP connections.

- Hiding any loss of TCP connections from applications by introducing mobile mapping mechanisms between TCP associations and TCP connections.

MSL provides an uninterrupted TCP association for TCP sockets between two MSL supported systems. When a mobile host moves, the TCP connection will be broken. However, both MSLs will continue to support socket I/O operations as long as sending buffer is not full, and make the TCP association alive. When the mobile host arrives at a new location, a TCP connection will be automatically reconnected by two MSLs, and then the queued I/O requests and new I/O requests are performed through the new TCP connection. In such circumstances, one association may be mapped to more connections. However, at the same time, the relationship between two sockets is constant. The Figure 15 schematically depicts the life time of TCP association and TCP connection.

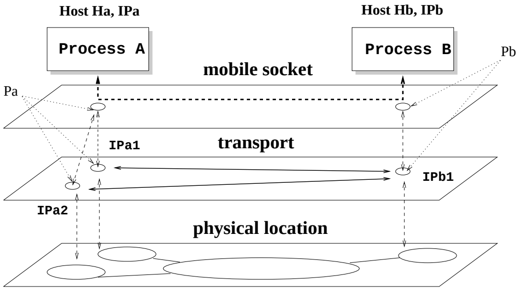

Figure 16: Mobile Connection

Figure 16 shows the mapping between a TCP (socket) association and a TCP (transport) connection. If a process A on mobile host H_a needs to talk to process B at mobile host H_b via TCP, a TCP association can be established between H_a and H_b at mobile socket level as $\{TCP, IPa, Pa, IPb, Pb\}$. Where both IP_a and IP_b are home-IP addresses. If a mobile host moves away from its home network, a TCP connection is broken and a new one has to be reconnected later. TCP connection must be reconnected using current-IP address. The mapping is from a TCP association

$$\mathit{(TCP,\ home\text{-}IP_A,\ Port_A,\ home\text{-}IP_B,\ Port_B)}$$

to a TCP connection

$$\{\mathit{current\text{-}IP_A,\ Port_A,\ current\text{-}IP_B,\ Port_B}\}$$

Note that the port numbers need not changing. As depicted in Figure 16, if mobile host H_a moves to a new location, its current-IP address changes from IPa_1 to IPa_2. Although the TCP socket association is still alive when moving, the old TCP transport connection, $(TCP, IPa_1, Pa, IPb_1, Pb)$, will be taken over by the new TCP transport connection, $(TCP, IPa_2, Pa, IPb_1, Pb)$. The data flow between mobile sockets is continuous through the new connection.

7.2 Issue 1: Distinguishing Old and New Connection

Whenever a mobile host changes its location and therefore its current IP address, the opening TCP socket association must be mapped on to the new TCP connection using the new current IP address accordingly. However, implementation of such a mapping and re-mapping an open association to a new connection is challenging. The major contribution to the complicity is the reuse of port numbers. Both server and client systems may reuse a local port number to make different connections with different peer sockets.

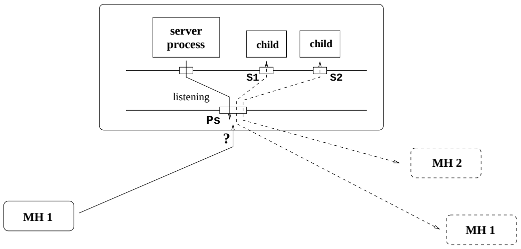

Figure 17: Reuse of Port Number

The mechanisms of reusing port numbers make it difficult to map a mobile TCP association to a new TCP connection. For example, as in Figure 17, a server program is running on a fixed host. The port number P_s is used to connect with two mobile systems. Suppose that MH_1 and MH_2 are talking to socket S_1 and S_2 on the fixed host. The server is also waiting for new incoming requests. Suppose that MH_1 moves away from current network. Then its TCP connection will be forced to terminate, but its corresponding TCP association will be kept alive. However, the server has no way to determine whether a new connection request is either a new request or a reconnection from MH_1, because the incoming request will have a new source IP address.

Another potential problem is from the reuse of the temporary current IP addresses. For example, a server has a TCP connection with a mobile host, MH_i, using a temporary current IP address IP_{curr}. After a while, MH_i moves away and leaves the server in the state of waiting for reconnection from MH_i. The IP_{curr} may be allocated to another mobile host, MH_j, which may try to set up a connection with exactly the same IP address and port number at server as used by MH_i. In spite of the same IP address and port number, the two mobile hosts shall be treated differently.

7.3 Issue 2: Keeping the Same Semantics

To keep the same TCP socket I/O semantics in our mobile socket environment is not simple. At first glance, a TCP association is mapped to a TCP connection and the semantics should be maintained by TCP entities which is supposed to be remained unchanged in mobile environments. However, a potential problem occurs when closing of connection. In a non-mobile network environment, a TCP association is always mapped to a single TCP connection. The only possible loss of data without further warning is at the end of communication. Whereas a mobile TCP association might be mapped to many TCP connections in different time periods (referring to Figure 15). At all ends of these connections, data may be lost.

There two possible cases. First, a mobile host tries to close TCP connection just before moving. The TCP entities at both ends will try to send out all data kept in the sending buffers before closing. But this process will take time. If a mobile host moves before completion of active/passive close, both sides have no idea whether the data remaining in TCP sending buffer have been successfully sent out. There are two further possible cases: a) the data have already been sent to the correspondent site and the only loss is the acknowledge which may be lost or has not been sent out; and b) the data have lost on the way toward the remote site. Therefore, the data cannot be simply resent when a new TCP connection is setup. Second, the network connection of mobile host gets lost in a non-voluntary way, so mobile hosts have no time to close their current TCP connections and are forced to abort it. For example, a user unplugs the network cable, or inadvertently goes beyond the covered area of current wireless LAN. In such cases, old TCP connections are aborted at both sides which detect loss of network connection or any network changes. Both systems do not know whether data in the sending buffer are correctly received by the remote side.

To mobile TCP association, any kind of loss of data or any suspicious situation might occur in the middle of life cycle of mobile association. This could change the TCP socket I/O semantics. Data might be lost when system moves consequently.

7.4 Issue 3: Distinguishing Between Different TCP Services

The MSL-supported systems are expected to communicate with both conventional and peer MSL-supported TCP systems so as to allow all applications to communicate without hassles of dealing with different systems and different services. Two solutions are possible. One is to change the conventional socket API interface that supports both conventional and mobile TCP services to applications. The other is to find ways to detect whether a request is from a conventional system or a MSL-enabled system. The former request to change the socket API syntax and therefore the semantics. Consequently, existing applications can not be able to access our mobile TCP services without recompilation. The latter incurs unnecessary system overhead when applications do not need mobile service.

8 Virtual Port

Our solutions to the first two issues mentioned in the previous section are introducing the concept of virtual port which is an additional port between socket and TCP port as shown in Figure 18. A virtual port acts as a TCP port from the viewpoint of a socket. When an application binds a socket to a TCP port, MSL creates a virtual port, and binds the socket to it accordingly. The mobile TCP association is established between two such virtual ports. All code above the virtual port is the same as used in the current socket layer, and extensive mobile functionality is hidden by a virtual port.

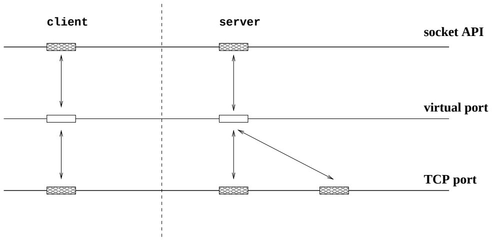

Figure 18: Virtual Port

When created, a virtual port is assigned to a home-IP address and a local port number. Therefore, what the socket sees is a TCP port with consistent permanent address. MSL maps a virtual port associated with a home-IP address and a port number, to a real TCP port associated with current-IP and port number, dynamically[5]. The TCP connection is set up between two real ports by conventional TCP/IP protocol stacks. Furthermore, like TCP ports in the current TCP/IP system, a virtual port can be shared by more than one sockets.

The following information must be managed in MSL system, namely, a six tuple $(A, V_{id}, P, B, B_n, F)$. A is a TCP association, represented as:
$(TCP, \text{home-IP}_1, \text{port-number}_1, \text{home-IP}_2, \text{port-number}_2)$.
As mentioned before, it is difficult to distinguish reconnection requests from connection requests due to port reuse at the server side. Hence an identifier, V_{id}, is added to identify a unique virtual association from all associations through the same virtual port. V_{id} itself does not need to be unique in Internet wide. The V_{id} is assigned by the server virtual port when a client cite requests to initiate a connection. The V_{id} will be passed to the client. Later, the client cite uses the V_{id} as an additional information to reconnect to the server virtual port. P is a connection between two addresses, namely, $(\text{current-P}_1, \text{port-number}_3, \text{current-P}_2, \text{port-number}_4)$. The home-IP is the permanent IP address for a mobile host, and current-IP is a temporary IP address for the mobile host. The MSL system maps home-IP addresses used in A to the corresponding current-IP addresses used in P. The port number in general can be different. F is used to indicate the type of local socket, either client or server. B is a sending buffer being managed inside of the MSL system. B_n is a serial number to indicate the serial number of data transfer. It starts from 0 at the beginning of mobile association.

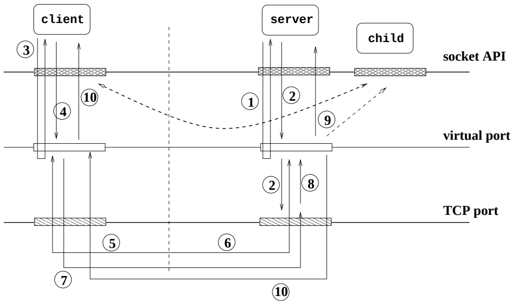

Figure 19: A Simple Example

8.1 Procedure for Connection

The procedure for connection between two sockets is given in this section. We illustrate the steps in Figure 19, assuming that the server system is a fixed host.

- **Step 1:** A server application creates a socket and binds it to a local address, $(\text{home-IP}_1, \text{port-number}_1)$. The MSL system at the server site creates a virtual port using the same address, accordingly.
- **Step 2:** The server application passively opens the socket. Correspondingly, the MSL system at the server site opens a passive TCP port, $(\text{current-IP}_1, \text{port-number}_1)$, and waits for incoming request. Here current-IP$_1$ is the same as home-IP$_1$.
- **Step 3:** A client application completes creating and binding its local socket, $(\text{home-IP}_2, \text{port-number}_2)$. The MSL system at the client site creates a virtual port using the same address as server does in Step 1.
- **Step 4:** The client process then initiates a connection procedure to establish a TCP association with the server socket. The address of the server socket given by the client process is a permanent address.
- **Step 5:** The MSL system at the client site maps both local and server permanent addresses to corresponding current TCP socket addresses, and starts a regular TCP three-way handshake to

[5] Note that the port number at server side might be changed, we will discuss this later.

set up a real TCP connection between two virtual ports at both the client and the server sites at first. This procedure is not visible to application processes at both sides.

- **Step 6:** The MSL system at the server site will then accept the TCP connection from (current-IP$_1$, port-number$_1$) to (current-IP$_2$, port-number$_2$) for further exchange of additional MSL control information. Noting that the TCP association between two sockets has yet to exist at this stage.
- **Step 7:** By the real TCP connection, the MSL system at the client site sends its virtual port address including additional information to its peer MSL system at the server site to indicate that it is an initial connection request.
- **Step 8:** Based on the information provided by the MSL at the client site, the MSL system at the server site decides whether or not to accept this request according to the parameters passed from the upper layer at the server site.
- **Step 9:** If this request is accepted, a new socket is created at the server site, but not a new virtual port. The MSL system at the server site will assign a V_{id} to this new association and also inform the MSL system at the client side of the acceptance.
- **Step 10-a:** The MSL system at the client side keeps the V_{id} internally and informs the client application of the successful connection.
- **Step 10-b:** If the server application refuses the connection at Step 9, the MSL system at the server site sends a deny information back to the MSL system at the client site. Then client MSL returns a error code to client.

Obviously, the major difference of the connection procedure between the MSL systems and the conventional systems is that a real TCP connection is set up before the socket association and before the connection between virtual port is accepted.

Next, we briefly describe how to implement the above steps using the API for the standard socket system calls. At the server side, for Step 1, a `socket()` creates a socket structure and a `bind()` assigns a local permanent address to the socket. For Step 2, the server calls both `listen()` and `accept()` to passively open the socket. The `accept()` will create a virtual port locally, and the MSL maps the virtual port address to a real TCP address. This `accept()` will return when a virtual port is accepted at Step 9. At the client site, for Step 3, the client process calls `socket()` to create its own local socket. The client can call `bind()` to explicitly assign an address to local socket. If the client does not call `bind()`, `connect()` will implicitly assign such an address, at Step 4. To both the server and the client processes, all the steps between Step 5 to Step 8 is transparent. The additional mobile operations are completed in `accept()` and `connect()` system calls. When these two system calls return successfully, a mobile TCP association is created.

The reconnection procedure will be the similar. The server virtual port is always waiting for incoming request from the underlying TCP port. The MSL system at the client site will send different request messages with a V_{id} to the MSL system at the server site. Hence the MSL system at the server site understands whether or not it is a reconnection, and is able to map it to correct destination socket.

The server application itself may be ceased before its child processes and its socket is closed. Normally, there is no alive socket listening on the corresponding TCP port. In other words, all sockets associated with this TCP port are created for child processes and are not in a passive open mode. Any incoming requests to this TCP port will be refused by the TCP layer. However, as for MSL, the processing is different in a certain circumstance. If there is a socket, which has active association with a client process which is not currently connecting to the TCP connection due to its movement, the MSL system at the server site will still passively open the underlying TCP port, and wait for reconnection requests.

The MSL systems have to distinguish the close of last TCP connection from all previous ones as well. At the last connection, the virtual ports and sockets at both sides need to be deleted. Otherwise, the virtual ports must be kept alive. When the client or server applications need to close an association, the MSL systems will explicitly close the TCP connection and will delete such virtual ports and associations.

8.2 I/O Semantics

The sending buffer, B, is aimed at two targets: providing continuous I/O services and remaining the TCP I/O semantics unchanged.

As for providing continuous I/O services, the local data buffering scheme is given as follows. If the underlying TCP connection between the two real TCP ports exists and functions, the virtual ports at both sides will not buffer any data. All data being sent out will be put into the TCP sending buffer directly. During the movement, there is no real TCP connection, and no I/O requests can be performed. The MSL system is expected to buffer data in its own sending buffer. These buffered data will be sent out when a new TCP connection is created later on. A virtual port can reject data when its buffer is full. The rejection is not an exceptional case by all means. For instance, in normal socket communications, network congestion can occur. In this case TCP cannot send more data when the sending window is full and the consequent sending requests will fail. The loss of underlying TCP connection looks the same as loss of network bandwidth to application processes in terms of I/O failures.

Regarding TCP I/O semantics, TCP uses both sliding window and acknowledgement mechanism to ensure correct delivery of all data being transfered between two TCP ports over unreliable networks. When a TCP connection is being closed, either side may discard data in its sending buffer. Hence, TCP associations are reliable during the data transfer period but there is no guarantee during the closing time. In mobile association environments, there might be more that one TCP connections to convey data between a pair of sockets, and exist no mechanisms to ensure that these TCP connections can be closed in a complete manner.

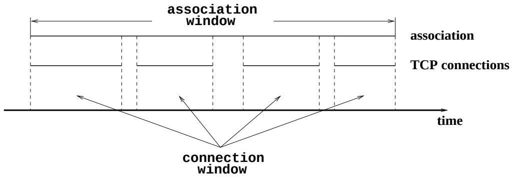

Figure 20: Association Window and TCP Window

To keep the I/O semantics unchanged, besides the buffer, B, we introduce a new mechanism, called association window, which spans over entire association lifetime and is independent of TCP window mechanism as shown in Figure 20. TCP windows make sure that data can be correctly transfered in single TCP connection, whereas association windows are aimed at the correct data transfer during entire lifetime of a mobile association. Association window gives all data being transfered in one direction a sequential number, B_n, like TCP window does. The initial number is always from zero. At each beginning of re-establishing underlying TCP connection, both virtual ports will exchange the sequential numbers in association window to indicate how many bytes in incoming data stream have been correctly received, hence the peer systems can resume transfer from correct starting points. Virtual ports will keep records of the sequential numbers of both last byte sent to the TCP layer and the last byte received, these numbers are updated when each time virtual port sends data to TCP sending buffer or receives data from TCP receiving buffer. Given the loss of data in TCP communications can only occur at the end of each TCP connection, only these data in TCP sending buffer might be lost. Therefore, MSL fetches back the remainder of data in TCP sending buffer into virtual port buffer, B, at the end of each TCP connection. Upon reconnecting, virtual ports will re-send all or partial portion of data in the buffer, B, according to the requested sequential number B_i sent by remote side. Both B and B_i are recorded in the aforementioned 6-tuple of virtual port.

8.3 Virtual Port Protocol

The Virtual Port Protocol(VPP) is a protocol used between a pair of virtual ports. The basic tasks of VPP are demultiplexing and mapping the incoming connection requests to proper associations and resynchronising data transfer after reconnections.

There are four types of Protocol Data Units (PDU) used in VPP, as given in Figure 21. After setting

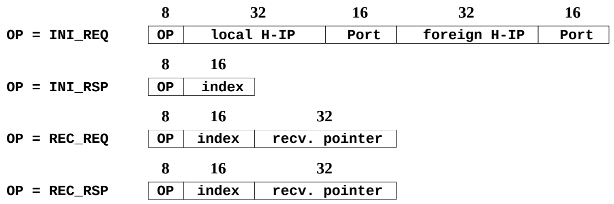

Figure 21: Protocol Data Units

up the underlying TCP connection between the pair of virtual ports, VPP PDUs are exchanged before any data transfer of application processes.

First, the MSL at the client site sends a `INI_REQ` with the definition of virtual connection *(local home-IP, local-Port, peer-home-IP, peer-port)*. The MSL at the server site then finds that the incoming request is for a new connection. According to the condition specified in the server application's `accept()`, the MSL system at the server site decides whether or not to accept this request. The result is sent back to the client in `INI_RSP` PDU. If the connection is refused, the value of V_{id} is assigned to a negative number, and underlying TCP will then be closed by server system immediately as well. If the request is accepted, V_{id} will be a positive number as the valid and unique association identifier among all associations of this virtual port at the server site. Afterwards, all incoming data will be directly passed to the sockets by virtual ports, and not further control information is embedded into the normal outgoing data stream as well.

When a client moves into a different location, the MSL system at the client site uses the V_{id} to reconnect. This request is sent in `REC_REQ` UDP. And the MSL system at the server site will recognise the reconnection request and find the corresponding association followed by replying a `REC_RSP` message. `REC_RSP` indicates that the current underlying TCP connection is successfully mapped to the destination socket. Like `INI_REQ`, if the MSL system at the server site cannot find a V_{id}, a response with an invalid V_{id} will be sent back. During reconnection period, both sides need to exchange the B_ns to inform the other side of the position from which data transfer should start.

8.4 The API of MSL

MSL provides two types of TCP services, namely, regular and mobile services at both server and client sides. As for the API of MSL, there are two solutions. One is to define a new API for a mobile protocol supported in the internet domain, say, `IPPROTO_MTCP`. The other is to use the existing internet domain, by which MSL must be able to handle both regular and mobile services inside the MSL layer. The advantage of the later approach is the transparency at the expense of extra cost for handling regular TCP services. Due to the compatibility issues, we adopt the latter approach for our prototype system.

Suppose that we use the current internet domain. Then, the next issue is how to make a regular/mobile connection. For example, the MSL at the client site always attempts to make a mobile connection with a remote virtual port following the VPP protocol, which defines the PDU's and exchange procedures. However, the server system may not support VPP. Therefore, the establishment of underlying TCP connection may have different meanings to the client and the server. The client will use VPP, whereas the server will consider the connection between two sockets has been completed and bypass PDUs up to the server application processes. Two solutions are available. The first solution is to use an additional TCP option. The client virtual port sends `INI_REQ` to normal well-known server port with a new TCP option which inform the server that the client system wants to set up a mobile connection. If the server system is MSL-enabled, it will correctly understands and responds such request with another new mobile TCP option. If the server system does not support mobile service, it must discard it and complete normal TCP connection according to the requirement to TCP/IP host system[16]. The second solution is to use the so called magic number. After establishing a underlying TCP connection, the MSL system at the client site sends the server a magic number at the beginning of data stream which is followed by the VPP `INI_REQ` or `REC_REQ` PDU's. From the magic number, the server will understand that the client system needs mobile services. Otherwise, a regular association is set up. As for the first approach, it requires to enhance the TCP layer. We adopt the magic number approach for our prototype system.

9 An Example

In this section, we demonstrate the functions of the virtual port protocol using an example.

Initial Connection of Association and Underlying TCP

Figure 22 shows the simplified procedure of initial connection between two sockets, namely, a server and a client.

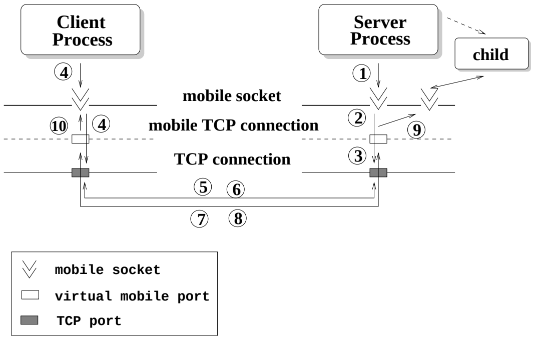

Figure 22: Initial Mobile Connection

In the first step, the server process creates a socket and binds it to a local permanent address (home-IP_S, $Port_S$). In Step 2, the MSL system at the server site creates a virtual port for this socket and translates the permanent address into a local current address, (current-IP_S, $Port_S$). Then server process listens on the socket in Step 3. Inside the MSL system, the virtual port listens to the real local TCP port on behave of the socket. In Step 4, the client process creates and binds an active socket to an assigned local address. The MSL system at the client site creates a underlying local virtual port, (home-IP_C, $Port_C$). Then, the client process initiates a connection with the server process using its permanent address in Step 4. The MSL system at the client site maps both permanent addresses, for the client and the server hosts, into current IP addresses, and then attempts to set up a regular TCP connection between these two current IP addresses, (current-IP_C, $Port_C$, current-IP_S, $Port_S$), in Step 5. In Step 6, the server virtual port accepts the connection request so that a regular TCP connection can be set up. The virtual port protocol entities exchange information between the pair of virtual ports in Step 6, 7 and 8, as depicted in Figure 23.

As shown in Figure 23, the client virtual port first sends an initial request (`INI_REQ` message) to the server virtual port, which contains both local and remote permanent addresses. The server will check to see if the request is destined to a correct system and if there is a running server. Suppose that the server MSL positively replies the request with `INI_RSP`. The 16-bit index V_{id} in `INI_RSP` is assigned for later reconnection. Typically five TCP segments are exchanged to establish a virtual connection. But we can combine `INI_REQ` with `SYN`, since it is legal under TCP protocol with most TCP implementation[21] The `INI_REQ` and `INI_RSP` messages are used internally in MSL and will not be delivered to the upper layers. If the server system rejects this virtual connection, it simply closes the real TCP connection. The client system will know the request is rejected and all internal data structures will be freed at both systems. Note that unlike TCP, no initial sequential number for association window needs to be exchanged, because it always starts from zero. Each of the two virtual ports maintains two pointers for receiving and sending association windows. The values of pointers indicate the sequential number of the last byte received and sent form and to TCP port. Initial values of these pointers are zero.

In Step 9 and 10, both the client and the server virtual port inform their corresponding sockets of the successful connection. Then, the underlying TCP connection is switched to normal data transfer

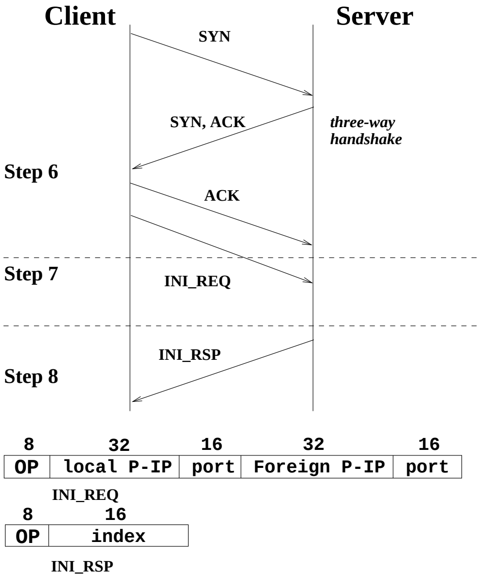

Figure 23: Initial Connection of Virtual Ports

status.

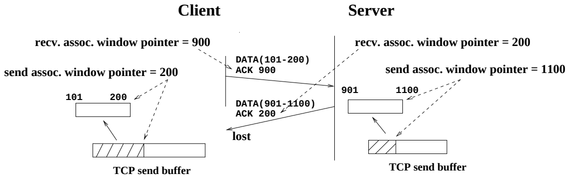
Client
Server
recv. assoc. window pointer = 900
send assoc. window pointer = 200
101
200
TCP send buffer
DATA(101-200)
ACK 900
DATA(901-1100)
ACK 200
lost
recv. assoc. window pointer = 200
send assoc. window pointer = 1100
901
1100
TCP send buffer

Figure 24: Loss of TCP Connection

Suppose that at later stage the client system starts to move and the TCP connection is lost between these virtual ports, as shown in Figure 24. Further assuming that the client received the first 900 bytes from the server, and sent the first 200 bytes into TCP port. We also suppose that the server has received all the 200 bytes, and sent 1100 bytes to the client. Here, the 901st byte to 1100th byte of data being sent by the server are lost. After the TCP connection is broken, both sides fetch the remaining data in the sending buffers of the TCP ports to their virtual port buffer. In the client, there are 100 bytes left in the TCP sending buffer due to the fact that the last IP datagram carrying the ack information from the server was lost. In the server, 200 bytes remain in the TCP sending buffer, and these data are totally lost.

Reconnection of Underlying TCP

When the client arrives at its new location, new TCP connection will be re-established to resume the communication between these two virtual ports. In our VPP, the client system always actively tries to connect to the server system. Envision that client system moves from network N_1 to a new network N_2, and current IP address is IP_{C_2}.

As shown in Figure 25, the server always listens on $Port_S$ if there are any alive virtual ports associated with it (Step 1). After the client is ready to communicate at its new location, it starts a *three-way handshake* to establish a new TCP connection with the server virtual port at first. The new TCP connection between these two virtual port is (current-IP_{C_2}, $Port_C$, current-IP_S, $Port_S$). The server will accept this connection first at the real TCP port level which is done in Step 2. In Step 3, the client virtual port sends reconnection request to the server virtual port with V_{id} and its receiving association window pointer. The server virtual port then looks up its association list, and checks to see if there is a

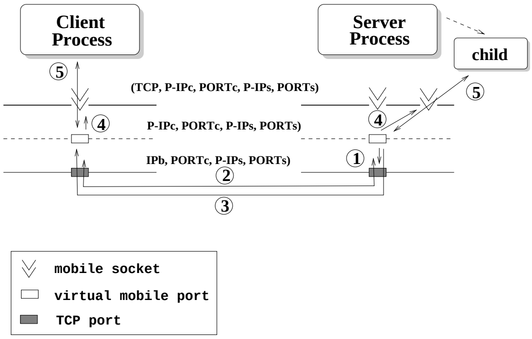
Client Process
Server Process
child
5
(TCP, P-IPc, PORTc, P-IPs, PORTs)
5
4
P-IPc, PORTc, P-IPs, PORTs)
4
IPb, PORTc, P-IPs, PORTs)
1
2
3
mobile socket
virtual mobile port
TCP port

Figure 25: Reconnection of Virtual Port

pending mobile association to wait for reconnection. If successful, a response is sent back to the client to inform the success of a reconnection (Step 3). Then, in step 4, both virtual ports will resend the data in the buffer of virtual port at first, and then continue the I/O services provided to sockets (in Step 5).

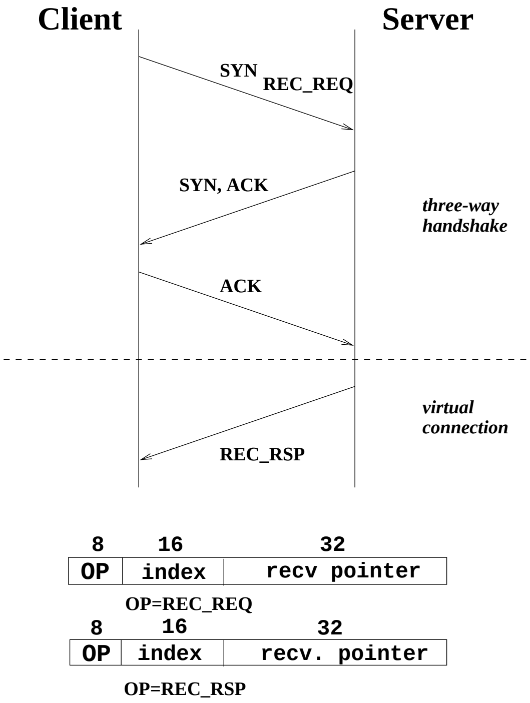

Figure 26: Reconnection of Virtual Port

During the reconnection, the messages changed between virtual ports are depicted in Figure 26. Firstly, the *three-way handshake* establishes a TCP connection between the two virtual ports. Then, the client virtual port sends a `REC_RES` to the server virtual port which contains the V_{id} and the receiving pointer of its receiving association window of 900. The server responses a `REC_RSP` with a pointer of 200 from which client virtual port understands that there is no need to resend the data. The server virtual port has to resend the data to the client starting from 901.

If the server system moves, or even both systems move during the same period, the reconnection procedure will be similar to the case when only the client system moves. The server system always passively listens on a certain TCP port. When the client system arrives at a new location, it will attempt to connect the server system at server's previous IP address. If it fails after timeout or ICMP error message goes back, the client system will try to fetch server system's new current IP address and try again until the re-connection is successfully completed or the association between mobile sockets is broken due to failure of connection.

System calls getpeername and getsockname will be used if a process enquires the IP address from DNS name. In a mobile environment, we need emphesize that an IP address returned by these two system calls is a permanent IP address. application processes are shielded from current IP address at all.

10 Conclusion

In this paper, we described the mobile TCP socket, one step of our two step solution to support mobility. In our mobile TCP socket, a mobile mapping is introduced which maps TCP associations to TCP connections. The mobile mapping can be implemented in the socket layer and on top of TCP/IP layers. We showed the implementation details in this paper as well. Our approach achieves high compatibilities with the current TCP/IP protocols due to the following reasons. First, we adopt the same API of the socket layer. Second, we do not require any special functionality in the network or the transport layers to support mobility. Finally, we do not require all networks, where mobile hosts may visit, to support special routing mechanisms. In terms of performance, our mobile solution can reduce both the propagation cost for distributing the location information of mobile hosts and the forwarding cost for forwarding IP datagrams across the Internet. As a future work, we will attempt to conduct a performance study on the two costs mentioned above.

References

[1] Ajay Bakre and B.R. Badrinath. I-TCP: Indirect TCP for mobile hosts. Technical Report DCS-TR-314, Department of Computer Science, Rutger University, Piscataway, NJ 08855, USA, October 1994.

[2] Stuart Cheshire and Mary Baker. Internet mobility 4x4. *ACM Computer Communications Review*, 26(8), August 1996.

[3] C. Perkins (editor). IP mobility support, version 17. IETF Internet Draft, May 1996.

[4] John Ioannidis, Dan Duchamp, and Gerald Q. Maguire Jr. Ip-based protocols for mobile internetworking. *ACM Computer Communications Review*, 21(5):235–245, September 1991.

[5] John Ioannidis and Gerald Q. Maguire Jr. The design and implementation of a mobile internetworking architecture. In *Proceedings of 1993 Winter USENIX Conference*, San Diego, CA, USA., January 1993. USENIX.

[6] David B. Johnson. Mobile host internetworking using loose source routing. Technical Report CMU-CS-93-128, School of Computer Science, Carnegie Mellon University, 1993.

[7] David B. Johnson. Scalable and robust internetwork routing for mobile hosts. In *Proceedings of the 14th International Conference on Distributed Computing Systems*. IEEE Computer Society, June 1994.

[8] David B. Johnson and Charles Perkins. Routing optimisation in mobile IP. Internet Draft Standard.

[9] Andrew Myles and David Skellern. Comparing four IP based mobile host protocols. *Computer Networks and ISDN Systems*, 26:349–355, 1993.

[10] Baiju V. Patel, Partha Bhattacharya, Yakov Rekhter, and Arvind Krishna. An architecture and implementation toward multiprotocol mobility. *IEEE Personal Communications*, 2(3):32–42, June 1995.

[11] C. Perkins and Y. Rekhter. Draft RFC, July 1992.

[12] C. Perkins and Y. Rekhter. Support for mobility with connectionless network layer protocols (transport layer transparency). Draft RFC, January 1993.

[13] Charles Perkins, Andrew Myles, and David Johnson. IMHP: A mobile host protocol for the internet. *Computer Networks and ISDN System*, 27:479–491, 1994.

[14] Charles Perkins, Andrew Myles, and David B. Johnson. The internet mobile host protocol (IMHP). In *Proceedings of INET'94/JENC5*, 1994.

[15] Xun Qu, Iain Macleod, and Hong Jiang. A practical method to achieve portable communication in internet context. In *Proceedings of GlobeCom'95*, pages 1512–1516, Singapore, November 1995. IEEE.

[16] Requirements for internet hosts — communication layers, RFC 1122, October 1989.

[17] Internet protocol(IP), RFC 791, September 1981.

[18] Transmission control protocol, RFC 793, September 1981.

[19] W. Richard Stevens. *UNIX Network Programming.* Pretice Hall, 1990.

[20] W. Richard Stevens. *TCP/IP Illustrated, Volume 1, The Protocols.* Addison-Wesley, 1994.

[21] W. Richard Stevens. *TCP/IP Illustrated, Volume 3, TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols.* Addison-Wesley, 1996.

[22] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro. A network architecture providing host migration transparency. *ACM Computer Communications Review*, 21(1):209–220, January 1991.

[23] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, volume 2, The Implementation.* Addison-Wesley, 1995.