Implementation of a Multihoming Agent for Mobile On-board Communication

Jun Yao, Yi Duan, Jianyu Pan, Kun-chan Lan, Mahbub Hassan
University of New South Wales
Sydney, NSW 2052, Australia
{jyao713, ydua280, jpan122, klan, mahbub}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-0508

Mar 2005

Abstract

The idea of using multiple access links (so called multihoming) is commonly used to improve the aggregate bandwidth and the overall service availability, which has been employed by large enterprises and data centers as a mechanism to extract good performance and enhance the network reliability from their service providers for a while. However, in the context of network mobility, because of the diversity and unreliability of the wireless links, adaptable multihomed prototype supporting mobile networks should be studied and implemented.

This report introduces us the details from the implementation project of a multihoming agent, which is developed on a Linux based platform and allows router to access and switch between multiple access links including WLAN and GPRS. The prototype is capable of dynamically switching users' data traffic from one network interface to another, i.e. Ethernet, WLAN, and GPRS, without breaking transport layer sessions, based on the interfaces' status and policy in use.

1 Introduction

The idea of using multiple access links (so called multihoming) is commonly used to improve the aggregate bandwidth and the overall service availability, which has been employed by large enterprises and data centers as a mechanism to extract good performance and enhance the network reliability from their service providers for a while.

However, in the context of network mobility, because of the diversity and unreliability of the wireless links, adaptable multihomed prototype supporting mobile networks (such as on-board LANs) should be studied and implemented. For this sake, this report would mostly address how to dynamically make use of the multi WAN interface base on the given policies and the real-time metrics collected from each WAN links, and how to response to the WAN outages, in such a way provides a more efficient, reliable and robust connectivity to the end-user in on-board LAN.

The objective of this paper is to develop and implement a multihoming agent on a Linux based platform to support the future on-board multihomed routers. This multihoming agent allows router to access and switch between multiple access links including WLAN and GPRS. The prototype would be capable of dynamically switching users' data traffic from one network interface to another, i.e. Ethernet, WLAN, and GPRS, without breaking transport layer sessions, based on the interfaces' status and policy in use.

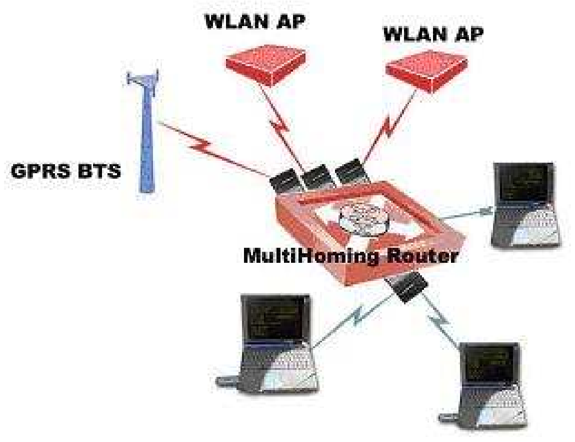


Figure 1: Proposed Prototype

The rest of the paper would be organized as follows; Chapter 2 would mainly study the past and related work to this project. Chapter 3 would illustrate the whole architecture of this prototype. In Chapter 4, Chapter 5 and Chapter 6, 3 main components of this prototype, namely Interface Metrics Collection Layer, Interface Selection Layer and Load Distribution Layer, are discussed respectively. In Chapter 7, the testbed and some relevant test results are shown to evaluate the proposed daemon. Chapter 8 is the last chapter to make the conclusion and suggestion for the future works.

2 Related Works

Current [1] mobile devices are often equipped with several network interfaces, which may be of different access technologies, both wireless and cellular. Different requirements of different applications can result in a different preference of the interface that should be used. Network connections should be placed in the best possible interface based on these requirements. During communication, changes in the availability or characteristics of an access network behind an interface may result in change of preference of interfaces for new connections and a situation where already established connections should be moved from one interface to another.

Several Internet drafts [2], [3], [4], [5] introduce the requirements and definitions for IPv4 and IPv6 multihoming. They present a few implementation proposals for interface selection in the host multihoming that would allow explicit policy definition. They also identify the interface selection problem and mention that the selection should base on some policies. But, those drafts do not give detail on policy-based interface selection system.

Ylianttila et. al in [6] present a handoff case study between GPRS and WLAN based on mobile IP. They have discussed about procedures, algorithms and metrics involved in handoff in heterogeneous wireless networks. A general level procedure was given for inter-technology roaming between WLAN and GPRS. They envision that WLANs should be embedded into wide area wireless data networks since the wideband wireless services will not have a comprehensive coverage. In the implementation, the handoff information is gathered at link layer (e.g., signal strength) and transmitted to a daemon program on the application level for decision making. Handoffs are made on account of implicit rules utilizing fuzzy logic.

Jukka Ylitalo et al present in [1] an interface selection architecture, which allows a user to dynamically create and modify interfaces selection policies and thus control how the network interfaces are used in a multihoming environment. The architecture makes it possible to define policies for different connections on account of user preferences. Each connection is bound to a profile that contains local routing rules. Therefore, it is possible to make a vertical handoff to a single connection or to a group of connections without affecting any other connections that are using the same interface.

Pablo Rodrigueze et al present in [7] a Mobile Access Router that utilizes multiple wireless access links to aggregate bandwidth and provide local users with a smoother, more reliable access network than can typically be provided by a single cellular link. A policy-based interface selection system is also implemented. Based on a particular MAR scheduling policy, MAR selects a given interface for each packet or request based on NAT technique. Moreover, the limitations of the current wireless access systems in the wide-area are discussed. A case study for exploiting the network diversity in wireless access is provided. Experiments with production networks show that there is a substantial overlap in terms of coverage being offered by many of these operators and also across networks.

3 Prototype Architecture

3.1 On-board mobile router

The idea of on-board router provides a set of local interfaces and wireless interfaces. The local interfaces provide access to local users/clients (on-board LANs), which can include both wire and wireless technologies to provide connectivity to laptops and other devices. On-board router provides a DHCP server that dynamically assigns private IP address to on-board LAN users from a pool of addresses. Local users are configured to use the proposed multihoming on-board router as their proxy or gateway.

Besides these local interfaces, on-board router also runs several wireless interfaces. This proposed multihoming agent system could automatically connect to Internet through all pre-configured wireless networks. IP addresses are assigned to each interface by corresponding wireless service provider.

3.2 Multihoming Agent

To be able to route requests originated from local clients through on-board router, the NAT mechanism is used to perform the data striping by the multihoming agent. Based on a particular policy and collected WAN status, system selects a given interface for each packet or connection. Therefore, this prototype performs bandwidth aggregation across multiple WLAN and GPRS network interfaces, which provides a faster, more robust, and reliable communication channel to local users. The core components of this system are shown in Figure 2, and they are embedded into 3 different layers, namely,

a) Interface Metrics Collection Layer
b) Policy Processing Layer
c) Traffic Distribution Layer.

As a whole, this multihoming agent could continuously monitor interfaces' status, and collect metrics such as Loss Rate, Signal Strength and Bandwidth respectively. After that, Interface Selector would process the metrics above according to the policy used and commands the traffic distribution module in kernel to redirection traffics go through the router using NAT techniques. Of course, multihoming agent could perform vertical handoffs (failovers) without breaking connection session when particular interface's unavailability is detected.

4 Interfaces Metrics Collection Layer

According to the design of this system, interface status detection is the first and necessary step for this project. After that, the interface selection system can make the right decision based on the various policies and interfaces status, and then produce the proper instructions to the load distribution function. Finally, the load distributor will assign different connections to the proper interfaces to achieve the load balance.

As mentioned in the previous part, the on-board router provides a number of Wireless interfaces that can accommodate a variety of wireless technologies (e.g. WLAN 802.11b, GPRS etc.). The router automatically connects to the pre-configured networks. It is known to all, signal strength is one of the most important elements, which can indicate the network status. In addition, loss rate and available bandwidth are also key standards for reflecting the network performances.

4.1 Signal Strength Collection (WLAN)

The first WLAN standard, IEEE 802.11 is based on radio technology operating in the 2.4 GHz frequency and has a maximum throughput of 1 to 2 Mbits per seconds. The general idea of WLAN was basically just to provide a wireless network infrastructure comparable to the wired Ethernet networks in use. Currently, the most spread and deployed standard is the IEEE 802.11b. As a fact of this, an 802.11b based WLAN card is chosen as one of the equipments of this implementation.

To obtain the signal strength, one way is to modify the driver to capture all kinds of 802.11b frames in the driver and then retrieves the signal strength from the relevant frames. This is a good idea, however, it will be driver dependent, this proposed system would not work whenever the WLAN card is changed or the driver is updated. What needed is a common way to get

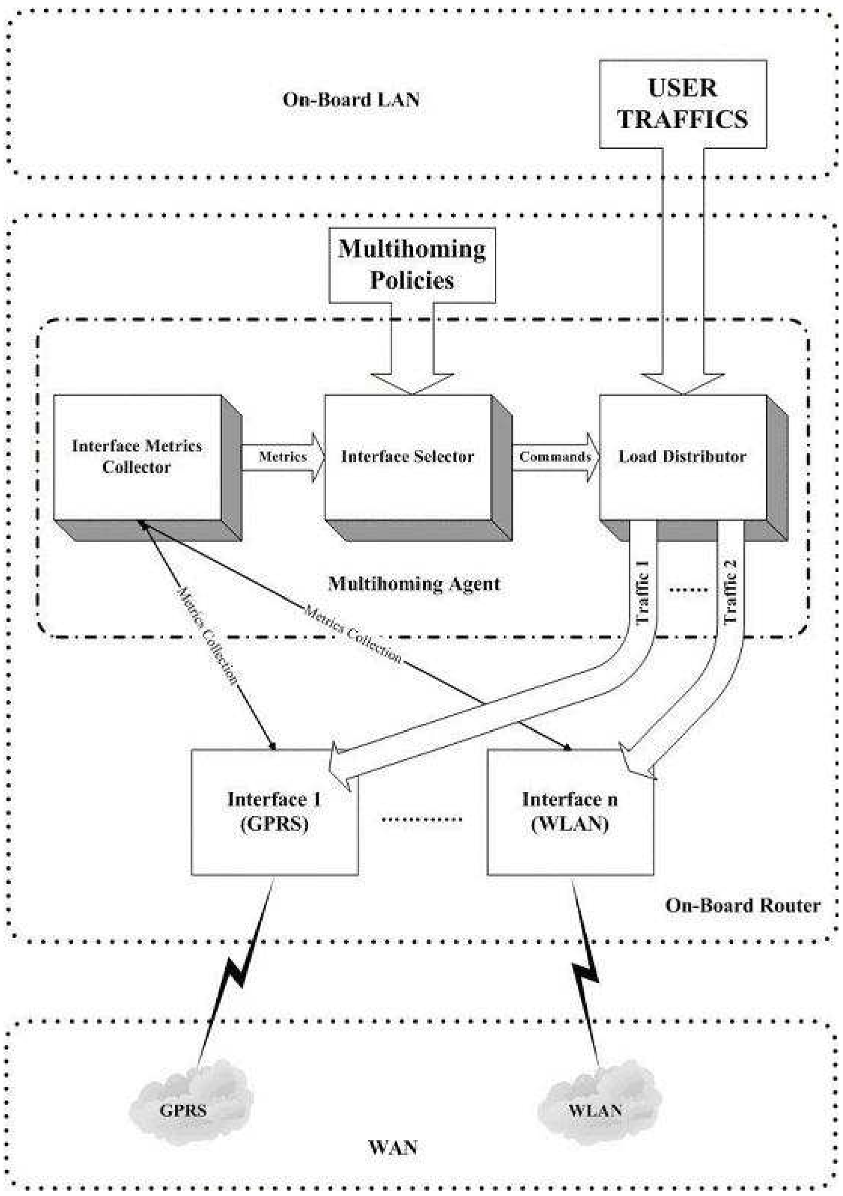
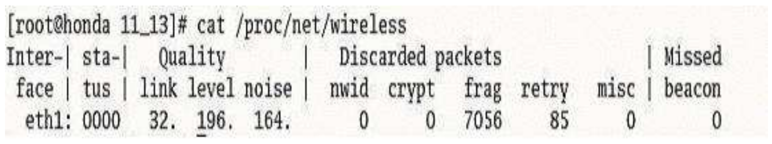


Figure 2: Multihoming Agent Architecture

the signal strength independent any particular driver or WLAN cards. Finally, an Open Source project sponsored by HP was found; this project contributes the Linux Wireless Extension. The Wireless Extension (WE) is a generic API allowing a driver to expose to the user space configuration and statistics to common Wireless LANs. It is really beautiful that a single set of API can support all the variations of WLANs regardless of their type (as long as the drivers support Wireless Extension. Actually, most of the drivers support WE now.) [8]

Moreover, the Wireless Extension is part of the Linux kernel now. (Mostly defined in wireless.h). In the rare cases where the kernel is not compiled with Wireless Extension (/proc/net/wireless non-existent), all needed is just to recompile it with Wireless Extensions (CONFIG_NET_RADIO enabled). Things become easy due to the WE support from Linux kernel and drivers of WLAN cards. All what the work need to do is just retrieve the signal strength, noise level, etc from /proc/net/wireless file, and convert to dBm, the content of /proc/net/wireless looks like Figure 3. The number of records in the wireless files depends on how many WLAN cards are working in current system.



```
[root@honda 11_13]# cat /proc/net/wireless
Inter-| sta-|   Quality        |   Discarded packets               | Missed
 face | tus | link level noise |  nwid  crypt   frag  retry   misc | beacon
  eth1: 0000   32.  196.  164.       0      0   7056     85      0        0
```

Figure 3: Wireless Statistics

4.2 Loss Rate Estimation

Loss rate is an important wide-area network characteristic for evaluating the performance of Internet applications. To estimate the end-to-end loss rate, the most easy and accurate way is deploying specialized software at both the sender and receiver, such as NIMI. Such method can produce highly detailed and accurate results. However, this method requires participation from both the sender and receiver. It is obvious not suitable for this system because estimation about the loss rate between on-board router and the ISP gateway needs to setup any specialized software in the ISP side, which is not feasible at this experimental stage. A tool called Sting [9] is based on the TCP protocol to measure the loss rate between a source host and some target host. It is even able to precisely distinguish which losses occur in the forward direction on the path to the target and which occur in the reverse direction from the target back to the source. The basic idea is that every TCP packet contains a sequence number and acknowledgement number. The sequence number identifies the bytes in each packet so they may be ordered into a reliable data stream. The acknowledgment number is used by the receiver host to identify which packets it has received, and indirectly, which it has not. When in-sequence data is received, the receiver sends an acknowledgement specifying the next sequence number that is expects and implicitly acknowledging all sequence numbers preceding it. Since packets may be lost or reordered in flight, the acknowledgement number is only incremented in response to the arrival of an in-sequence packet. Consequently, out-of-order or lost packets will cause a receiver to issue duplicate acknowledgments for the packet it is expecting. It is really very good to estimate the loss rate and quiet accurate. Unfortunately, this algorithm also requires the target host running at least one TCP-based services. However, this condition could not be ensured, so this idea was abandoned.

Another way to estimate the loss rate could just need participation from one side. One dominant method for measuring loss rate is sending probe packets to a host, and measure loss by observing whether or not response packets arrive within some time period [10]. The loss rate can be derived as:

1 - (packetsreceived / packetssent)

In addition, this method only requires participation from the sender. This is quite suitable for this implementation because it is impossible to run a receiver program in the destination host. This implementation of loss rate estimation is based on this algorithm by using ICMP protocol. It relies on the universal deployment of the ICMP Echo services to coerce response packets from a host. [11] In the real implementation, the raw sockets are used on Linux to send ICMP packets to remote host, and use the process number to identify each packet sent previously, the identity could be used to distinguish the response packet expected from others. Till now, loss rate could be evaluated from the formula above and then it is sent to the Interface Selection Layer by local sockets communication.

4.3 Available Bandwidth Estimation

Available bandwidth is another network characteristic that indicate the network performance directly. The available bandwidth estimated in this project is end-to-end bottleneck bandwidth. i.e. from on-board router to the gateway.

One of the most popular tools about measuring bandwidth is Pathchar. Pathchar takes advantages of the time-to-live (ttl) in an IP packet. The ttl determines how many links a packet can traverse before it expires. If a router receives a packet that has expired, it drops the packet and sends an ICMP error packet back to the sender. Pathchar works by sending out a series of probes with varying values of n and varying packet sizes. For each probe it measures the time until the error packet is received. [12] The bandwidth can be computed by analysis of these measurements. It is obvious that Pathchar is a good tool and quiet accurate. However, Pathchar needs sends a lot of probes and the increase of number of probes is huge with the increase of hops and it may take more than ten minutes to get the result of bandwidth from one end to another. It is not suitable for this project because the interface status is needed from time to time, and the interval should be in seconds level. So, a well-known algorithm called packet pair technique is taken to measure the bandwidth of the path (bottleneck bandwidth) from the dispersion (spacing) experienced by two back-to-back packets [13] [14] [15]. When a packet is transmitted in a link, it encounters a transmission or serialization delay due to the physical bandwidth limitations of the link and hardware constraints of the transmitting equipment. In a link of bandwidth Bi and for a packet of size L, the transmission delay is Ti = L/Bi. A packet pair experiment consists of two packets sent with closely spaced timing (back-to-back), i.e., with a spacing that is as short as possible, from the source to the sink. Without any cross traffic in the path, their inter-arrival time at the receiver directly reflects the bottleneck bandwidth along the path. If also the returning path is uncongested, the corresponding ACKs are received at the sender with the same spacing. The source can thus estimate the bottleneck bandwidth by dividing the length of the sent data packets by the inter-arrival time between the corresponding ACKs.

Tdiff = L / B, so B = L /Tdiff

Here, L is the size of the back-to-back packets and B means the bandwidth, Tdiff is the inter-arrival time between the corresponding ACKs. Figure 4 illustrate such technique.

In the real implementation, the characteristic that a UDP with invalid destination port number is sent to the destination host will result in an ICMP acknowledgement is used. Moreover the corresponding ICMP packets with ICMP port unreachable error. So, a pair of UDP packets with invalid destination port number are sent to the gateway, and the process number is used as the source port. Whenever receiving the packets, by checking whether this packet is ICMP packet and analyzing the IP header to validate the source port and destination port, the expected return packets could be recognized. Furthermore, it also needs to check whether the ICMP type is ICMP_PORT_UNREACH. If all these are true, it can be made sure that this packet is the corresponding acknowledgement from the destination host. Once first back packet arrives, the arrival time is recorded and another response is being expected. The time difference between these two back-to-back UDP packets are calculated, as a result, the bandwidth also could be computed base on the formula given before.

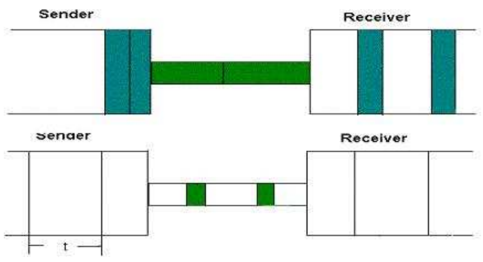


Figure 4: Graphical illustration of the packet pair technique

To make the estimation more accurate and also consider the time issues, ten test samples are taken and the most probability bandwidth would be chosen as the final bandwidth result.

5 Interface Selection Layer

Interface Selection Layer consists of a policy scheduler, which exerts certain policy at certain time, and an interface selector, which allows for dynamic decision-making during the operation of a mobile device.

5.1 Interface Selection Mechanism

The interface selection system is based on three components:

a) Command is a kind of instruction to the load distribution layer where the command is actually executed. Commands specify the priority of interfaces to be used for connections based on evaluation of policies.

b) Policy governs the command. Only one command can take place at a time in a policy. A policy set contains several policies possible defined by different users.

c) Mechanism evaluates policies against connection related information and decides which interface to be used with a specific connection.

5.1.1 Network Information

In wireless networks signal quality and related metrics [1] play an important role when selecting which interface to use. Though other universal factors must be taken into account, link quality is imperative, since it dictates what quality of service demands and policies can be fulfilled and how much of the theoretical bandwidth is actually available. Therefore, Network Information from upper Interfaces Metrics Collection Layer like Signal strength, Loss Rate and Available Bandwidth are interested here.

5.1.2 Policies

Interface selection policies describe the preference of different network interfaces in various situations. The interface selection decisions are based on these preferences and requirements when they are evaluated against gathered information about transport characteristics. The operation of the interface selection mechanism must be continuous as the information may change at any point of time, e.g., a network interface may become unavailable. Therefore, there have to be a policy set and mechanism to hold and maintain rules for interface selection. In this project, six policies were implemented. They can be classified as network-dependent-policy and network-independent-policy.

Network-dependent-policy means the policy needs network data or status such as signal strength, loss rate, and available bandwidth to be evaluated. In another word, the decision based on this kind of policy is made according to the link quality.

However, network-independent-policy means the evaluation of the policy has nothing to do with network status. The purpose of having both network-dependent-policy and network-independent-policy first is to increase the performance of mobile service and second is to compare the performance of mobile service under different policies.

Network-dependent-policy

A. LeastLoaded Policy

It always selects the interface with higher bandwidth. In addition to that, a priority interface queue is given based on the descending order of bandwidth of each interface. Due to the delay, the interface chosen by the mechanism is not guaranteed to an available one. In that case, the interface, which has one level less priority will be chosen alternatively.

B. BestSignal Policy

It always selects the interface with higher signal strength. Due to the same concern, a priority interface queue is given based on the descending order of signal strength of each interface.

C. LeastLoss Policy

It always selects the interface with less data loss rate. Also a priority interface queue is given based on the ascending order of data loss rate of each interface.

D. Weighted Policy

Parameter must be supplied, which means interface information (type, number etc.) is known in advance. It selects particular interface which always has higher priority. For example, (Weighted Interface 1 > Interface 2), this policy always selects interface 1 as long as interface 1 has available bandwidth. If interface 1 has no bandwidth left, it selects interface 2 instead. Once the bandwidth of interface 1 recovers, it selects interface 1 back again.

Network-independent policy

A. RoundRobin Policy

It selects each interface on round robin or randomly without regarding the status of the interface and distributes the load sequentially for load sharing. No parameter is supplied.

B. Specify Policy

Parameter must be supplied. It specifies interfaces to use and ignore the others. If more than one interface is specified, those interfaces will share the load equally. E.g. (Specify Interface 1, Interface 2), 50% of the traffic goes through Interface 1 and the other 50% goes through Interface 2.

Every policy can be applied more than once. It depends on what the requirements are. The policies that are going to be used in each run form a policy set which is stored in a policy configuration file (policy_config.txt). The format of the policy set is

t1 policy1

t2 policy2

...

t1, t2 is the time when policy 1 and poilicy2 should be executed by policy scheduler. A sample of policy set is

0 LeastLoaded

10 BestSignal

20 LeastLoss

30 RoundRobin
40 Weighted 1>2
50 Specify 1,2

After 50 time units, only Specify Policy with parameter 1>2 will be executed until the program is forced to exit.

5.1.3 Updating Network Information

To network-dependent-policy, interface's information is important. Each interface's information (signal strength, available bandwidth, loss rate, interface type, IP address etc.) collected by the Interface Metrics Collection Layer is stored in a structure. All the interfaces' information is stored in a vector. The information is updated whenever new data is received from metrics collecting process.

To give the priority queue of interfaces, the idea is to sort the elements in the vector by the attribute associated with current policy. For instance, if current policy is LeastLoaded, interfaces are sorted by available bandwidth (Interface1 > Interface2 > Interface3). Then the sorted order (Interface1 Interface2 Interface3) will become the command later. But how often should the priority queue be given or how often should the command be sent? It is controlled by another parameter (Frequency) in the policy configuration file (policy_config.txt). So the completed policy configuration file is composed of policy set and frequency. A sample of policy configuration file is

POLICY
0 LeastLoaded
10 BestSignal
20 LeastLoss
30 RoundRobin
40 Weighted 1>2
50 Specify 1,2
FREQUENCY
2

Number '2' at the last line means to calculate priority queue or send command 2 times per time unit. In the program, time unit is set to second.

5.2 Implementation Details

5.2.1 Architecture

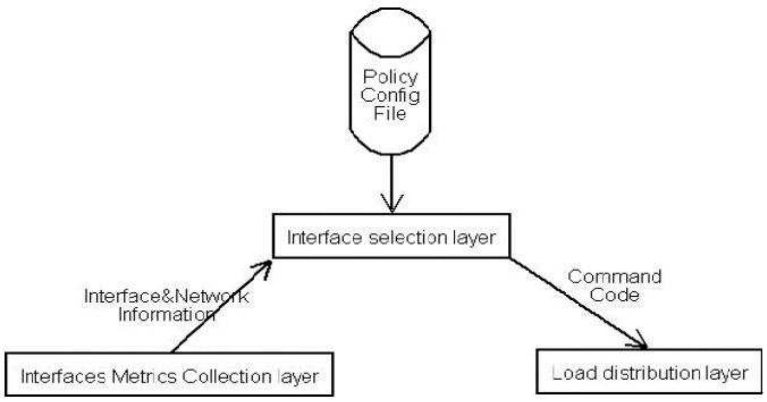

Policy Config File
Interface selection layer
Interface&Network Information
Command Code
Interfaces Metrics Collection layer
Load distribution layer

Figure 5: Interface Selection System Architecture

Basically, interface selection mechanism's input comes from two resource, policy configuration file and metrics collection layer. Policy scheduler is set up according to the information in policy configuration file. The interface information from the Interface Metrics Collection layer helps to make interface selection decision based on current policy. The output of interface selection process is instruction, which is sent to Load Distribution Layer. The frequency of sending instruction is controlled by parameter 'Frequency' in policy configuration file.

5.2.2 Program Flow Chart

Both policy scheduler and interface selector are dead loop. They must run parallel without waiting for one another. To solve the problem, there are two ways, process and thread. Thread is chosen because there is not much communication involved between those two subprograms. As a result, there is little possibility of collision. Moreover, thread is very efficient and easy to implement. So two threads are created from main thread for scheduler and selector respectively.

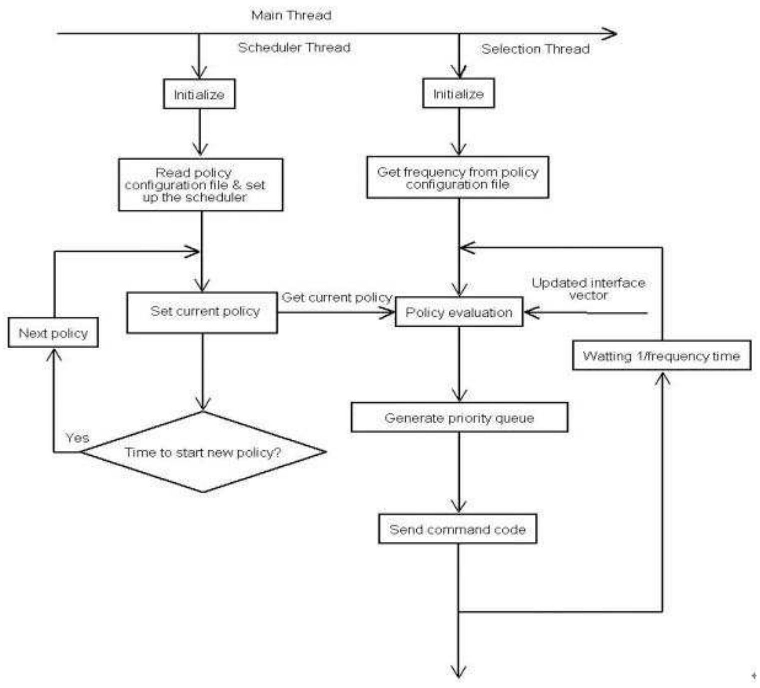


Figure 6: Interface Selection System Program Flow Chart

a) Scheduler Thread

After initialization, scheduler thread first reads from configuration file. It stores policy name, start time and end time of policy as a record into a policy record vector. There is no end time value of last policy record. A pointer is initialized to point the beginning of the policy record vector. When the time hit the end time of a policy, which means the start time of another policy, the pointer will move to point to the next policy record. And current policy will be set to the policy that is pointed by the pointer.

b) Selection Thread

After initialization, this thread first gets the frequency value from policy configuration file. Then, it evaluates current policy with data from updated interface vector, which is updated in other processes and threads. Afterwards, it generates a command and sends it to execution process. Finally, it waits 1/frequency time and does the evaluation again.

c) Communication Between Threads

The variable that stores current policy is set to global because global variable is visible to different threads in the same process. Each time before selection thread dose the evaluation, it will check current policy and do the corresponding evaluation. Each time when a new policy should start, scheduler thread will change current policy variable to the next policy stored in the policy record vector.

6 Load Distribution Layer

A fundamental goal of this project is to dynamically redirect the user traffics to different interfaces, in other words, a robust, reliable and efficient load distribution/ load balancing layer would be expected to be responsible for striping data from Local Network and distributing these traffic into different interfaces. In this Session, the design and implementation of the Load Distribution Layer is fully discussed.

6.1 Data Striping Mechanism Selection

Striping is a tem that has been widely used to describe instances where physical resources have been aggregated together to obtain higher performance [16]. A reliable and robust load distribution system should be built on top of a reasonable and suitable data striping mechanism. Therefore, before looking into any implementation, a striping approach should be determined.

Generally, there are 3 sorts of striping scenario in the context of network traffic striping, namely Connection Striping, Packet Striping and Byte Striping, as shown in Figure 7:

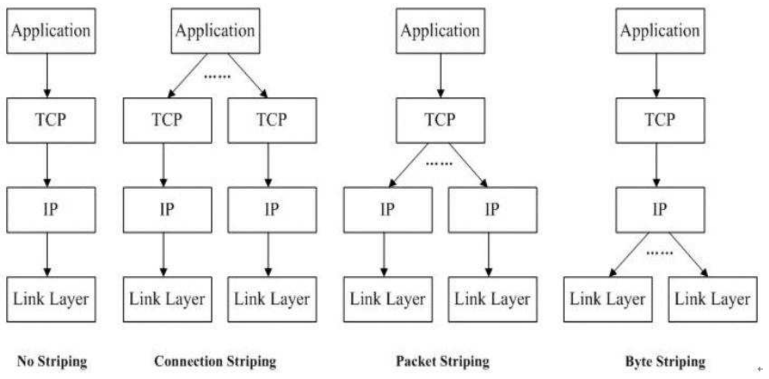


Figure 7: Different Striping Scheme

Byte striping refers to striping data into different underlined physical layers at link layer. It is useful to 1-to-N mapping, when one link layer maps to multiple physical layers. After byte striping, data would be redirected based on bytes, which means data in a single packet could end up into different physical channels. Although higher flexibility and efficiency could be realized by using byte striping, a lot of complicated issues have to be addressed, including byte ordering preservation and byte synchronization etc. Moreover, because each NIC in the prototype being developed has only one physical link, that is, each link layer only maps one underlined physical channel, byte striping doesn't applicable to this implementation obviously.

Aggregated traffics could be striped across multiple connections (a connection here refers to typically TCP connections, or each UDP or ICMP packet when the connectionless protocols are in use), so called connection striping. Based on connection striping, all data belonging to the same connection would travel through the same interface for the duration of that connection. For this sake, the TCP semantics are preserved, and no extra packet synchronization mechanism needed, which brings about the simplicity to the implementation and transparency to the end user. One drawback of connection striping is it could not make use the aggregated bandwidth efficiently. For example, due to the nature of connection striping, it is possible that end-user would end up waiting for the connections that are assigned to the congested channel previously, while other data channels are idle or less congested.

A more reasonable solution for data striping could be packet striping at the network layer, which means redirecting the packets across multiple interfaces regardless their connection id. Hopefully, packet striping could make use of the full capacity of each available interface. As a result, the efficiency issues might be addressed by using packet striping. However, it should be noticed that because packets belonging to the same connection are distributed across multiple interfaces at multihomed prototype, it is possible to result in a serious packet reordering problem due to the variety significant link variety in bandwidth, loss rate and congestion status etc. In other words, the packets travel through the worse quality link would likely be delayed more than those go through the better performance links, which potentially incur more packet reordering events in the networks. Especially in the context of wireless, the difference between the performance of two distinct interfaces could dramatically, say WLAN has more than 5Mbps bandwidth while GPRS has only less than 100Kbps bandwidth, which is 50 times less than the former. [17]. This potential high packet loss rate would definitely do harm to the actual network expected throughput. To address this side-affect, a suitable packet synchronization scheme should be developed first before any implementation.

In this implementation, connection based striping technique is chosen. Although it is possible not the most efficient data striping scheme, it is more straightforward to implement and compatible existing TCP semantics, which it could keep the connection intact and does not require any support from extra ordering preservation and synchronization mechanisms.

6.2 Traffic Redirection Scheme

6.2.1 Transport Layer Approaches

Connection striping could be achieved at either transport layer or network layer. Different transport layer striping approaches could be found at [18], [19], [20], [21], [22]. These approaches developed different new transport layer protocols that support the multihoming/ mobility either or both. Nevertheless, because they are new from original TCP, both sides of end-users' network subsystem have to support the new protocol other than using the existing TCP protocol. Therefore, transport layer approaches are not transparent to end-user, which is definitely not the expected feature of this multihoming prototype.

6.2.2 Network Layer Approaches

When implementing the connection striping at the network layer, 2 popular schemes could be taken into account, that is tunneling and NAT.

A. Tunneling

Tunneling is the transmission of data intended for use only within a private, usually corporate network through a public network in such a way that the routing nodes in the public network are unaware that the transmission is part of a private network. Tunneling is generally done by encapsulating the private network data and protocol information within the public network transmission units so that the private network protocol information appears to the public network as data. Tunneling allows the use of the Internet, which is a public network, to convey data on behalf of a private network, and it is widely used in today's VPN networks, IPv6 implementation on top of IPv4 and Mobile IP etc. However, tunneling basically needs some support from Ingress and Egress nodes in the public networks to perform the encapsulation and de-encapsulation. In this implementation, because all job are done on the on-board router side, which is possible just the ingress node, without the de-encapsulation from the other side of the egress node, tunneling could not be the proper choice for the striping.

B. NAT

NAT (Network Address Translation) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. Typically, a company maps its local inside network addresses to one or more global outside IP addresses and unmaps the global IP addresses on incoming packets back into local IP addresses. NAT originally helps ensure security since each outgoing or incoming request must go through a translation process that also offers the opportunity to qualify or authenticate the request or match it to a previous request. NAT also conserves on the number of global IP addresses that a company needs and it lets the company use a single IP address in its communication with the world.

In this implementation, because the multihoming agent is sitting inside the on-board router, which serves the on-broad LAN, it is quite naturally to use NAT to translate the addresses from on-board LAN to the public addresses on-board router is using. For this sake, with some specific NAT mechanism, striping could be realized by NATing the source private addresses to the proper outgoing IP addresses of the on-board router. By using NAT, once a given interface is selected by the Interface Selection Layer, all on-board LAN originated packets are source-NATed (both IP address and Port number) by this Load Distribution Layer using the new port number and IP address of the selected WAN interface (i.e. GPRS or WLAN). When incoming traffics arrive at the WAN interface, which the destination IP address belongs to, this Load Distribution Layer would look up the previous source-NAT details, and then decide and perform a proper de-NAT. Therefore, the traffics could be directed to the right client in the on-board LAN by using the de-NATed IP address and port number.

6.3 Implementation Details

In order to support the interaction between Interface Selection and the Load Distribution Layer to perform an expected NAT/de-NAT function to the user traffic. 2 main components are introduced into the Load Distribution Layer, which are the NAT module in the Linux kernel and Instruction Processor at the Application Layer.

6.3.1 NAT Module

In this section, the implementation details of the NAT module of Multihoming Agent are fully discussed. This proposed module consists of 2 main functionalities, including NAT/DNAT and Failure Recovery.

A. Netfilter Framework Introduction

Since 2.4.x, Netfilter Framework was introduced into the Linux Kernel. Netfilter is the framework in Linux 2.4 kernels that allow for firewalling, NAT, and packet mangling. It is quite interesting this Linux Kernel extension has provide some sort of support to the NAT functions, which facilitates the implementation a lot. Therefore, it is natural to start the work with the Netfilter Framework.

Netfilter is a framework for packet mangling, outside the normal Berkeley socket interface. Generally, it has 3 parts. [23]

Firstly, each protocol defines "hooks" (IPv4 defines 5) which are well-defined points in a packet's traversal of that protocol stack. At each of these points, the protocol will call the Netfilter framework with the packet and the hook number.

Secondly, parts of the kernel can register to listen to the different hooks for each protocol. So when a packet is passed to the Netfilter framework, it checks to see if anyone has registered for that protocol and hook; if so, they each get a chance to examine (and possibly alter) the packet in order, then discard the packet (NF_DROP), allow it to pass (NF_ACCEPT), tell Netfilter to forget about the packet (NF_STOLEN), or ask Netfilter to queue the packet for userspace (NF_QUEUE).

The third part is that packets that have been queued are collected (by the ip_queue driver) for sending to userspace; these packets are handled asynchronously.

In addition, various modules have been written which provide functionality like the extensible NAT system, and the extensible packet filtering system. The whole architecture of Netfilter is shown in Figure 8.

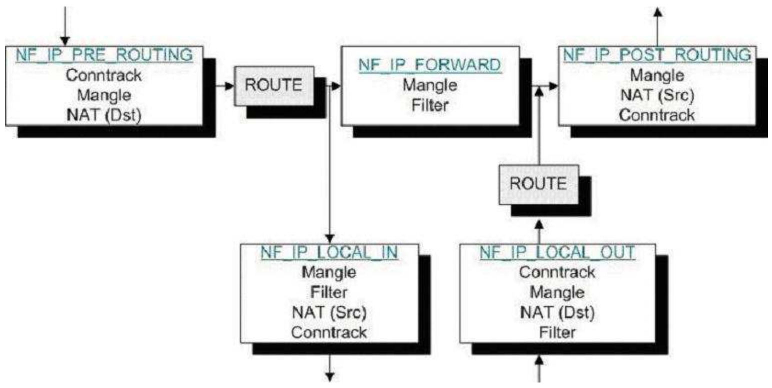


Figure 8: Netfilter Framwork

There are two functionality in this framework should be noticed in this project, namely Conntrack and NAT(both Src and Dst).

Conntrack (Connection tracking) refers to the ability to maintain state information about a connection in memory tables, such as source and destination ip address and port number pairs (known as socket pairs), protocol types, connection state and timeouts.

Conntrack states are as follows:

a. INVALID Packet is associated with unknown connection

b. ESTABLISHED Packet is associated with a connection which has seen packets in both directions

c. NEW Packet has started a new connection, or otherwise associated with a connection which has not seen packets in both directions

d. RELATED Packet is starting a new connection, but is associated with an existing connection, such as an FTP data transfer, or an ICMP error

As seen from Figure 8, Connection tracking is done either at the NF_IP_PREROUTING hook, or the NF_IP_LOCAL_OUT hook for locally generated packets. So, by using the state information provided by the Conntrack Module, different NATs could be done at following hook points:

1) NF_IP_PRE_ROUTING & NF_IP_LOCAL_OUT - DNAT the packet.

2) NF_IP_POST_ROUTING & NF_IP_LOCAL_IN - SNAT the packet.

Based on this sophisticated system, the proposed Load Distribution system could be achieved via attaching the proper source-NAT and destination-NAT information on each packet in order to let the source-NAT module to source-NAT for this proposed agent.

B. NAT/DNAT

As connection striping is in use in this prototype, only the packet marked as state "NEW" should be interested by the source NAT function. That is, after checking the connection state via the information provided by Conntrack, whenever a new connection request from end-user reaches at this on-board router, Load Distribution Layer would set the NAT information to the sk_buff structure, which represents a packet in Linux Kernel. After that, when the packet hooked by the source-NAT module of Netfilter, the latter would source NAT this packet according to the NAT information set already. Because of the nature of the state, the NAT information set to the packet is also set to the connection the packet belongs to, so that, the following packet from the same connection would be source-NAT automatically. Correspondingly, without any help, by correctly souce-NAT the out-going packets, Netfilter would be responsible for the D-NAT automatically.

The following graph is the flow chart of NAT information setting for the outgoing traffics.

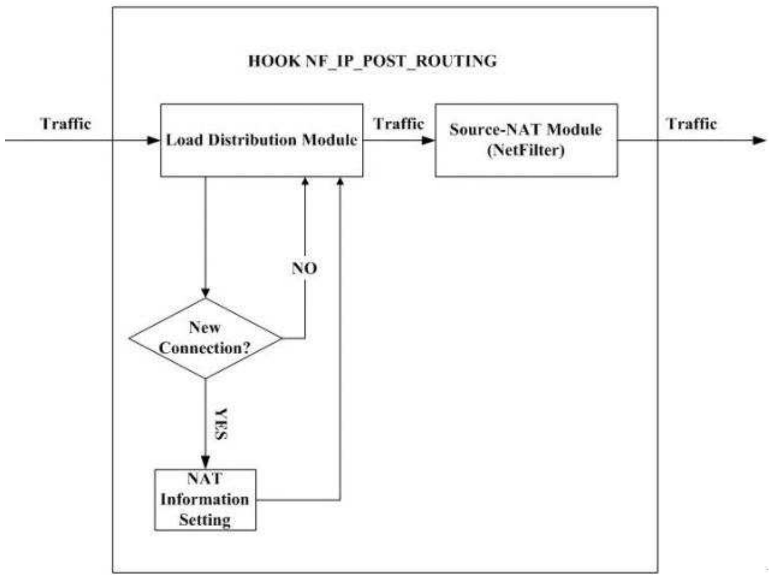


Figure 9: NAT Information Setting

C. Failure Recovery

The NAT module above provides a basic load distribution/ load sharing function to the multihoming agent. Nevertheless, the idea of multihoming is not only enhance the bandwidth utilization but also the ealiability and robustness. Therefore, an extra failure recovery funtionality should be built on top this prototype to redirect the traffic allocated to the unavailable link to the available links, in such a way, to provide a redundency failover feature.

Because the NAT machanism is in use to select the outgoing WAN interface of the on-board, this kind of failover could be achieve by reNAT the originally NATed packet from the IP address of unavailable WAN interface to a corrently available WAN interface chosen by the Interface Selection Layer or the Instruction Processor mentioned later on.

By reNAT the packets, the user traffic could survive through other available interfaces, such as ICMP and UDP packets, although there are serveral packets lost due to the interface failover decision latency.

However, because of using NAT, just reNAT the IP address of packets could not help the connections using TCP protocol to survive. This is because although the packet could go through via new path, the TCP connection would be regarded as still broken at the recieving side due to the source IP address of the packet could not match the original connection' s source IP address. To address this, some extra proxy server (such as the HA of mobile IP) should be developed in the middle to NAT each packet again, providing a consistent outgoing source IP address to trick the end reciever on the other TCP connection side.

In addition, this implementation is realized base on the Netfilter Framwork, the Conntrack function of Netfilter Framwork is the precondition of the NAT function. However, because the definition of a tracked connection is identified by source IP address, dstination IP address, session ID (for ICMP) or Source Port(UDP or TCP) and Destination Port(UDP or TCP), and each entry in the Conntrack table could be used to specify the NAT information which contains the Source-NAT Source IP address for the client-originated packets and the Destionation-NAT Destination IP for the client-desitined packets. When a failover happen, even after changing the NAT source address, the identifier of this new packet would be the same as the old ones, so this connection would not be inserted into the Conntrack table, which means this newly NAT information could not be retrived of this connection. As a result, the the implementation could not make the inprogress data sessions which using connectionless protocol such as UDP and ICMP timely, because even the packets were NATed properly, and the response from destination can arrive at the multihoming router, these response packets would be dropped due to the unexistance in the Conntrack table. Actually at this moment, all other new ICMP or UDP connections with new destination information are working fine, and after 30 seconds, which means the old fail connections are timeout in the Conntrack table, those connections would recover and work again.

The problem mentioned above could be resolved by deleting the fail entries in the Conntrack table. This is implemented in this project but it still needs to improve the robustness. Another better solucion for this is bypassing the Conntrack functionality provided by Netfilter and implementing some flexible new connection tracking mechanism. This solution could benefit the program by giving the control and whole picture of all inprogress connections rather than depend on the information in the sk_buff structure of each incoming packet which reflects the current information in the Conntrack table. However, because NAT is based on Conntrack, without using Conntrack the Netfilter Framework does not have too much meaning on this project, which means everything would have to be done from scratch.

6.4 Instruction Processor

Instruction Processor is the User Space component of the Load Distribution Layer, which is in charge of receiving the instruction from Interface Selection Layer, probing the availability of interfaces and issuing the proper NAT instruction to Kernel NAT module discussed above.

6.4.1 User Space/Kernel Space Communication

Because the Multihoming agent is a deamon resides in the application layer, which is the User Space of Linux, the NAT module is a Kernel Space component. The communication between them is one of the issues need to be addressed here. In this implementation, the communication between User Space and Kernel Space is done via Netlink Sockets.

Netlink soket is used to transfer information between kernel modules and user space processes, it provides kernel/user space bidirectional communication links. It consists of a standard sockets based interface for user processes and an internal kernel API for kernel modules. Full detail about Netlink Socket could be referenced from [24].

6.4.2 Interface Failure Detection

Besides listening the commands to choose the interface, Instrution Processor has its own Failure Detection function, which monitoring the interfaces in use. Once any failure or outage happens at any used WAN interface (say AP is down in WLAN), this detection component would response by issuing the failover commands to Kernel NAT module.

The detection is realized by using the ioctl() API provided by Linux. This API provides a gateway to retrieve a serious of current settings and status of each devices, including the WAN NICs.

The prototype retrieves the current status of interfaces through ioctl() API from time to time, including those of Ethernet and WLAN. Unfortunately, due to the equiment issues, GPRS NIC probing is not realized in this project, which could be a part of future work issue.

Whenever there is a packet to send through the NAT module, the latter would send a notice to the Instruction Processor, asking the processor to probe the interface going to be used. If the Instruction Processor finds anything wrong with the requested interface, it would issue a notifying message to NAT module via Netlink socket. When the NAT module recieves this command, it would follow the command to do failover and put the fail interface into a linklist, which contains the fail interfaces and indicates the NAT module to ignore the using towards them.

If later on, the Instruction Processor finds the interface is back, it would send an recovery command to NAT module; if NAT module also confirms the involved interface is in the failover linklist indeed, it would simply delete the entry in the failure linklist and perform any regular operations on it. In such a way, the interface recover from the failure.

7 Experiment Results

7.1 Testbed setup

This Multihoming Agent is implemented on the Linux machine with a Pentium IV 1.4GHz CPU and 256 MB memory, which equipped with 3 NICs, 2 Ethernet NICs and 1 802.11b NIC. The GPRS NIC is replaced with an Ethernet due to the equipment shortage. Actually, except the Signal Strength Collection, other functionalities are all identical regardless what kind of NIC is in use in this prototype. Moreover, the Signal Strength Collection function for GPRS has not been implemented successfully indeed in this project due to the same reason mentioned above, so the Ethernet NIC is used to replace that of GPRS. Figure 10 shows the details of the Testbed.

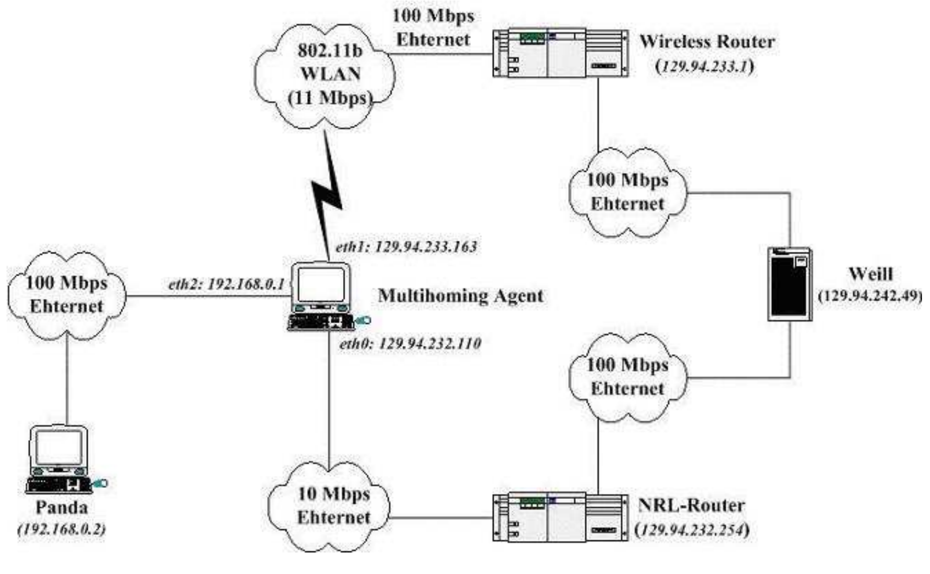


Figure 10: Testbed architecture

In practice, the effectiveness of the Interface Metrics Collection Layer and Interface Selection Layer is verified quite well, the following tests would focus on the total performance enhancement of this prototype based on the Testbed established above, which evaluating the effectiveness of traffic striping through Load Balancing and Failure Recovery tests.

7.2 Bandwidth Estimation

Under the normal condition, the Interface Metrics Collection Layer could produce the approximate interface metrics to the Interface Selection Layer, However, Figure 11 displays the exception in the bandwidth estimation part. This pie chart illustrates the inaccurate feature of the bandwidth estimation, from which can get the proportion of abnormal bandwidth and normal bandwidth in congested situation. (10Mbps traffic through the Multihoming Agent)

In some cases, especially when large traffic data transferring happens, the time difference between two returning back-to-back packets becomes very small. Sometimes even 4 or 5 us. The bandwidth will become huge according to the formula packetSize/Tdiff. The problems are found after insight to this issue. The time difference is as normal as before in MAC layer, however, when the packets go up from MAC layer to Application layer, the time difference decreases rapidly to an unbelievable small value.

Now, the truth is discovered, the change of time difference is caused by the Linux operating system when huge data traffic passed through this agent. Extra effort on Linux kernel is needed to solve this problem in the next stage of this project.

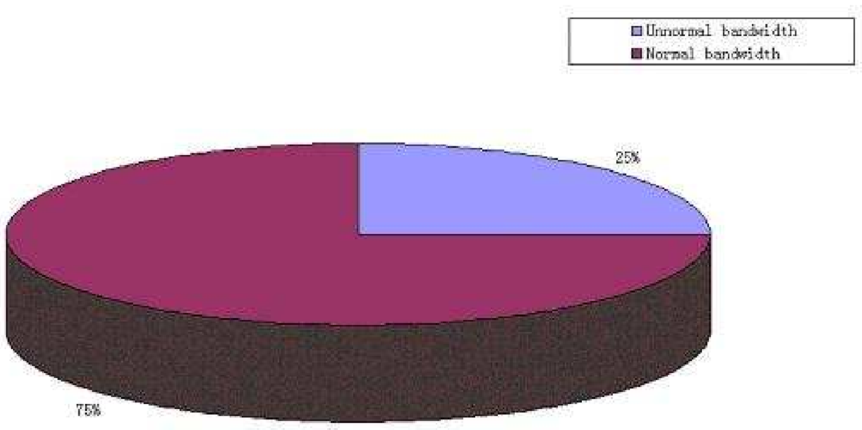


Figure 11: Bandwidth Estimation in Congested Situation

7.3 TCP Performance

7.3.1 Instantaneous Bandwidth

Figure 12shows the Instantaneous Bandwidth of the Multihoming Router. The data was collected by sending continueous TCP data on both Eth0 and Eth1 in the duration of 60 second. From Figure 12, it could be observed that with the sorted bandwidth of rough 7Mbps (in blue) on Eth0 and 4 Mbps (in green) on Eth1, a total aggregated bandwidth of 11Mbps could be achieved (in red).

Figure 13 and Figure 14 are TCP traces on Eth0 and Eth1 respectively. In Figure 14, the accumulative received data are increased nearly linear, while the data line of Eth0 in Figure 13 has an unstable slope. Correspondingly, the unstable slope in Figure 14 could be realized by the fluctuated instantaneous bandwidth of Eth0 in compare of the horizontal-line-like curve of instantaneous bandwidth of Eth1.

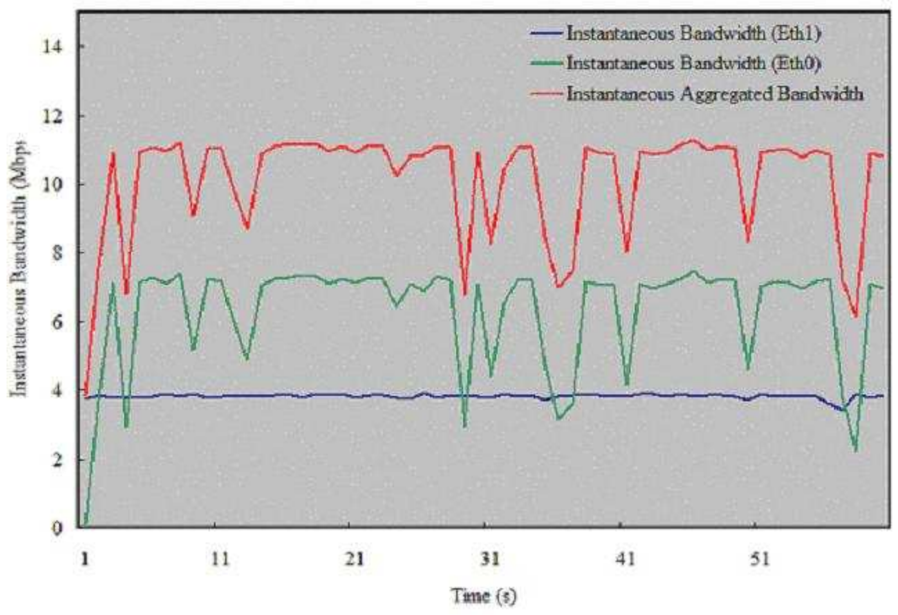


Figure 12: Instantaneous Bandwidth of Multihoming Router (TCP)

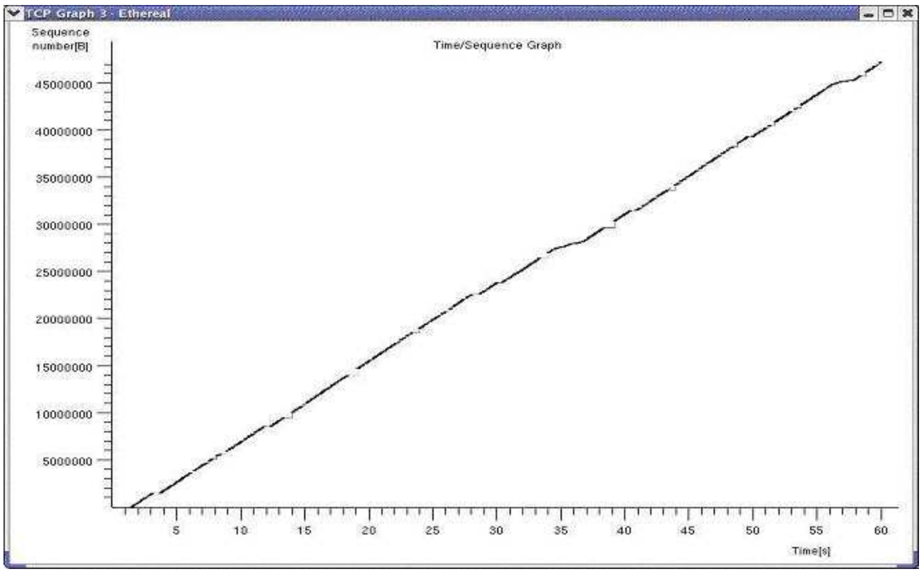


Figure 13: Instantaneous Bandwidth of Multihoming Router (TCP)

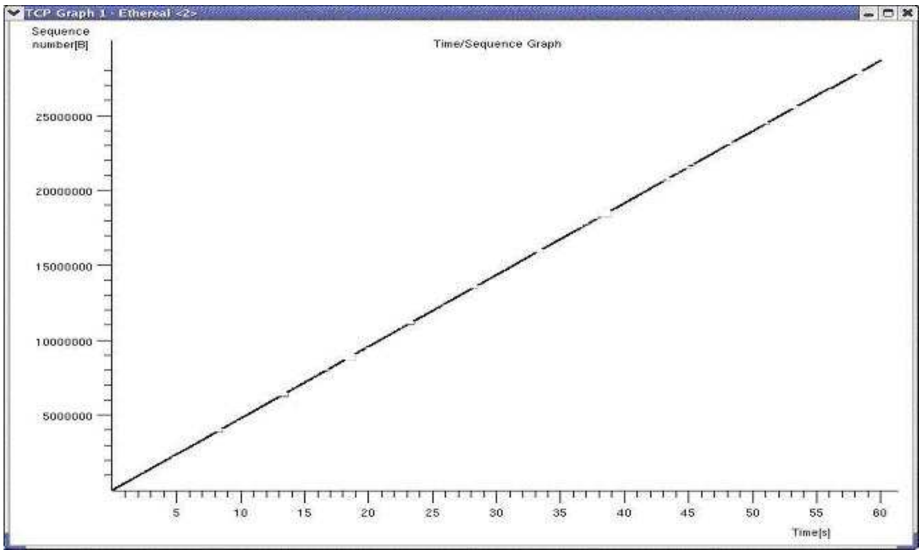


Figure 14: Instantaneous Bandwidth of Multihoming Router (TCP)

7.3.2 Efficiency

The following Figure 15 was built based on the data collected in a fashion of measuring the total elapsed time when transferring 20MB data through each one of the interface and that from using both interfaces to carry TCP data.

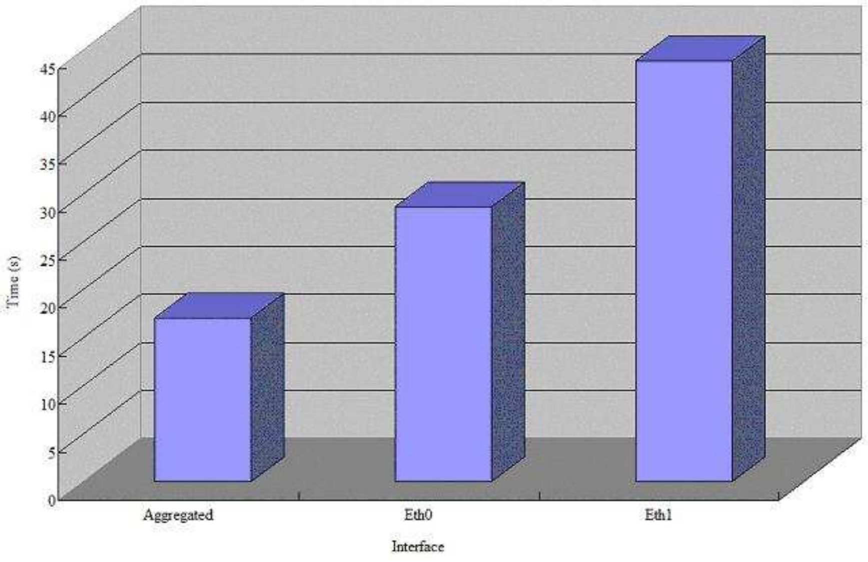


Figure 15: Elapsed time when transferring 20M data using TCP

From these three bars, it could be concluded that the Multihoming Router could efficiently transfer 20MB data in only 15 seconds, which has a 45% and 65% of performance increasing on using Eth0 and Eth1 only respectively.

7.4 UDP Performance

7.4.1 Instantaneous Bandwidth

Figure 16 shows the UDP Instantaneous Bandwidth of the Multihoming Router. The data was collected by sending continueous UDP data on both Eth0 and Eth1 in the duration of 60 second. From Figure 16, it could be observed that with the sorted bandwidth of rough 7Mbps (in green) on Eth0 and 3.8 Mbps (in blue) on Eth1, a total aggregated bandwidth of 10.7 Mbps could be achieved (in red).

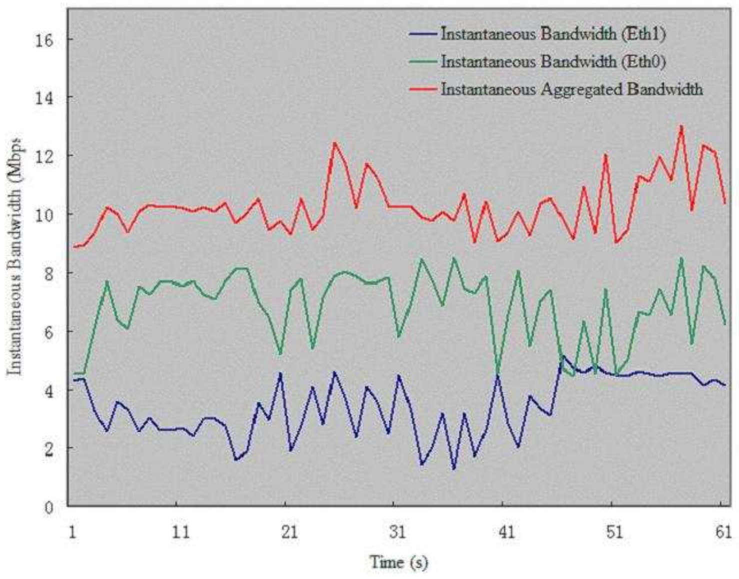


Figure 16: Instantaneous Bandwidth of Multihoming Router (UDP)

Unlike data collected by TCP protocol above, data in this graph fluctuate a lot regardless the originating interface. However, it still reflects the effectiveness on increasing the aggregated bandwidth on this multihoming system.

7.4.2 Efficiency

Two graphs (Figure 17 and Figure 18) below are illustrating the efficiency by using this system to transfer UDP data.

Figure 17 shows the effectiveness by the data collected from sending continueous UDP data during the course of 20 seconds. From the 8 sets data collected, it could be observed that this system has an ability to relay more than 21MB data by using data striping through 2 available interfaces.

In contrast, as could be seen from Figure 18, by using single interface to transfer data, a router could only deal with 3MB data (using Eth1) and 18MB data (using Eth0). This implies the efficitiveness of the bandwidth aggregation and load balancing brought by the proposed multihoming prototype.

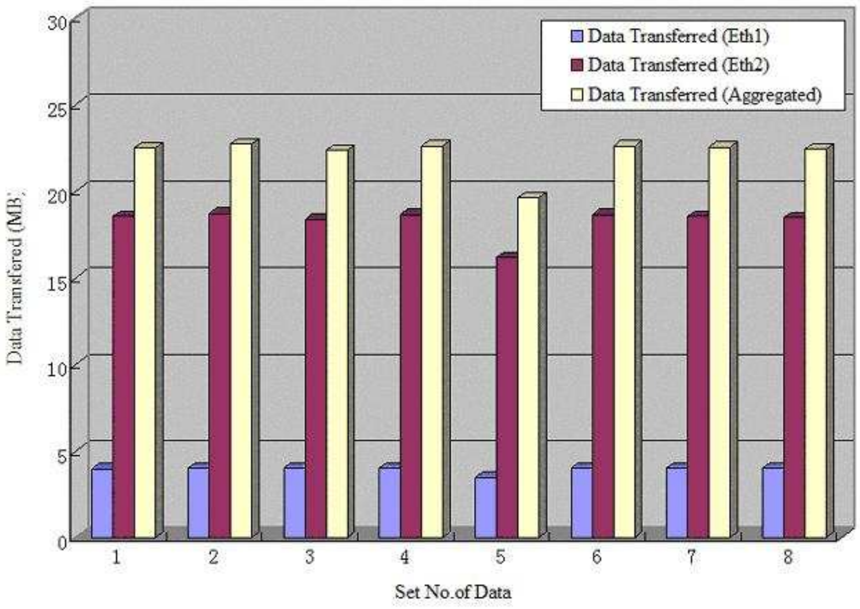


Figure 17: UDP aggregated

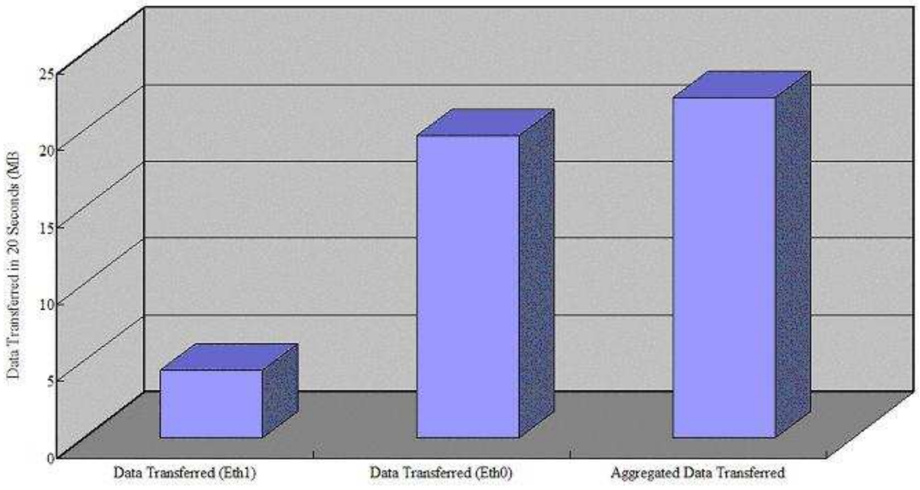


Figure 18: UDP data transferred in 20 Seconds

7.4.3 Comparison with TCP

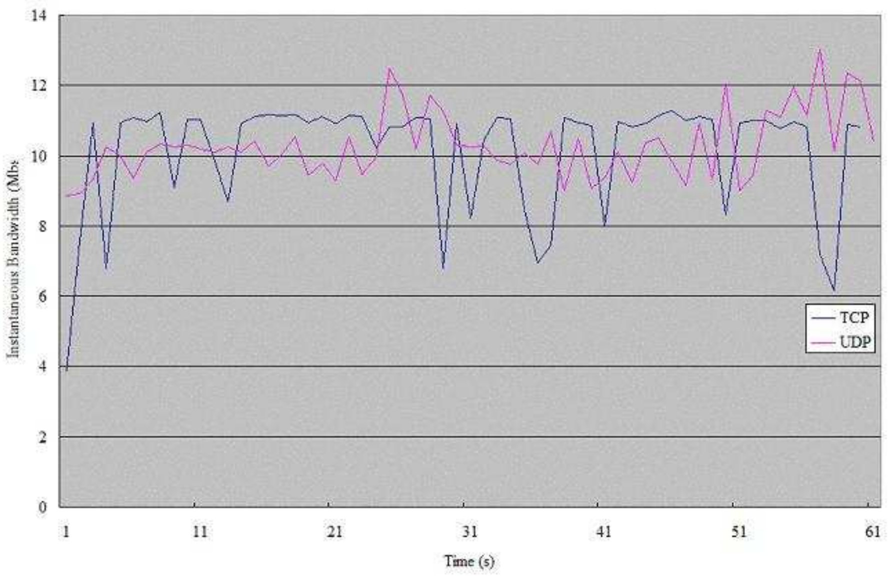


Figure 19: Comparion with TCP

Figure 19 shows the UDP and TCP instantaneous bandwidth of the Multihoming router. The data is collected by sending UDP or TCP data in 60 seconds, assuming that the network condition remains the same during UDP data collection and TCP data collection. Theoretically, UDP should have better performance. But based on the experiment data, there is not much difference between UDP and TCP performance due to immature data distribution method. This can be improved in later work.

7.5 Http Performance

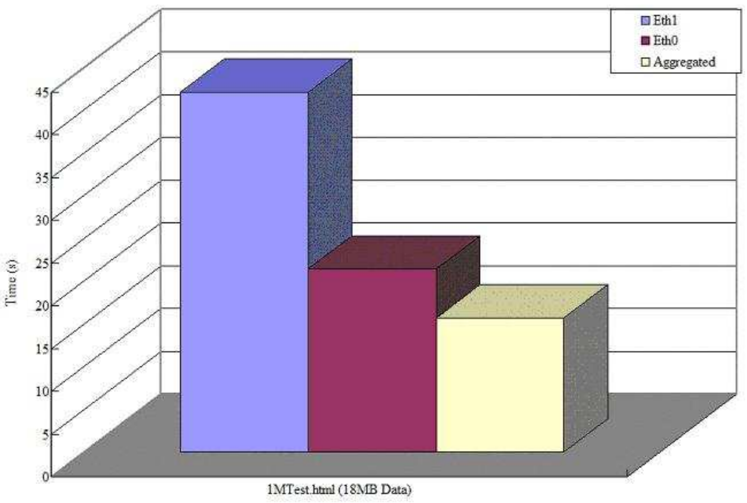


Figure 20: HTTP Performance

Figure 20 is used to evaluate the web performance of the prototype on real traffics in contrast with the previous results, which are collected based on the traffics generated by the traffic generator software. The web page visited on weill is called 1MTest.html. The 1Mtest.html was created by 18 photo objects, which has the size of 1MB, so the total traffic caused by this page should be roughly 18MB. The client machine Panda generated HTTP requests on these two pages respectively through the Multihoming Agent. The final results are quite encouraging, which implies the total HTTP response time could be significantly reduced by using this multihoming prototype. It is shown that the total response time of downloading 18M data would be reduced by 28% and 63% compared with that of using Eth0 and Eth1 respectively. All these results are generally consistent with those got in Section 7.3.2.

7.6 Effectiveness of Failover Recovery

The recovery functionality was achieved in this project in the Load Distribution Layer, the following 2 snapshots shows the real effect of failover functionality.

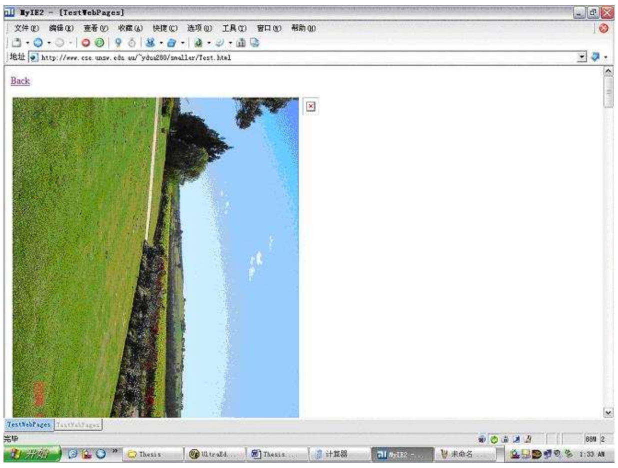
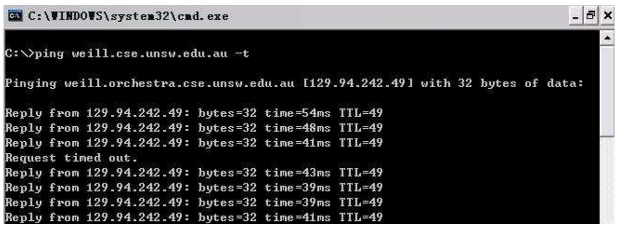


Figure 21: Failover Snapshot

In Figure 21, there should be 2 side-by-side photos on the web page origanally, but a interface failure happened (cable pulled off or signal outage) when downloading this test page (downloading right hand side one, before downloading the left one). As a result, the connection of the photo on right hand side was reset due to the src-NAT address replacing. (This could be improved by introducing some proxy server as mentioned before to maintain this kind of fail connections) However, thanks to the failover mechanism in use, the photo on the left hand side was transferred thoroughly through a new chosen interface.



```
C:\WINDOWS\system32\cmd.exe

C:\>ping weill.cse.unsw.edu.au -t

Pinging weill.orchestra.cse.unsw.edu.au [129.94.242.49] with 32 bytes of data:

Reply from 129.94.242.49: bytes=32 time=54ms TTL=49
Reply from 129.94.242.49: bytes=32 time=48ms TTL=49
Reply from 129.94.242.49: bytes=32 time=41ms TTL=49
Request timed out.
Reply from 129.94.242.49: bytes=32 time=43ms TTL=49
Reply from 129.94.242.49: bytes=32 time=39ms TTL=49
Reply from 129.94.242.49: bytes=32 time=39ms TTL=49
Reply from 129.94.242.49: bytes=32 time=41ms TTL=49
```

Figure 22: Failover Snapshot (ICMP)

Figure 22 shows the failover functionality works well on ICMP (of course UDP as well). As could be discovered, during the failover procedure, only 3 ICMP packets was lost, and system keep working fine afterwards.

It should be mentioned that the failover latency of UDP or ICMP is not mainly determined by the failure detection function. It is affected by the latency to delete the old failed entries in the Conntrack table, fot more detail and possible improvement, it could be referred to Part C of Section 6.3.1.

8 Conclusion and Future Works

This project develops and implements a multihoming agent on a Linux based platform to support the future on-board multihomed routers. This multihoming agent allows router to access and switch between multiple access links including WLAN and GPRS. The prototype would be capable of dynamically switching users' data traffic from one network interface to another, i.e. Ethernet, WLAN, and GPRS, without breaking transport layer sessions, based on the interfaces' status and policy in use.

By correctly collecting metrics from Interface Metrics Collection Layer, Interface Selection Layer would process the raw metrics and generate commands to the Load Distribution Layer base on the Policy running. This architecture brings the intelligent to the on-board routing system and brings about at least 50% performance enhancement and failure recovery features to the on-board LAN end users.

Although a preliminary success has been achieved, more future works are expected.

In the Interface Metrics Collection Layer, the accuracy and efficiency should be improved. In addition, signal strength of GPRS is not implemented in current project and this should be addressed in the future.

As to Interface Selection Layer, when defining the policy, no application type is considered. Future work should add or modify policies taking application type into account as well. Also future work could consider policies towards handoff. For example, there is signal strength threshold for WLAN and GPRS. Decision-making should consider the threshold instead of simply comparing the signal strength of different interfaces.

In the Load Distribution Layer, more reasonable and efficient data striping mechanism should be developed and implemented to avoid the unfair data distribution based on connection striping and the packet reordering in the packet striping mechanism. Of course, the robustness and latency should be improved while any failure happen to interfaces in the future.

Finally, this prototype might be implemented in other platform such as Windows and Solaris in the future.

References

[1] Jukka Ylitalo, Tony Jokikyyny, Tero Kauppinen, Antti J. Tuominen, and Jaako Laine. Dynamic network interface selection in multihomed mobile hosts. In *Proceedings of the 36th Hawaii International Conference on System Sciences*, 2002.

[2] H. Berkowitz and D. Krioukov. To be multihomed: Requirements and definitions. *Internet Draft*, July 2001.

[3] C. Huitema and R. Draves. Host-centric ipv6 multihoming. *Internet Draft*, July 2001.

[4] J. Abley, B.Black, and V. Gill. Ipv4 multihoming motivation, practices and limitations. *Internet Draft*, June 2001.

[5] B. Black, V. Gill, and J. Abley. Requirements for ipv6 site-multihoming architectures. *Internet Draft*, November 2001.

[6] M. Ylianttila, R. Pichna, J. Vallstrom, J. Makela, A. Zahedt, P. Krishnamurthy, and K. Pahhlavan. Handoff procedure for heterogeneous wireless networks. In *Global Telecommunications Conference - Globecom '99*, pages 2793–2787, December 1999.

[7] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. Mar: A commuter router infrastructure for the moble internet. In *Proceedings of ACM MobiSYS*, 2004.

[8] Wireless tools for linux.
http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html.

[9] S.Stefan. Sting: a tcp-based netowrk measurement tool. In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, October 1999.

[10] J.Postel. Internet control message protocol. *RFC-792*, September 1981.

[11] R.Braden. Requirements for internet hosts-communication layers. *RFC-1122*, October 1989.

[12] V. Pathchar Jacobson. A tool to infer characteristics of internet paths.
ftp://ftp.ee.lbl.gov/pathchar/.

[13] R.L. Carter and M.E.Crovella. Measuring bottleneck link speed in packet-switched network. *Performance Evaluation*, 27:297–318, 1996.

[14] V.Paxson. End-to-end internet packet dynamic. *IEEE/ACM Transaction on Networking*, 7(3):277–292, June 1999.

[15] K.Lai and M.Baker. Measuring bandwidth. In *Proceedings of IEEE INFOCOM*, April 1999.

[16] C. B. Traw and J. Smith. Striping within the network subsystem. *IEEE Network Magazine*, 9(4):22–32, July 1995.

[17] Colin Perkins Ladan Gharai and Tom Lehman. Packet reordering, high speed networks and transport protocol performance. In *Proceedings of ICCCN 04*, 2004.

[18] Mark Handley Eddie Kohler and Sally Floyd. Datagram congestion control protocol (dccp). *Internet Draft*, February 2004.

[19] L. Ong and J. Yoakum. An introduction to the stream control transmission protocol (sctp). *RFC-3286*, May 2002.

[20] H. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multihomed mobile hosts. In *IEEE MobiCom*, pages 35–46, Atlanta, USA, September 2002.

[21] D. S. Phatak and T. Goff. A novel mechanism for data streaming across multiple ip links for improving throughput and reliability in mobile environments. *IEEE INFOCOM*, 2:773–781, June 2002.

[22] Tom Goff and Dhananjay Phatak. Unified transport layer support for data striping and host mobility. *IEEE Journal of Selected Areas in Communications, Special Issue on Wireless IP networks*, 2004.

[23] Rusty Russell and Harald Welte. Netfilter hacking howto.
http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO.html.

[24] Gowri Dhandapani and Anupama Sundaresan. Netlink sockets overview. *http://qos.ittc.ukans.edu/netlink/html*.